

Learning Event-Based Temporal Abstractions for Hierarchical Prediction and Planning

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Christian Gumbsch
aus Stuttgart

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der
Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	17.07.2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. Martin Butz
2. Berichterstatter/-in:	Prof. Dr. Justus Piater
3. Berichterstatter/-in:	Prof. Dr. Olivier Sigaud

ABSTRACT

Over the last decade, deep reinforcement learning systems have made remarkable progress in various domains, partially reaching superhuman performance when trained extensively. However, no artificial system has yet reached the flexibility and efficiency with which intelligent animals learn to solve novel problems. The goal of this thesis is to close this gap to some extent by taking inspiration from biological cognition to enhance the goal-directed behavior of artificial agents, in particular through the ability to hierarchically decompose sensorimotor experience into events.

The central hypothesis of this work is that model-based temporal abstractions of events not only play a crucial role in human behavior but by learning such structures artificial agents can also acquire more adaptive, far-reaching, goal-directed behavior. To investigate this, a formal framework for the event-based learning of hierarchical models with nested time scales is introduced that is suitable for both computational cognitive modeling and sequential decision making.

First, the cognitive plausibility of the approach is investigated by modeling human anticipatory behavior. In these modeling experiments, an agent is equipped with a pre-structured model of event-based abstractions. When the agent selects its gaze to minimize uncertainty across the hierarchical predictions of its model, goal-anticipatory gaze behavior develops similarly to the eye fixation behavior in infants.

Learning hierarchical predictions requires a mechanism that suitably decomposes activity into events. For this purpose, a recurrent neural network is introduced next, which learns in a self-supervised way to compress dynamics into latent states that are sparsely updated over time. Including this mechanism in different prediction and planning systems improves their generalization abilities, their sample efficiency, and the explainability of the learned representations.

Finally, components of the previous methods are combined to learn a hierarchy of world models from scratch. The high-level model in the hierarchy is only trained based on sparse latent state changes of a low-level dynamics model. When the system selects its gaze focus based on the hierarchical predictions, goal-anticipatory gaze behavior emerges similarly to how it develops in infants during their first year of life. Furthermore, the learned hierarchical predictions can be seamlessly integrated into model-based reinforcement learning and planning agents to improve their performance in challenging problems with long task horizons.

Taken together, this thesis not only provides practical methods for learning event-based temporal abstractions, but also demonstrates how such structures can explain human behavior and enhance sequential decision making in artificial agents.

KURZFASSUNG

In den letzten Jahrzehnten haben Systeme mit tiefem Verstärkungslernen bemerkenswerte Fortschritte erzielt und erreichen nach ausgiebigem Training teilweise übermenschliche Leistungen. Allerdings hat noch kein künstliches System die Flexibilität und Effizienz erreicht, mit der intelligente Tiere lernen neue Probleme zu lösen. Ziel dieser Arbeit ist es, diese Lücke ein Stück weit zu schließen, indem Inspirationen von der biologischer Kognition genommen werden, um das zielgerichtete Verhalten von künstlichen Agenten zu erweitern, insbesondere um die Fähigkeit, sensorimotorische Erfahrungen hierarchisch in Ereignisse zu zerlegen.

Die zentrale Hypothese dieser Arbeit lautet, dass modellbasierte zeitliche Abstraktionen von Ereignissen nicht nur eine entscheidende Rolle im menschlichen Verhalten spielen, sondern dass künstliche Agenten durch das Erlernen solcher Strukturen auch adaptiveres und weitreichenderes zielgerichtetes Verhalten erlangen können. Um das zu untersuchen, wird ein formeller Rahmen für das ereignisbasierte Lernen hierarchischer Modelle mit verschachtelten Zeitskalen vorgestellt, der sich sowohl für die kognitive Modellierung als auch für die Verbesserung der sequenziellen Entscheidungsfindung eignet.

Zunächst wird die kognitive Plausibilität des Ansatzes durch die Modellierung von menschlichem antizipativem Verhalten untersucht. Für die Modellierungsexperiment wird ein Agent mit einem vorstrukturierten Modell ereignisbasierter Abstraktionen ausgestattet. Wenn der Agent seinen Blick ausrichtet, um Unsicherheit über die hierarchischen Vorhersagen des Modells zu minimieren, entsteht zielantizipatives Blickverhalten ähnlich der Augenfixationen bei Säuglingen.

Das Erlernen von hierarchischen Vorhersagen setzt einen Mechanismus voraus, der Aktivität in Ereignisse einteilt. Für diesen Zweck, wird als Nächstes ein rekurrentes neuronales Netzwerk vorgestellt, das selbstständig lernt Dynamiken in latente Zustände zu komprimieren, die zeitlich selten aktualisiert werden. Die Integration dieses Mechanismus in verschiedene Vorhersage- und Planungssysteme verbessert deren Generalisierungsfähigkeit, Lerneffizienz und Erklärbarkeit.

Schließlich werden Komponenten aus den vorherigen Methoden kombiniert, um eine Hierarchie von Weltmodellen von Grund auf zu lernen. Das übergeordnete Modell in der Hierarchie wird nur aufgrund von punktuellen latenten Zustandsänderungen eines untergeordneten Dynamikmodells trainiert. Wenn das System seinen Blickfokus anhand der hierarchischen Vorhersagen auswählt, entsteht zielantizipatorisches Blickverhalten, ähnlich wie es sich bei Säuglingen im ersten Lebensjahr entwickelt. Darüber hinaus können die erlernten hierarchischen Vorhersagen nahtlos in modellbasierte Verstärkungslern- und Planungsagenten integriert werden, um deren Verhalten bei anspruchsvollen Problemen mit langen Aufgabenhorizonten zu verbessern.

Zusammengefasst bietet diese Arbeit nicht nur praktische Methoden für das Erlernen ereignisbasierter zeitlicher Abstraktionen, sondern zeigt auch, wie solche Strukturen menschliches Verhalten erklären und die Entscheidungsfindung künstlicher Agenten verbessern können.

Contents

ABSTRACT	i
1 INTRODUCTION	1
1.1 Foundations of Adaptive Problem Solving	2
1.2 Transfer Learning and Generalization	3
1.3 Hierarchical Planning	5
1.4 The Need for Sensorimotor Abstractions	7
1.5 Thesis Outline	8
2 THEORETICAL BACKGROUND	10
2.1 Event Cognition	10
2.2 Sequential Decision Making	15
3 TOWARDS SEQUENTIAL DECISION MAKING WITH EVENTS	26
3.1 THICK Markov Decision Processes	26
3.2 Hierarchical Planning	32
3.3 Examples	33
3.4 Conclusion	36
4 DEVELOPING EVENT-BASED GOAL ANTICIPATIONS	38
4.1 Introduction	39
4.2 Development of Goal Anticipations	40
4.3 Cognitive Action Prediction in Infants (CAPRI)	42
4.4 Experiments	46
4.5 Related Work	52
4.6 Discussion	53
5 SPARSELY CHANGING LATENT STATES FOR PREDICTION AND PLANNING	56
5.1 Introduction	57

5.2	L_0 Regularization of Latent State Changes	58
5.3	Gated L_0 Regularized Dynamics (GATELORD)	60
5.4	Experiments	63
5.5	Related Work	72
5.6	Discussion	74
6	HIERARCHICAL PREDICTIONS FROM DISCRETE LATENT DYNAMICS	75
6.1	Introduction	76
6.2	Learning Hierarchical Predictions	78
6.3	Experiments	83
6.4	Discussion	91
7	HIERARCHICAL WORLD MODELS	93
7.1	Introduction	94
7.2	THICK World Models	95
7.3	Downstream Applications of THICK World Models	101
7.4	Experiments	104
7.5	Related Work	108
7.6	Discussion	110
8	DISCUSSION	112
8.1	Summary of Contributions	112
8.2	Limitations and Extensions	115
8.3	The Bitter Lesson and Cognition-Inspired AI	119
8.4	Integrating Theories on Cognition	120
8.5	Outlook	124
	APPENDIX A BACKGROUND AND APPROACH: SUPPLEMENTARY MATERIAL	126
	APPENDIX B CAPRI: SUPPLEMENTARY MATERIAL	130
	APPENDIX C GATELORD: SUPPLEMENTARY MATERIAL	143
	APPENDIX D SKIP NETWORK: SUPPLEMENTARY MATERIAL	166
	APPENDIX E THICK WORLD MODELS: SUPPLEMENTARY MATERIAL	169
	REFERENCES	197
	ACKNOWLEDGMENTS	223

List of Figures

1.1	An Example of Zero-Shot Problem Solving	2
1.2	Prerequisites for Adaptive Problem Solving	3
1.3	Contributions to Adaptive Problem Solving	8
2.1	Event Partonomy	12
2.2	Markov Decision Processes and Models thereof	17
2.3	Partial Observability and Models	21
3.1	THICK MDP	27
3.2	Context Partonomy in THICK MDPs	29
3.3	Models in THICK MDPs	30
3.4	Goal-Anticipatory Gaze in THICK MDPs through Active Inference	35
4.1	Modeling Hypotheses for the Development of Goal Anticipation	41
4.2	Event Schemata in CAPRI	43
4.3	Simulation Environment of CAPRI	48
4.4	Exemplary Event and Policy Inference	49
4.5	Mean Event Inference	50
4.6	Anticipatory Gaze Behavior of CAPRI and in Infants	51
5.1	GATELORD Architecture	61
5.2	Simulation Environments of GATELORD	64
5.3	Billiard Ball Prediction	66
5.4	Generalization in Robot Remote Control	67
5.5	Precise Memorization in Shepherd	68
5.6	Sample Efficient RL in MiniGrid	69
5.7	Zero-Shot Policy Transfer in MiniGrid	70
5.8	Explainability of the Latent States	71
6.1	Hypothesis for Modeling Goal Anticipations via Event Codes	76

6.2	Skip Network Architecture	79
6.3	Prediction and Gate Regularization for Scripted Events	85
6.4	Exemplary Sensorimotor Sequences and Latent States	86
6.5	Skip Predictions for Scripted Events	87
6.6	Exemplary Event Boundary Predictions	88
6.7	Prediction Errors and Skip Predictions for RL Interactions	89
6.8	Goal-Anticipatory Gaze of our System	90
6.9	Goal-Anticipatory Gaze in Infants	90
7.1	THICK World Models	94
7.2	C-RSSM World Model	96
7.3	High-Level Segmentation	99
7.4	Temporal Abstract Predictions	101
7.5	Exemplary Context Changes	105
7.6	Exemplary High-Level Actions	106
7.7	Long-Horizon RL in MiniHack	107
7.8	Sample Efficient RL in VisualPinPad	108
7.9	Zero-Shot MPC in Multiworld	109
8.1	Towards Object-Centric World Models	117
8.2	Actions, Events, and Context in THICK MDPs	122

List of Symbols

Latin

a	action
\mathbf{a}	refers to <i>agent</i> in CAPRI (Chap. 4)
A	high-level action (Chap. 7)
c	context, state component in THICK MDPs (Chap. 3) or C-RSSM (Chap. 7)
d	episode termination / “done” (Chap. 7)
\mathcal{D}	dataset
e	event schema of CAPRI (Chap. 4)
h	RNN latent state (Chap. 5-6) or deterministic latent state component (Chap. 7)
\mathcal{H}	entropy
i	image input of C-RSSM (Chap. 7)
J	utility function for MPC (see Eq. 2.3)
K	planning horizon for MPC (see Eq. 2.3)
l	level of a hierarchy (Chap. 3)
\mathcal{L}	loss function (of low-level model)
\mathcal{L}	loss function of high level (Chap. 6-7)
\mathcal{N}	normal distribution
o	observation
p	refers to <i>patient</i> in CAPRI (Chap. 4)
P	probability
r	reward
R	reward function
s	state of MDP (Chap. 2) or internal state of world model (Chap. 7)
t	time
u	sampled update gate input (Chap. 5)
v	value function (Chap. 2) or critic (Chap. 7)
w	low-level world model (Chap. 7)
W	high-level world model (Chap. 7)
x	network input (Chap. 5-6)
y	network output (Chap. 5-6)
z	stochastic latent state (Chap. 7)

Greek

α	gaze focus (Chap. 6)
β	loss scale; for sparsity loss (GATELORD, Chap. 5 – 6) or various losses (Chap. 7)
γ	discount factor in RL (Eq. 2.2)
Δ	denotes change in a vector
ζ	loss scale for high-level model (Chap. 7)
η	learning rate of neural networks
θ	parameters of high-level model (Chap. 6-7)
ϑ	parameters of the recognition density (Sec. 2.2.4)
Θ	Heaviside step function
κ	intrinsic reward scale for hierarchical planning (Eq. 3.5, Eq. 7.26)
Λ	ReTanh gate activation function (Eq. 5.7) or (bold, $\mathbf{\Lambda}$) vector of gates (Eq. 6.2)
μ	mean of a normal distribution
ν	parameters of sparse gates (Eq. 5.3)
ξ	parameters of high-level critic (Chap. 7)
Ξ	“clock function”, determines objective time t (Eq. 3.2)
π	policy
σ	variance of a normal distribution
Σ	covariance matrix of a normal distribution
τ	function determining time step of next context change (Eq. 3.1, Eq. 6.12, Eq. 7.10)
ϕ	parameters of (low-level) model
χ	parameters of low-level critic (Chap. 7)
ψ	trade-off factor for short- and long-horizon reward estimates (Eq. 7.23)
ω	logits of stochastic state z (Chap. 7)

1

Introduction^{1.1}

Humans and several other intelligent animal species have the ability to break down complex problems into simpler, previously learned sub-problems. This hierarchical approach allows them to solve previously unseen problems in a zero-shot manner, i.e., without any trial and error. For example, Fig. 1.1 depicts how a crow solves a novel non-trivial food access puzzle that consists of three causal steps: It first picks a stick, then uses the stick to access a stone, and then uses the stone to activate a mechanism that releases food (Gruber et al., 2019). There exist numerous analogous experiments that attest similar capabilities to primates, octopuses, and, of course, humans (Butz & Kutter, 2017; Perkins & Salomon, 1992).

In the last decade, sequential decision making agents, and in particular deep reinforcement learning (RL) agents, have made remarkable progress in various domains (Mnih et al., 2015; Silver et al., 2016; Schrittwieser et al., 2020; Degraeve et al., 2022). However, these systems typically rely on extensive training and more or less clearly defined problem spaces. Adaptive problem-solving behavior in continuous space that is comparable with the crow’s behavior in Fig. 1.1 has not yet been accomplished with any artificial

^{1.1}This chapter is based on the first two sections of the publication:

Eppe, M., **Gumbsch, C.**, Kerzel, M., Nguyen, P. D., Butz, M. V., & Wermter, S. (2022). Intelligent problem-solving as integrated hierarchical reinforcement learning. *Nature Machine Intelligence*, 4 (pp. 11-20).

The figures and key ideas were adopted from the first two sections of the publication. The text was largely rewritten to serve as an introduction to this thesis.

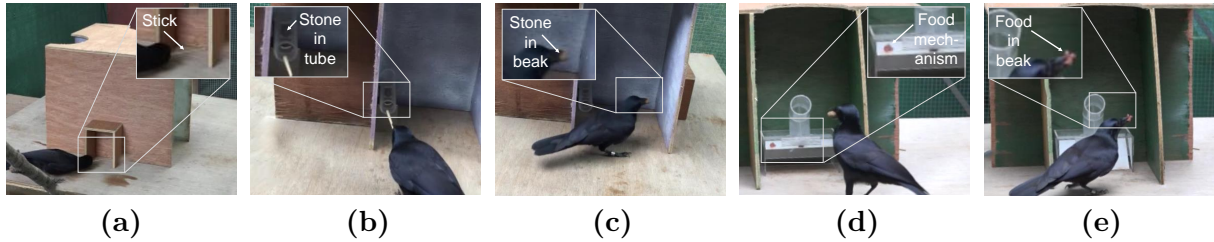


Figure 1.1: An example of zero-shot problem solving taken from Gruber et al. (2019) and previously published in Eppe et al. (2022). (a) First, the crow picks up a stick. (b-c) Then it uses the stick to pull a stone out of a tube. (d-e) Finally, it uses the stone to activate a mechanism that releases food. Although the crow has never solved this problem setup before, it is able to solve it instantly after a brief inspection phase.

system. This raises the question of how we can equip intelligent artificial agents with similar hierarchical learning and zero-shot problem-solving abilities.

A crucial ingredient to achieve flexible problem solving could be the ability to form hierarchical abstractions of an agent’s interactions with the world. Such abstractions not only allow an agent to plan goal-directed behavior on multiple levels of abstraction but could also help to reuse knowledge about past experiences to solve novel problems. This is a central theme of my thesis, which I will briefly motivate in the following.

1.1 Foundations of Adaptive Problem Solving

Few-shot problem solving is the ability to solve unknown problems with few ($\lesssim 5$) trials. *Zero-shot problem solving* is a special case of few-shot problem solving, where no additional training at all is required to solve a new problem (Kirk et al., 2023). For example, a crow can use a stick as a tool for a novel food access problem without further training (Fig. 1.1), given it has previously solved related problems (Gruber et al., 2019).

What are the cognitive abilities to achieve such adaptive decision making for complex tasks? On the one hand, **transfer learning** is required such that the skills and representations developed by the agent **generalize** across situations. For example, the problem-solving crow has previously learned solutions to similar sub-problems which are transferred and combined to solve the new problem at hand (Gruber et al., 2019). On the other hand, the agent needs the ability to **hierarchically plan goal-directed behavior**. Our puzzle-solving crow, for instance, must plan ahead, because it can only see one side of the cubic setup at a time (Gruber et al., 2019).

Thus, generalization and hierarchical planning seem to be essential components of adaptive problem solving (see Fig. 1.2): However, currently there is still a large mismatch

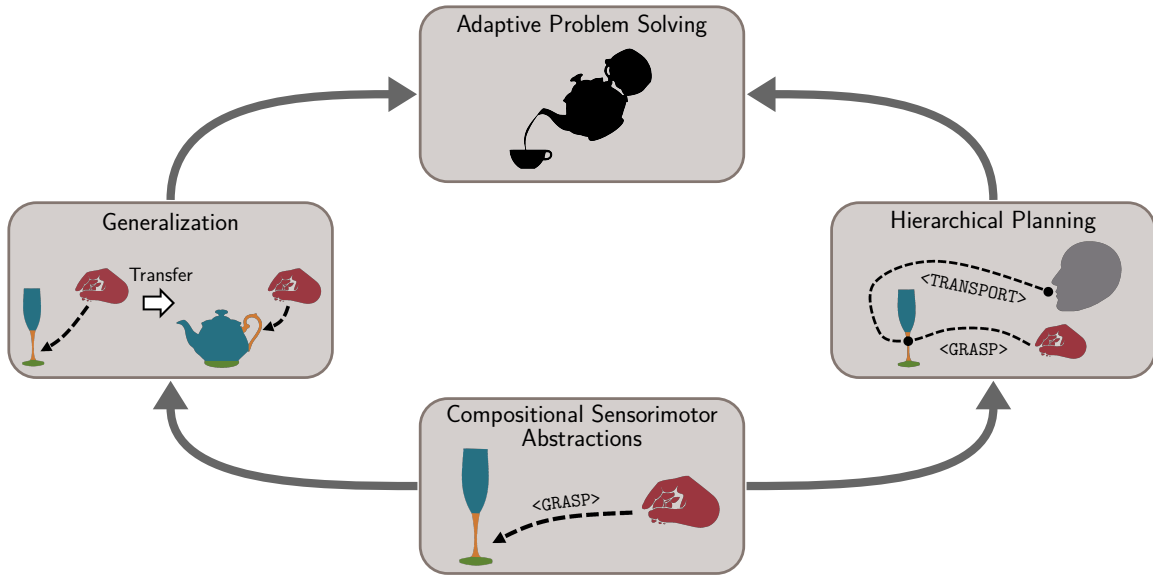


Figure 1.2: Prerequisites for adaptive problem solving: *Compositional sensorimotor abstractions* allow an agent to decompose its interactions with the world into reusable subprocesses, e.g. learning to encode a grasping event from interacting with a glass. This improves *generalization*, since the agent can reason about novel situations by transferring past solutions to novel problems, e.g. by executing suitable grasps onto other objects such as a teapot. Based on sensorimotor abstractions, the agent can flexibly achieve desired goal states by *hierarchically planning* how different processes would unfold, e.g. by planning to drink from a glass by first suitably grasping and then appropriately transporting it. In summary, these abilities enable *adaptive problem solving*.

between biological and artificial agents when it comes to these abilities. What causes this mismatch? And how can we improve artificial agents to close this gap? As we will see in the following, **compositional sensorimotor abstractions** could be one of the missing pieces needed to close this gap.

1.2 Transfer Learning and Generalization

In biological agents the ability to generalize, also known as *transfer learning*, is crucial for adaptive behavior. With respect to problem solving, transfer learning allows applying the solution of a previously solved task to novel, but analogous, unseen tasks. This ability can occur at different levels of abstraction (Perkins & Salomon, 1992). In *near transfer*, skills are transferred across similar domains, e.g. learning to grasp a glass and

then learning to grasp other objects. In contrast, *far transfer* requires the transfer of more abstract solutions between different situations, typically via analogies. As an example of far transfer, consider the medical scenario where a doctor has to figure out how to destroy an aggressive tumor through radiation without damaging the surrounding tissue (Duncker & Lees, 1945). When participants in an experiment were faced with this thought problem, they more often found the solution to attack the tumor through multiple weak beams of radiation after reading an analogous story of how soldiers took over a fortress by attacking simultaneously from multiple directions (Gick & Holyoak, 1980). Such *analogical reasoning* has been considered a critical cornerstone of higher-order cognitive processes, including reasoning, communication, creativity, and scientific discovery (Gentner & Maravilla, 2017; Gentner et al., 1997; Gentner, 2006).

One proposed mechanism enabling transfer learning is *compositional generalization* (Lake et al., 2017; Frankland & Greene, 2020; Ito et al., 2022). In linguistics, expressions are compositional if they are composed of sub-expressions and rules to determine the semantics of the composition. The language of thought theory (Fodor, 2001) transfers the compositionality principle from language to abstract mental representations, claiming that thought must also be compositional. For example, the interaction <grasping a glass> could be represented compositionally by the action <grasping> and the target <glass>. This enables the generation of a novel interaction given a new target, e.g. <grasping a teapot> (see Fig. 1.2). While there exists behavioral evidence (Lake et al., 2019; Franklin & Frank, 2020; Dekker et al., 2022) as well as neural evidence (Haynes et al., 2015; Frankland & Greene, 2020; Ito et al., 2022) for compositional generalization in human decision making, it is not clear how compositional representations are learned from sensorimotor experiences.

In artificial systems we can distinguish between two types of generalization: *in-distribution* generalization and *out-of-distribution* (OOD) generalization, depending on whether the new data is drawn from the same distribution as the training data or not. Current machine learning approaches, e.g. deep learning, are built around the assumption that their data is independent and identically distributed (IID) (Bishop, 2006). Assuming that sufficient training data is available, these systems can handle in-distribution generalization within the IID setting very well (Schölkopf, 2019). In the more challenging OOD setup, test data on which generalization is tested stems from parts outside of the distributions from which the training samples were drawn (Schölkopf, 2019; Goyal & Bengio, 2022). Causality research claims that OOD generalization is a fundamental challenge for deep learning, which purely models the statistical dependencies of training data but not their causal relationships (Schölkopf, 2019; Peters et al., 2017; Schölkopf et al., 2021). For example, a vision-based agent that is prolific at stacking colorful blocks might fail when encountering a white block, despite that color is causally irrelevant for block stacking.

Generically improving OOD generalization is not possible, because improving the ability to generalize in one setting might harm generalization in another setting (Kirk et al., 2023). However, by discovering the causal structure of the world via suitable inductive biases a system could learn to generalize across various scenarios that share the same causal structure.

How are the generative processes in our world causally structured? Principles from causality research propose that generative processes of a system can be decomposed into entities and mechanisms affecting them (Schölkopf et al., 2021; Goyal & Bengio, 2022)^{1,2}. Causal mechanisms are assumed to be independent of each other and tend to affect variables only locally (Schölkopf, 2019; Schölkopf et al., 2021). For example, a block stacking robot could learn that collisions selectively affect some variables that describe the entities involved, e.g. their velocities, but do not influence other variables, e.g. color. As a result, the robot could seamlessly generalize to stacking blocks with unseen colors. Although recent research has made significant advances in extracting entities (Locatello et al., 2020; Kipf et al., 2021; Traub et al., 2023; Seitzer et al., 2023; Zadaianchuk et al., 2023) or causal mechanisms (Pitis & Garg, 2020; Goyal et al., 2021a,b; Seitzer et al., 2021) from raw data, these approaches still have several drawbacks, e.g. detecting a predefined number of objects or mechanisms, which hinders their universal application.

1.3 Hierarchical Planning

In **biological agents** behavior is traditionally divided into two categories: Stimulus-driven habitual behavior and goal-directed planned behavior (Dayan, 2009; Dolan & Dayan, 2013; O’Doherty et al., 2017). Habitual behavior refers to forms of reflexive control that are strongly automated, computationally efficient, and primarily learned from past reinforcements (Dolan & Dayan, 2013). In the early 20th century, theories on motor control mostly focused on habitual accounts of behavior. Behaviorists, such as Watson (1920) and Washburn (1916), proposed that complex behavior, e.g. speech, could be explained by associative reflex chains: stimulations from performing a certain action would trigger an associated next action in the chain. This view was challenged by a number of protagonists, such as Tolman (1948) and Lashley (1951).^{1,3} The idea that humans and other animals can hierarchically plan their behavior in a goal-directed fashion was a central insight that contributed significantly to the cognitive revolution (Botvinick, 2008).

^{1,2}Note that this assumption is very similar to the compositional representations proposed for human cognition. It is possible that humans and other animals have evolved the tendency to represent their experience in a compositional format because this best reflects the causal structure of the world.

^{1,3}We provide more details of their prepositions and contributions in Suppl. A.1.

Many contemporary theories and models in cognitive science agree that behavior in humans and other animals are encoded in a hierarchical part-whole organization (Cooper & Shallice, 2000; Zacks & Tversky, 2001; Botvinick, 2008; Butz, 2016; Pezzulo et al., 2018). When planning one’s behavior, humans seem to make use of this hierarchy. By now, there exists numerous experimental findings that humans plan their behavior hierarchically in various domains, ranging from simple coordinated button presses (Cohen et al., 1990; Keele et al., 1990), to spatial navigation (Wiener & Mallot, 2003; Solway et al., 2014; Balaguer et al., 2016), and speech production (Lee et al., 2013).

In artificial agents there have been tremendous amounts of effort to integrate hierarchies into problem-solving and planning approaches since the early days of AI. Already the famous general problem solver by Newell et al. (1959) attempted to solve problems by hierarchically breaking them down into simpler sub-problems. In robotics different behavior decompositions have been proposed (Schaal, 2003; Ijspeert et al., 2013; Peters & Schaal, 2008; Toussaint et al., 2018) that chunk complex movements into simpler parts, which can then be easily optimized. However, none of these approaches attempts to learn the segmentation of behavior from scratch. Instead, the primitive elements of behavior are predefined, or a segmentation is provided by an expert.

The strive to learn hierarchies of behavior and make use of these hierarchies for goal-directed behavior, spawned the field of hierarchical reinforcement learning (HRL) (Sutton et al., 1999b; Pateria et al., 2021; Eppe et al., 2022). In HRL, an agent selects actions on different levels along a hierarchy in order to maximize future expected rewards. Actions on a higher level, also known as options (Sutton et al., 1999b) or skills (Eysenbach et al., 2018), typically encode a temporally extended sequence of low-level actions. For example, a <grasp>-option could involve primitive actions that first move a gripper to an object and then close the gripper. Although there has been great research to advance HRL in the last two decades (see Pateria et al., 2021 for a review), groundbreaking achievements in the field of RL were primarily achieved by flat, non-hierarchical approaches^{1,4}. HRL systems typically introduce more hyperparameters than their flat counterparts, which requires a more elaborate hyperparameter search, while only achieving minor gains in standard benchmarks (Bacon et al., 2017; Gürtler et al., 2021; Hafner et al., 2022). Despite the evidence for hierarchical outcome-oriented planning in biological agents and the success of integrating model-based planning into RL (Silver et al., 2016; Schrittwieser et al., 2020), most HRL approaches learn reactive, habitual high-level policies (Pateria et al., 2021; Eppe et al., 2022). The few HRL approaches that leverage model-based planning across multiple levels of a hierarchy rely on hand-crafted dependencies (Pateria et al.,

^{1,4}For example, groundbreaking achievements of RL include beating humans for the first time in a game, such as Atari video games (Mnih et al., 2015) or the challenging game of Go (Silver et al., 2016) or achieving superhuman performance in multiple games with the same system (Schrittwieser et al., 2020).

2021; Eppe et al., 2022). Thus, apparently HRL still lacks robust mechanisms to segment sensorimotor sequences into stable, model-based, hierarchical representations to reach its full potential.

1.4 The Need for Sensorimotor Abstractions

I have argued that flexible zero-shot problem solving depends on *robust generalization* as well as *hierarchical planning*. Both of these abilities rely on suitable compositional abstractions of the underlying sensorimotor processes. From a causality perspective, discovering and encoding the modular causal mechanisms which constitute the generative model of the world could foster OOD generalization of deep learning agents. From a planning perspective, hierarchical methods, such as HRL, need robust mechanisms to develop model-based temporal abstractions of behavior in order to plan on multiple time scales. In sum, problem-solving agents that encode their interactions with the world through such model-based abstractions of sensorimotor dynamics could potentially plan complex behavior in novel or changing environments.

Are there corresponding representations proposed for human cognition from which we can take inspiration? A large body of findings from different fields of cognitive science suggest that humans perceive, remember, and predict sensorimotor activity in terms of **events** (see Radvansky & Zacks, 2014 for an overview). Different theories (Hommel et al., 2001; Zacks et al., 2007; Butz et al., 2021) propose that internal event encodings, also called event codes, event models, or event schemata (Radvansky & Zacks, 2014), encode particular spatiotemporal processes, e.g. a grasping interaction. These event encodings seem to exist on different hierarchical levels of abstraction and for different nested temporal granularities (Zacks & Tversky, 2001; Zacks et al., 2001b; Cooper, 2021; Kuperberg, 2021). For example a <grasping a teapot>-event may be part of a longer <preparing tea>-event (Kuperberg, 2021). Event encodings have been proposed to play a crucial role in causal inference (Radvansky & Zacks, 2014), planning behavior (Hommel et al., 2001; Butz, 2016; Cooper, 2021), action understanding (Elsner & Adam, 2021; Kuperberg, 2021), and language learning (Knott, 2012; Gärdenfors, 2014).

In this thesis, I attempt to enhance artificial agents by the ability to encode their experiences in terms of events and use these encodings to hierarchically predict and plan their interactions with the world. Thereby, I follow an interdisciplinary approach based on cognitive science by taking inspiration from the event processing proposed for biological agents. Besides employing the derived mechanisms for planning and RL, their cognitive plausibility is verified by modeling human behavioral data. Some of the contributions towards flexible problem solving are illustrated in Fig. 1.3. The remainder of this chapter provides an outline of the thesis.

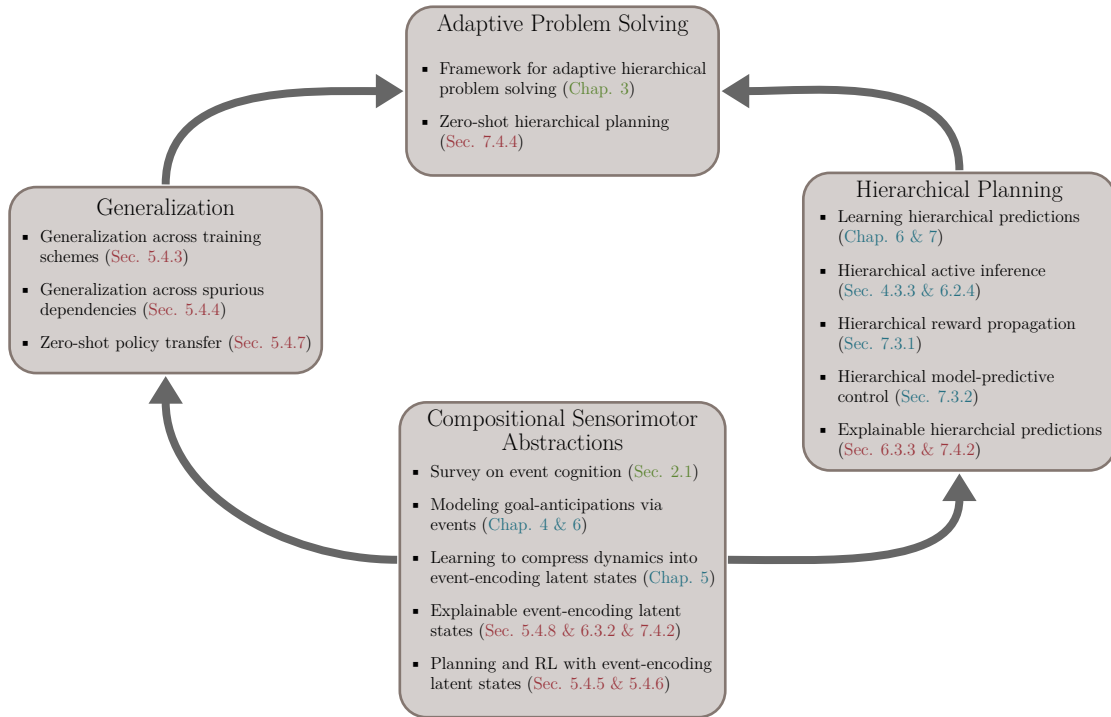


Figure 1.3: Contributions to adaptive problem solving of the present research categorized according to four key mechanisms (cf. Fig. 1.2). Theoretical results and literature surveys are shown in green. Computational models and methods are displayed in blue. For every mechanism exemplary empirical results are linked in red.

1.5 Thesis Outline

Chapter 2 provides the **theoretical background** for event cognition in humans and for sequential decision making in artificial agents. **Chapter 3** integrates these insights to develop a **theoretical framework of hierarchical temporal abstraction** suitable for model-based planning. In the following chapters, concrete computational implementations are introduced that implement various aspects of this general approach. In **Chapter 4**, a Bayesian model is outlined that is able to **model the development of anticipatory behavior** in humans based on event encodings. **Chapter 5** introduces GATELORD, a recurrent neural network which develops **event-encoding latent states** that compress particular sensorimotor dynamics into sparsely changing codes. This compression can improve various aspects of state-of-the-art planning and RL methods, especially their ability to generalize across spurious dependencies in the training data. In the remainder

of the thesis, GATELORD will serve as the central deep learning building block to develop temporal abstractions. **Chapter 6** revisits the modeling of the development of goal-anticipatory behavior, but this time by **learning temporal abstractions** from scratch based on the latent states of GATELORD. In **Chapter 7** this mechanism of hierarchical abstraction is extended to learn a **hierarchy of world models** for hierarchical problem solving. The resulting system learns high-level action abstractions and can **hierarchically plan goal-directed behavior**, showing some degree of zero-shot planning capabilities in highly challenging tasks with pixel-based inputs. Finally, in **Chapter 8** I **discuss the general approach** of this thesis.

2

Theoretical Background

Enhancing decision making agents via the ability to learn sensorimotor abstractions is a highly interdisciplinary endeavor. We need to bridge the gap between artificial intelligence research on **sequential decision making**, including paradigms such as reinforcement learning and model-predictive control, and the research area of **event cognition** which studies how humans encode their experience. In this chapter, I separately review the theoretical background on event cognition (Sec. 2.1) and sequential decision making (Sec. 2.2). In the following chapter, I will build upon the insights laid here to develop an integrated framework.

2.1 Event Cognition

We humans *continuously* act on our world and perceive a *continuous* stream of high-dimensional, perceptual information. However, a large body of psychological, neurological, and linguistic evidence suggests that we perceive, memorize, and predict activity in terms of *discrete* events (Zacks & Tversky, 2001; Radvansky & Zacks, 2014; Butz et al., 2021).

What is an event? Zacks & Tversky (2001) define an event as a “segment of time at a given location that is conceived by an observer to have a beginning and an end” (Zacks & Tversky, 2001, p. 3). Baldwin & Kosie (2021) describe events as “structured, describable, memorable units of experience” that are constructed from multi-dimensional sensory flow of information (Baldwin & Kosie, 2021, p.82). While both definitions are somewhat vague,

they already allude to a number of important properties: Namely, (1.) that events arise in perception, (2.) they extend over some time, and (3.) that events are more or less clearly separated in time. These separations between two events are commonly referred to as *event boundaries*.

In the following, I will review evidence and theories that attempt to explain which type of representations give rise to event perception and how these mechanisms influence other cognitive processes such as causal inference and planning. For a more detailed review on event cognition see [Radvansky & Zacks \(2014\)](#) or the special issue on event-predictive cognition introduced by [Butz et al. \(2021\)](#).

2.1.1 Segmentation of Events

A large body of findings on event perception come from video segmentation tasks, originally introduced by [Newtson \(1973\)](#). In these tasks, participants watch videos of everyday activities, such as cooking or cleaning. Participants are instructed to press a button when “one meaningful unit of activity ends and another begins” ([Radvansky & Zacks, 2014](#), p. 81). Participants consistently segment the videos at similar points in time, producing reliable segmentations both between subjects as well as within the same subject at different testing times ([Newtson & Engquist, 1976](#); [Speer et al., 2003](#)). Additionally, neuroimaging studies revealed that the neuronal activity in multiple brain regions strongly changes at the points of event boundaries, even when just passively watching the movies ([Zacks et al., 2001a](#)). Similar results were found for participants reading narrative text ([Speer et al., 2007](#)). These findings suggest that segmentation occurs automatically and that people naturally segment perceived activity ([Zacks & Swallow, 2007](#)).

This process of segmenting activity into events appears to be a hierarchical process ([Zacks & Swallow, 2007](#); [Zacks & Tversky, 2001](#)). For example, the granularity of partitions in the video segmentation paradigm can be easily manipulated: Participants can be instructed to segment videos into *fine* or *coarse* units that appear meaningful to them ([Newtson, 1973](#)). When participants segment videos with different granularities, their coarse boundaries tend to align with fine boundaries and each coarse segment consists of multiple fine segments ([Zacks et al., 2001b](#)). [Zacks & Tversky \(2001\)](#) propose that representations of events are hierarchically organized as a *partonomy* with nested time scales, illustrated in [Fig. 2.1](#). That is, an event at a certain level of the hierarchy, e.g. <fill a cup with tea>, can be part of a longer event at a higher level, e.g. <prepare a cup of tea>, and can itself be partitioned into smaller parts on a lower level of the hierarchy, e.g. <grab teapot>, <pour tea into a cup>, etc.

Why have humans evolved the tendency to segment activity in such a hierarchical fashion with strict boundaries? Event Segmentation Theory (EST) ([Zacks et al., 2007](#))

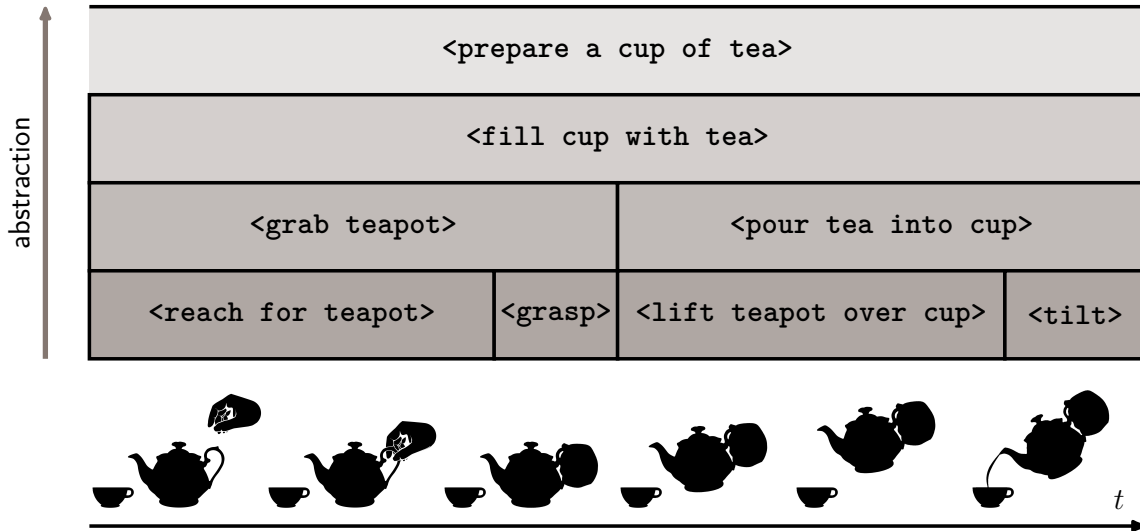


Figure 2.1: Event partonomy: Hierarchical organization of events proposed by [Zacks & Tversky \(2001\)](#). An event on a certain level of the hierarchy can be partitioned into multiple parts on a lower level of the hierarchy. The time scales are nested, i.e. event boundaries on a high level coincide with event boundaries on a lower level.

argues that humans perceive activity in this way, because these segmentations mirror the internal representation of the unfolding experience. According to EST, and in line with various theories on predictive processing ([Friston, 2010](#); [Hohwy, 2013](#); [Clark, 2015](#)), humans continuously attempt to predict future perceptual input. EST argues that *event schemata* are formed to better predict the next immediate perceptions ([Zacks et al., 2007](#); [Radvansky & Zacks, 2014](#); [Richmond & Zacks, 2017](#)). These event schemata are similar to scripts ([Schank & Abelson, 1975](#)) and encode the typical dynamics of familiar activities. For example, the schema for *<prepare tea>*, could detail all the steps involved in the process, which allows predicting someone’s behavior when observing them preparing a cup of tea. According to EST, at every point in time *event models* are active, i.e. concrete instances of previously learned event schemata. During an ongoing event, the activation of the current event models are robust towards perceptual inputs and remains constant ([Zacks et al., 2007](#)). However, when transient prediction errors occur, the active model may change to improve the predictions ([Zacks et al., 2007](#)). Once such an event boundary is perceived, the catalogue of event schemata may give rise to a new event model. For example, if during the *<prepare a cup of tea>*-event, the teapot unexpectedly drops and shatters on the floor, the new event model may be instantiated from the *<clean the floor>*-event schema.

2.1.2 Events and Causality

Besides simplifying predictions, Radvansky & Zacks (2014) argue that event perception could also guide causal inference. Changes or interventions in causal processes tend to coincide with event boundaries, e.g. the successful execution of the <grasp the teapot>-event causes the teapot to move together with the hand. In classical experiments, Michotte (1964) studied the perception of causality by showing participants videos of simple shapes moving. In one line of experiments, one shape moved in one direction, stopped, and then a second shape continued the movement. The perception of one shape causing the other to move was most vivid when the shapes were close and the pause was brief. Mapping this onto the outlined paratomy of events, translates to events <shape A moves> and <shape B continues the motion> being directly connected by an event boundary, instead of being interspersed with the event <shape A stands still>. On a higher level the two events could be combined to one event of <shape A causes shape B to move>. The perception of such a *single* high-level event could be critical to perceive the motions as a causal interaction (Radvansky & Zacks, 2014). Furthermore, there is a large body of evidence from memory tasks that a series of events that are causally connected are retained better in memory than events without a causal connection (Trabasso & van den Broek, 1985; Radvansky & Copeland, 2000; Radvansky et al., 2005). Since chunking is an effective technique to memorize sequences (Miller, 1956), a potential explanation for this effect is that causally connected low-level events are combined to form high-level chunks, whereas causally disconnected events are not.

2.1.3 Events and Actions

So far, I have mostly outlined how events influence perception and memory. What is the role of events in the generation of actions? Theory of Event Coding (TEC) (Hommel et al., 2001) argues that action and event perception involve identical processes that use the same common representations, i.e. *event codes* (Hommel et al., 2001; Hommel, 2019). For example, the event <reach for a teapot>, is both linked to the motor codes for conducting the reaching movement as well as to the perceptual features of the final action effect, e.g. the visual percept and haptic sensation of the hand touching the teapot. According to TEC, learning of event codes starts early in infancy by first associating involuntary reflex-like motions with the perception of the thereby produced effects (Elsner & Hommel, 2001). With more coordinated movements, more sophisticated event codes may develop. These event codes can then be used for anticipatory behavioral control (Hommel, 2009), in line with ideomotor theories of action (Stock & Stock, 2004). By considering a desired effect, e.g. perceptual features of holding a teapot, the associated event code, e.g. <reach for a teapot>, is activated, which automatically triggers the

corresponding motor commands. Note that what TEC refers to as an action effect, e.g. a grasp or opening a door (Hommel et al., 2001), typically corresponds to an event boundary. TEC provides a unifying explanation for various experimental findings that support strong links between action and perception (Hommel et al., 2001; Hommel, 2019).

In short, TEC proposes that events act as mediators between actions and goals. However, the explanations of TEC found in the literature (Hommel et al., 2001; Hommel, 2019) focus mainly on rather simple behavior, such as reaching movements. Is it possible that this relationship scales up to more complex action sequences? Recent theories on event-predictive cognition (Butz, 2016; Butz et al., 2021; Kuperberg, 2021; Cooper, 2021; Elsner & Adam, 2021) argue how hierarchical organization of predictive event representations could enable complex reasoning and planning. Butz (2016) sketches out how hierarchical abstraction and chunking of events could lead to abstract symbol-like concepts, e.g. `<working on someones career>`, grounded in sensorimotor experiences. When invoking these event encodings for hierarchical model-based planning, they link back to concrete sensorimotor consequences. Kuperberg (2021) proposes that hierarchical generative models of events could not only be used to select own goals while planning but also to infer the goals of others when observing them. Along similar lines, Cooper (2021) argues that top-down goals trigger events within the hierarchy during sequential action production, whereas during event perception bottom-up sensory cues cause their activation.

2.1.4 Conclusion

In summary, there is converging evidence from different fields of cognitive science that humans structure their sensorimotor experience into highly specialized event encodings. The integration of various theories on event cognition implies that events guide the perception of the present, structure memories about the past, and enable prediction and planning of the future. **Box 1** briefly summarizes key findings about the proposed structure and functions of event encodings.

Box 1 Event Encodings

Humans appear to represent temporal activity in terms of events. By integrating different theories and experimental evidence on event cognition (Zacks & Tversky, 2001; Hommel et al., 2001; Zacks et al., 2007; Radvansky & Zacks, 2014; Butz, 2016; Butz et al., 2021), I hypothesize the following properties to be crucial for the underlying event encodings:

- **Temporal persistence:** events discretize the continuous stream of experience into segments with stable activation of event encodings and distinct points in time, i.e. event boundaries, where activations change.
- **Hierarchically nested time scales:** event encodings are hierarchically organized in a partonomic structure, where a longer event on a high level can be partitioned into multiple shorter events on a lower level of the hierarchy.
- **Model-based predictions:** when activated, an event encoding predicts how a certain event unfolds.
- **Action-goal-association:** representations of event boundaries that encode the end state or effect of an event are associated with the actions necessary to produce the corresponding effect.

2.2 Sequential Decision Making

In this section, I will briefly review the theoretical foundations of sequential decision making in Markov Decision Processes and variants thereof. Thereby, I will shortly introduce common machine learning and sequential decision making methods to approach such problems, including reinforcement learning and model-predictive control. Finally, I will also introduce the active inference framework, which provides an objective based on computational neuroscience for sequential decision making. For more detailed reviews, see Sutton & Barto (2018) for reinforcement learning, Bertsekas (2012) for control theory, and Parr et al. (2022) for active inference.

Notation In this section and the remainder of this thesis, I will use the following notation: Bold lowercase letters denote vectors (e.g. \mathbf{x}). Numbers (e.g. x) and functions (e.g. $f(\cdot)$) are written in light typeface. Subscript typically denotes time indices for numbers or vectors (e.g. \mathbf{x}_t or x_t). Temporal sequences are also denoted using subscript and a colon defining the range (e.g. $\mathbf{x}_{1:t} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t)$). For functions with adjustable parameters, e.g. neural networks, subscript denotes the parameters (e.g. $f_\phi(\cdot)$ for neural network f with parameters ϕ). Superscript denotes dimensions of a vector ($\mathbf{x} = [x^1, x^2, \dots, x^n] \in \mathbb{R}^n$) or additional information (e.g. \mathbf{x}^{info}).

2.2.1 Decision Making in Fully-Observable Processes

An idealized mathematical framework for sequential decision making is the **Markov Decision Process** (MDP) (Bellman, 1957; Sutton & Barto, 2018). In an MDP there exists an *agent* that can interact with the *environment* at discrete time steps t , as illustrated in Fig. 2.2a. Formally, an MDP can be defined by the 5-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, with a set^{2.1} of states \mathcal{S} , a set of actions \mathcal{A} , a transition function T , a reward function R , and a discount factor γ . At every time step t , the MDP is in some state $\mathbf{s}_t \in \mathcal{S}$. The agent observes this state \mathbf{s}_t and can take an action $\mathbf{a}_t \in \mathcal{A}$. In some cases, not all actions are available in every state, e.g. in chess an agent can only move the pieces that are still in the game. For such problems, we can denote the available actions for state \mathbf{s}_t as $\mathcal{A}(\mathbf{s}_t)$. After executing action \mathbf{a}_t , the environment transitions to a new state \mathbf{s}_{t+1} . The transition is modeled by the transition function T , with $T(\mathbf{s}_{t+1} | \mathbf{a}_t, \mathbf{s}_t)$ describing the probability of reaching state $\mathbf{s}_{t+1} \in \mathcal{S}$ when action \mathbf{a}_t is executed in the previous state \mathbf{s}_t . Additionally, upon each transition the agent has a chance to receive a scalar reward $r_{t+1} \in \mathbb{R}$. The reward obtained is defined over the probabilistic reward function $R(r_{t+1} | \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$.

Thus, both the transitions and the reward distribution are stochastic processes. Taking an action in a certain state does not necessarily guarantee a certain outcome. This enables MDPs to model sources of randomness in the agent-environment interaction. However, an important property of the MDP is that the transitions *only* depend on the current state and action. Transitions are conditionally independent of all previous states and actions. We can express this property as

$$T(\mathbf{s}_{t+1} | \mathbf{s}_{1:t}, \mathbf{a}_{1:t}) = T(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) \quad (2.1)$$

for the transition probability T and sequences of states $\mathbf{s}_{1:t}$ and actions $\mathbf{a}_{1:t}$. This property is called the **Markov property**. For this property to hold, a single state-action-pair needs to contain all relevant information to predict a transition.^{2.2} This is a strong assumption that typically does not apply to realistic processes. We will loosen this assumption later with respect to time (see Sec. 2.2.2) and with respect to which information is available or observable for the agent (see Sec. 2.2.3).

In **reinforcement learning** (RL), the goal is to maximize the discounted cumulative rewards the agent receives in the long-run or over its life time (Sutton & Barto, 2018). For this purpose, the agent learns a *policy* π . A policy is a function $\pi(\mathbf{a}_t | \mathbf{s}_t)$ mapping from a state $\mathbf{s}_t \in \mathcal{S}$ to the probability of selecting an action $\mathbf{a}_t \in \mathcal{A}$. We can quantify how

^{2.1}For simplicity, in this section I assume finite sets for actions and states instead of continuous spaces. In principle, all formulations can be extended to continuous state- or action spaces by replacing sums, e.g. $\sum_{\mathbf{s} \in \mathcal{S}}(\cdot)$, with integrals, e.g. $\int_{\mathcal{S}}(\cdot) d\mathbf{s}$, when marginalizing over \mathcal{S} (analogously for \mathcal{A}).

^{2.2}The Markov property typically not only holds for state transitions but also for rewards and can be formulated equivalently to Eq. 2.1 for the reward function $R(\cdot)$.

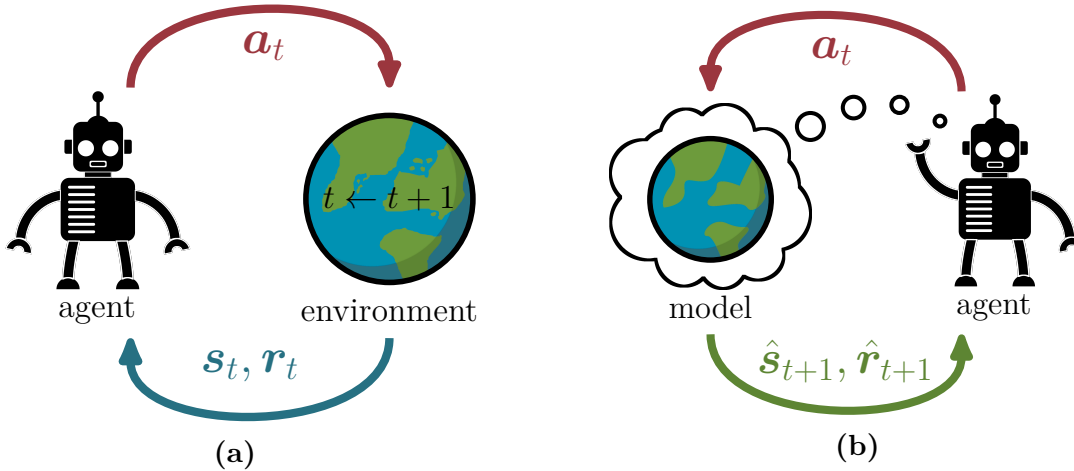


Figure 2.2: Markov Decision Processes and models thereof: (a) In a Markov Decision Process an agent interacts with its environment at discrete time steps t . At every time t the agent observes a state \mathbf{s}_t and may receive a reward r_t . Then the agent executes an action \mathbf{a}_t and the environment transitions to a new state. (b) In model-based approaches the agent learns an internal model of the transitions.

well a policy is suited to maximize cumulative rewards in terms of the expected utility function G , with

$$G(\mathbf{s}_t, \pi) = \mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k R(r_{t+k+1} \mid \mathbf{s}_{t+k}, \mathbf{a}_{t+k}, \mathbf{s}_{t+k+1}) \quad (2.2)$$

from a starting state \mathbf{s}_t . The expectation over the policy denotes that all actions are distributed according to the policy π , i.e. $\mathbf{a}_t \sim \pi(\mathbf{a}_t \mid \mathbf{s}_t)$ and state transitions follow this action distribution.^{2,3} Additionally, Eq. 2.2 introduces the **discount factor** $\gamma \in [0, 1]$. The discount factor is a parameter that determines how much future rewards affect the evaluation of the present (Sutton & Barto, 2018). For example, for $\gamma = 0$ the agent focuses purely on maximizing the current reward. On the other hand, larger values of γ increase the effect of future outcomes on the current utility G . An important effect of the discount factor is that when rewards are bounded and $\gamma < 1$, the sum in Eq. 2.2 is bounded despite summing over an infinite number of elements (detailed in Suppl. A.2.2).

The overall objective in RL is to find an optimal policy π^* that maximizes the objective defined in Eq. 2.2. A typical approach is to learn a **value function** v^π . The value function

^{2,3}I provide a more formal definition in Suppl. A.2.1.

$v^\pi(\mathbf{s}_t)$ assigns a scalar *value* to every state \mathbf{s}_t by approximating $G(\mathbf{s}_t, \pi)$ for the policy π . The policy can then be optimized from the value estimates.

RL in its classical form is **model-free** (Sutton & Barto, 2018): The agent does not have access to the underlying transition function of the MDP nor does the agent attempt to learn a model of it. Common approaches of model-free RL are temporal difference learning (Sutton, 1988), policy gradient methods (Sutton et al., 1999a), or actor-critic approaches (Konda & Tsitsiklis, 1999; Schulman et al., 2017).

In contrast, in **model-based** RL a model of the transition function is involved (illustrated in Fig. 2.2b). In some cases, the underlying transition function is known, e.g. the rules of a game such as chess. More commonly, transition functions are unknown and a model of the MDP needs to be learned from the data collected by the agent. To this end, a function $f_\phi(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ with parameters ϕ can be learned to approximate the transition function $T(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$.^{2.4} A straightforward use case for such a model is data generation, as proposed by the DYNA algorithm (Sutton, 1991): Here, the policy is trained not only on the data collected by the agent but also on simulations using the model. Since model simulations are often computationally less expensive than taking steps in the environment, this process can speed up training. Contemporary deep RL approaches (Ha & Schmidhuber, 2018; Hafner et al., 2019a, 2020a, 2023) use this technique to boost sample efficiency when training a policy.

Another application of a model is to deploy it for **model-based planning**. Traditionally, planning methods optimize a policy π for a finite horizon K (Bertsekas, 2012). Thus, instead of Eq. 2.2 we can phrase the objective as

$$J(\mathbf{s}_t, K, \pi) = \mathbb{E}_\pi \sum_{k=0}^K R(r_{t+k+1} \mid \mathbf{s}_{t+k}, \mathbf{a}_{t+k}, \mathbf{s}_{t+k+1}) \quad (2.3)$$

with actions \mathbf{a}_k ^{2.5} and expected states \mathbf{s}_k . Note, that the discount factor γ is not required here, unlike in Eq. 2.2 where it is needed to ensure that the sum over a infinite number of rewards is bounded (see Suppl. A.2.1).

In its simplest form, the policy is a sequence of actions^{2.6} (Bertsekas, 2012). For planning, a model is used to simulate outcomes and optimize the policy. If the policy is

^{2.4}The function f_ϕ can also model the reward function, i.e. $f_\phi(\mathbf{s}_{t+1}, r_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. For simplicity, this will not be mentioned explicitly in the remainder of this chapter.

^{2.5}In control theory a different notation is commonly used: actions are denoted with \mathbf{u}_t and states are denoted with \mathbf{x}_t . Besides that, in control theory the objective is usually to minimize a cost rather than maximizing rewards per step (Bertsekas, 2012). For simplicity, we can treat the reward function R as a cost function with a flipped sign and minimize the objective instead of maximizing it.

^{2.6}Thus, in this case $\pi = \mathbf{a}_{t:t+K}$. Note, that this is just a special case of the more general definition of a policy as a mapping from a state to the probability of selecting an action. In this special case, the probability for action \mathbf{a}_t in the sequence is 1 at time t and the probability for all other actions is 0.

optimized at time t and subsequently executed without further optimization, the process is called *open-loop control*. In contrast to *closed-loop control*, no feedback is taken into account while executing the policy. Open-loop control is typically not desired, since model error can quickly accumulate and the agent cannot react to unpredicted situations. **Model-Predictive Control** (MPC) tackles this problem by repeatedly planning each time step. At each time step t , the policy π is optimized according to Eq. 2.3, the first action \mathbf{a}_t is taken, feedback on the new state \mathbf{s}_{t+1} is received, and the process is repeated (Glad & Ljung, 2018). Thus, by iteratively performing the process, the control loop is closed.

There are various algorithms for model-based trajectory optimization in MPC. A common optimization method for planning (Chua et al., 2018; Hafner et al., 2019b; Pinneri et al., 2021a; Sancaktar et al., 2022; Scholz et al., 2022) is the **Cross Entropy Method** (CEM) (Rubinstein & Davidson, 1999). When using CEM for MPC with a planning horizon K , a sequence of actions is sampled from K distributions over actions. The outcomes of executing an action sequence are predicted by means of a model and evaluated based on some optimization criterion (e.g. R in Eq. 2.3). After evaluating a fixed number of sampled trajectories, the sampling distributions are adapted based on the best samples. This process is repeated several times before executing the first action of the best policy. CEM is related to evolutionary optimization algorithms (Beyer & Schwefel, 2002). Since CEM is a zero-order optimization method and makes no assumptions about the model involved, CEM is a versatile method for many planning applications. Other planning methods include tree search algorithms for discrete actions (Silver et al., 2016; Schrittwieser et al., 2020; Bagatella et al., 2021), or gradient-based optimization for differentiable models (Otte et al., 2017; Scholz et al., 2022). Model-based planning can also be seamlessly combined with RL to guide the optimization of a policy based on previously planned action sequences (Schrittwieser et al., 2020; Pinneri et al., 2021b).

2.2.2 Temporal Abstractions

MDPs assume discrete time steps, for which an action is taken in every time step. However, in the real world time is continuous, actions can be taken at arbitrary points in time, and different actions might take different durations to execute. To model processes with transitions of variable duration, **Semi-Markov Decision Processes** (SMDPs) have been proposed (Howard, 1964; Puterman, 1990). SMDPs generalize MDPs by formalizing the time between transitions via a random variable (Howard, 1964). SMDPs can be formally defined as a 6-tuple $(\mathcal{S}, \mathcal{A}, T, R, F, \gamma)$. In SMDPs, the system evolves based on an action^{2.7} $\mathbf{a} \in \mathcal{A}$ by remaining in a state $\mathbf{s} \in \mathcal{S}$ for a random amount of time before

^{2.7}For clarity, I omit time indices in this subsection.

transitioning to the next state \mathbf{s}' according to the transition probability $T(\mathbf{s}' | \mathbf{s}, \mathbf{a})$ (Puterman, 1990). The temporal transition function $F(t | \mathbf{s}, \mathbf{a})$ denotes the probability that a transition occurs within t time steps (Puterman, 1990). Crucially, when only looking at the points of transitions, and ignoring the true underlying time, the Markov property still holds for SMDPs (Howard, 1964).

Sutton et al. (1999b) observed that SMDPs enable the modeling of temporal abstractions of actions. One goal of hierarchical RL is to define action abstractions that subsume multiple atomic actions (Khetarpal et al., 2022; Eppe et al., 2022). Sutton et al. (1999b) provide a framework to unify different approaches to action abstraction, called the **options framework**. Formally, an option is a 3-tuple $(\mathcal{I}, \pi, \beta)$ with an initiation set \mathcal{I} , a policy π and a termination function β . The initiation set $\mathcal{I} \subseteq \mathcal{S}$ defines the subset of states from which the option can be taken. Once the option is initiated, the policy π selects actions that are executed until the termination condition β is met. The termination function β can be defined as a mapping from states to a termination probability, i.e. $\beta : \mathcal{S} \rightarrow [0, 1]$. However, to avoid getting stuck in an option, it is useful to include a timeout-condition in β , that terminates the option after a fixed time. Sutton et al. (1999b) showed that if we replace the action set \mathcal{A} of a given MDP with a set of options, the resulting decision process is an SMDP.

2.2.3 Partially Observable Processes

While MDPs can model a variety of problems, ranging from board games to robotic simulations, the assumption that the state is fully observable typically does not hold for more realistic processes. In the real world the observation an agent receives about its environment is noisy and incomplete. For such problems, **Partially observable Markov Decision Processes** (POMDP) (Lovejoy, 1991; Kaelbling et al., 1998) have been proposed (illustrated in Fig. 2.3). A POMDP can be formally described by a 7-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma, \Omega, O,)$. Thus, the POMDP extends the MDP tuple (cf. Sec. 2.2.1) by a set of observations Ω and an observation function O . The main difference from MDPs is that the agent does not observe the full state of the world $\mathbf{s}_t \in \mathcal{S}$ at each time step. Instead, after executing an action $\mathbf{a}_{t-1} \in \mathcal{A}$ and reaching the next state $\mathbf{s}_t \in \mathcal{S}$, the agent receives an observation $\mathbf{o}_t \in \Omega$ based on the observation function O . The observation function $O(\mathbf{o}_t | \mathbf{s}_t, \mathbf{a}_{t-1})$ describes the probability of observing $\mathbf{o}_t \in \Omega$ when taking action \mathbf{a}_{t-1} and reaching state \mathbf{s}_t .

Kaelbling et al. (1998) observed, that every POMDP can be transformed into an MDP by estimating beliefs about the probable real state of the world \mathbf{s}_t from past observations and actions. In the resulting **Belief MDP**, the Markov Property holds (Kaelbling et al., 1998). However, for the exact inference of the belief state, knowledge of transitions and

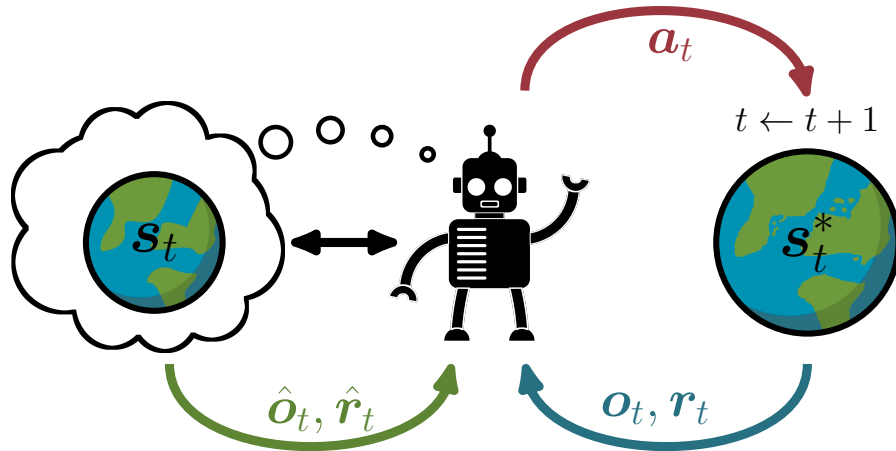


Figure 2.3: Partial observability and models: In POMDPs the true state s_t^* of the environment is hidden and the agent only receives incomplete observations o_t . Thus, the agent needs to maintain an internal belief s_t about the world. In model-based approaches, e.g. active inference (Parr et al., 2022), the agent learns an internal model that attempts to mirror the true generative process of the environment (green line).

reward functions of the underlying POMDP is required (Ni et al., 2022). To make matters worse, even if the underlying POMDP is known, solving the problem optimally is often computationally intractable (Hauskrecht, 2000).

Instead of inferring the exact state of the POMDP, a contemporary approach in deep RL is to use recurrent neural networks (RNNs) (Wierstra et al., 2007; Hausknecht & Stone, 2015; Igl et al., 2018; Ha & Schmidhuber, 2018; Hafner et al., 2020a, 2023; Ni et al., 2022). RNNs embed past inputs into a hidden state or *latent state*. When applied to POMDPs, the latent state can be used as a surrogate for the state information in the policy and the value function. As a result, the problem can be treated as an MDP to apply standard RL techniques, e.g. optimizing a policy using policy gradients (Wierstra et al., 2007).

2.2.4 Active Inference

MDP frameworks and their derivatives simplify sequential decision making by expressing rewards as numeric values given to the agent by the environment. However, in biological

agents there are no external rewards that simply fall from the sky.^{2.8} Biological behavior seems to be driven by a multitude of objectives, including homeostatic needs, e.g. hunger or thirst, epistemic motivation, e.g. curiosity or desire to play, and social motivations, e.g. approval or affiliations.

The **Free Energy Principle** (FEP) has the ambitious goal of finding a unifying principle to explain the mechanisms of the brain and its changes during development and evolution (Friston et al., 2006; Friston, 2009, 2010). Thus, it also attempts to provide an objective based on which biological agents select their actions. The following summary of the FEP and active inference is mainly based on Friston et al. (2011), Sajid et al. (2021a), and Parr et al. (2022). Please note that for the FEP there exist a multitude of formulations with different notations and simplifications. I discuss the notable differences in the literature in Suppl. A.2.3.

The FEP assumes that a biological agent and its environment can be modeled by a POMDP (see Sec. 2.2.3). The FEP postulates that all adaptive agents strive to minimize *surprise* (Friston et al., 2011; Sajid et al., 2021a). For an observation $\mathbf{o}_t \in \mathcal{O}$, surprise $S(\mathbf{o}_t)$ can be defined as the negative log-likelihood of this particular observation, i.e.

$$S(\mathbf{o}_t) = -\log P(\mathbf{o}_t). \quad (2.4)$$

How would an agent know if a certain observation is surprising? According to the FEP, the agent maintains internal belief states^{2.9} $\mathbf{s}_t \in \mathcal{S}$ about the world and a generative model f_ϕ with adaptive parameters ϕ . At every point in time t the belief state \mathbf{s}_t attempts to approximate the unobservable true state \mathbf{s}_t^* of the world. Just like the generative process of the world progresses to a new state and generates a new observation upon each transition (POMDP functions O and T in Sec. 2.2.3), the generative model $f_\phi(\mathbf{o}_t, \mathbf{s}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1})$ predicts the next state and observation based on past experiences (green line in Fig. 2.3). Surprise can then be estimated via

$$S'(\mathbf{o}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}) = -\log \mathbb{E}_{\mathbf{s}_t \sim \mathcal{S}} f_\phi(\mathbf{o}_t \mid \mathbf{s}_t, \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1}). \quad (2.5)$$

However, estimating surprise in this way is not trivial due to the potentially intractable expectation over states (Sajid et al., 2021a). The FEP proposes a different and more elegant solution, namely to minimize variational *free energy* (FE) (Friston et al., 2011).

^{2.8}In the last decades, there has been converging evidence for neural correlates of reinforcement learning in the brain (Niv, 2009), e.g. linking the phasic activity of dopaminergic neurons with reward predictions (Schultz et al., 1997). Nonetheless, it is not clear how our own body distributes these internal reward signals and whether more complex motivations, e.g. curiosity, are controlled via dopaminergic signals.

^{2.9}In the remainder of this chapter \mathbf{s}_t will refer to the internal state of the agent and not the true state of the environment. I do this to align the notation with the active inference literature, e.g. Sajid et al. (2021a). Additionally, since the true state of the world is assumed to be unknown, it is rarely mentioned.

FE is an upper bound on the surprise, and thus, by minimizing FE, the agent implicitly minimizes surprise (Friston, 2010). To define FE, a recognition density $q_{\boldsymbol{\vartheta}}$ with adaptable parameters $\boldsymbol{\vartheta}$ is required. The recognition density $q_{\boldsymbol{\vartheta}}(\mathbf{s}_t \mid \mathbf{a}_{1:t-1})$ attempts to approximate the current belief state \mathbf{s}_t given past actions $\mathbf{a}_{1:t-1}$. In most formalizations of the FEP (e.g. Friston et al., 2015, 2016; Sajid et al., 2021a; Schwöbel et al., 2018), the approximate posterior $q_{\boldsymbol{\vartheta}}$ is not conditioned on past observations unlike the true posterior density $p(\mathbf{s}_t \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})$.^{2.10}

Given past observations and actions, FE at a certain point in time t can be computed as

$$\text{FE}(\mathbf{o}_{1:t}, \mathbf{a}_{1:t-1}, \boldsymbol{\vartheta}) = \underbrace{\text{KL}[q_{\boldsymbol{\vartheta}}(\mathbf{s}_t \mid \mathbf{a}_{1:t-1}) \parallel p(\mathbf{s}_t \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})]}_{\text{evidence bound}} - \underbrace{\log p(\mathbf{o}_t)}_{\text{surprise}}, \quad (2.6)$$

with KL denoting the Kullback-Leibler divergence. Free energy is composed of two terms, the evidence bound described by the KL-term, and the surprise term from Eq. 2.4. Since the KL divergence is always ≥ 0 , Eq. 2.6 should make it clear that FE is strictly larger or equal to surprise. Furthermore, we can see that FE should be minimized with respect to the parameters $\boldsymbol{\vartheta}$, when the approximate posterior $q_{\boldsymbol{\vartheta}}$ converges to the true posterior. However, the agent cannot optimize Eq. 2.6 directly because the true posterior $p(\mathbf{s}_t \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})$ is unknown and surprise is still potentially intractable. However, we can use the generative model $f_{\boldsymbol{\phi}}$ and the recognition density $q_{\boldsymbol{\vartheta}}$ to estimate these quantities:

$$\begin{aligned} \text{FE}'(\mathbf{o}_{1:t}, \mathbf{a}_{1:t-1}, \boldsymbol{\vartheta}, \boldsymbol{\phi}) &= \text{KL}[q_{\boldsymbol{\vartheta}}(\mathbf{s}_t \mid \mathbf{a}_{1:t-1}) \parallel f_{\boldsymbol{\phi}}(\mathbf{s}_t \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})] \\ &\quad - \log \mathbb{E}_{\mathbf{s}_t \sim q_{\boldsymbol{\vartheta}}} [f_{\boldsymbol{\phi}}(\mathbf{o}_t \mid \mathbf{s}_t, \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t-1})]. \end{aligned} \quad (2.7)$$

This reformulation of FE now only depends on the past experience of an agent, its generative model $f_{\boldsymbol{\phi}}$, and the recognition density $q_{\boldsymbol{\vartheta}}$. Thus, it can be directly evaluated.

The FEP proposes that biological agents constantly strive to minimize free energy on multiple time scales through perception, action, and learning (Friston, 2010). During perception, the agent infers $\boldsymbol{\vartheta}$ and $\boldsymbol{\phi}$ on a short-time scale, leading to short-term adaptations of the densities $q_{\boldsymbol{\vartheta}}$ and $f_{\boldsymbol{\phi}}$ to explain the underlying causes of current observations. During life-long learning, the agent forms long-term adaptations of the parameters $\boldsymbol{\vartheta}$ and $\boldsymbol{\phi}$, akin to using Eq. 2.7 as a loss function for training the weights of a machine learning model.^{2.11} Heavily simplified, FEP proposes that the agent attempts to align its internal models

^{2.10}In principle, the approximate posterior $q_{\boldsymbol{\vartheta}}$ can also be conditioned on observations, as discussed in more detail in Suppl. A.2.3.

^{2.11}Variants of this objectives are also used in contemporary deep RL to train internal world models of model-based RL agents (Hafner et al., 2019a, 2020a, 2023).

with the outside world. However, agents can not only align world and internal models by adjusting the model parameters - they can also use their actions to align the outside world to fit their internal models (Friston et al., 2009; Hafner et al., 2020b; Parr et al., 2022). This process is called **active inference**. Here, free energy can be used as an objective to optimize a policy.

For active inference, we need to modify Eq. 2.7, because it only considers past experiences and does not take the future into account. We can do this via two steps (Sajid et al., 2021a): First, we include expectations over the beliefs about future states and observations. Second, we replace the state posterior $f_\phi(\mathbf{s}_t \mid \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1})$ in Eq. 2.7 with a prior over preferred states $g(\mathbf{s}_t)$. The prior preferences g can express desired goal states on an arbitrary level of abstraction, ranging from clear homeostatic needs, such as thirst, to abstract goals, such as physical safety. Incorporating these changes, we can formulate *expected free energy* (EFE) over a future time horizon K as

$$\text{EFE}(\pi, K, \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1}) = \frac{1}{K} \sum_{k=t}^{t+K} \underbrace{\mathbb{E}_{\mathbf{a}_{t:k} \sim \pi} \text{KL}[q_\theta(\mathbf{s}_{k+1} \mid \mathbf{a}_{1:k}) \parallel g(\mathbf{s}_{k+1})]}_{\text{predicted divergence}} \quad (2.8)$$

$$+ \underbrace{\mathbb{E}_{\mathbf{a}_{t:k} \sim \pi, \mathbf{s}_{t:k} \sim q_\theta, \mathbf{o}_{t:k} \sim f_\phi} (\mathcal{H}[f_\phi(\mathbf{o}_{k+1} \mid \mathbf{s}_k, \mathbf{o}_{1:k}, \mathbf{a}_{1:k})])}_{\text{predicted uncertainty}},$$

with \mathcal{H} denoting entropy. An active inference agent selects its policy in order to minimize EFE. This can be realized by using Eq. 2.8 as an objective for sequential decision making (e.g. for MPC by replacing J from Eq. 2.3).

Policy optimization based on active inference entails two behavioral consequences based on the two terms of EFE: The first term in Eq. 2.8 describes how much the agent expects to diverge from the distribution of desired future states $g(\mathbf{s}_k)$. Inferring actions to minimize this term promotes greedy, goal-directed behavior. The second term encourages epistemic, curious behavior, by incentivizing the agent to seek out states to minimize uncertainty about the future. Thus, active inference offers a principle to tackle the **exploration-exploitation dilemma** of RL (Sutton & Barto, 2018), which raises the question when an RL agent should stop exploring its environment and focus solely on reward maximization. Active inference encourages both the exploitation of past knowledge via the first term and exploration via the second term. Active inference showed strong results in idealized settings of this dilemma, especially for dynamical problems (Marković et al., 2021).

Various behavioral tendencies of humans and other animals have been modeled using active inference (see Da Costa et al., 2020 for an overview). Nonetheless, the FEP is a controversial framework. The FEP has been criticized for being overly general, containing little explanatory power, and its falsifiability has been questioned (Colombo & Wright, 2021). Furthermore, many formulations of active inference use open-loop policies

or simple action sequences (e.g. [Friston et al., 2015, 2016](#); [Sajid et al., 2021a](#)), which seems unrealistic given the highly adaptive behavior shown by many animals. Despite the advanced mathematical formulations, active inference is often employed in simple, low-dimensional and discrete settings (e.g. [Friston et al., 2015, 2016](#)). Recently, there has been increasing research towards combining active inference with modern deep learning techniques (e.g. [Fountas et al., 2020](#); [Sancaktar et al., 2020](#)). However, it is unclear whether deep active inference scales as well to high-dimensional problems as simpler deep RL approaches ([Da Costa et al., 2022](#)).

3

Towards Sequential Decision Making with Events

Previously, I reviewed experimental evidence and theoretical considerations that humans encode their experience in terms of hierarchically organized event representations. This chapter analyzes these insights through the lens of sequential decision making. The goal is to derive an integrative framework to model event perception and hierarchical action generation in the language of Markov Decision Processes. This framework will guide the design of all systems presented in the remainder of this thesis.

3.1 THICK Markov Decision Processes

Let us consider a problem that can be described by a POMDP (see Sec. 2.2.3). Due to the partial observable nature of the problem, an agent needs to maintain internal beliefs about the unfolding processes. Humans seem to encode processes they experience in terms of hierarchically organized models of events (see Sec. 2.1). Inspired by human event cognition, let us attempt to solve such problems using a hierarchy of Markov decision processes (MDPs). In this section, I introduce an algorithm that describes how a decision making hierarchy can be developed such that the resulting abstractions satisfy a number of properties of human event encodings (see Box 1). This algorithm is named **Temporal**

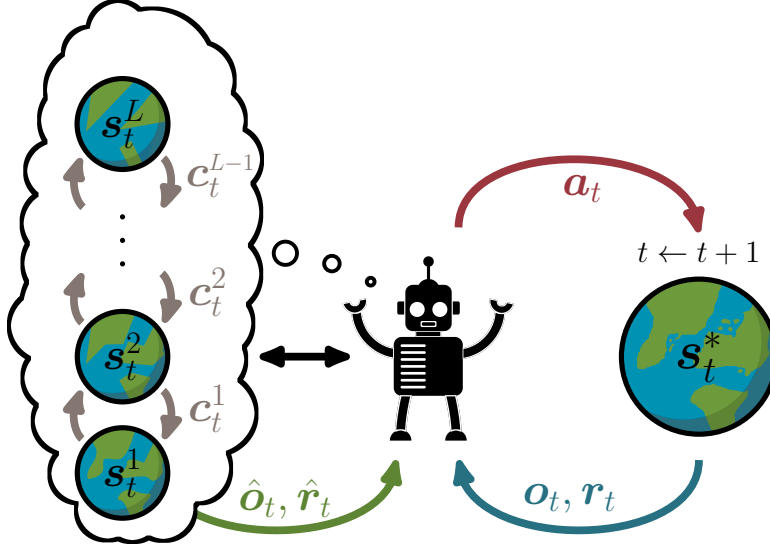


Figure 3.1: THICK MDPs model a POMDP via L different MDPs \mathcal{M}^l for each level l . The levels interact bidirectionally via the event context c^l . In a bottom-up direction, a context change on level l prompts a state transition in level $l + 1$. In the top-down direction, the level $l + 1$ provides context c^l as a goal for planning on level l .

Hierarchies from Invariant Context Kernels (THICK)^{3.1}. Our framework is called THICK MDP.

A THICK MDP encodes the underlying POMDP through L MDPs, as illustrated in Fig. 3.1. The decision process \mathcal{M}^l at each level l is defined with $1 \leq l \leq L$, as $\mathcal{M}^l = (\mathcal{S}^l, \mathcal{A}^l, T^l, R^l)$, with the set of states \mathcal{S}^l , set of actions \mathcal{A}^l , transition function T^l , and reward function R^l . The different processes \mathcal{M}^l represent **temporal abstractions** of the true processes of the environment maintained internally by the agent. Thus, with increasing levels of abstraction l , processes should cover a longer durations per transition. Each level l operates on a level-dependent time scale t^l . Level-dependent time scales t^l are slower than or equal to the *objective* time scale t of the ground-truth POMDP. To relate the level-dependent time t^l to the objective time t , we assume that the agent maintains an “internal clock function” Ξ , such that $\Xi(t^l) = t$. The function Ξ allows the agent to look up the objective time t given the level-dependent time t^l .

Now that the general setup and notation have been established, let us start to integrate

^{3.1}In philosophy, the term ‘thickness’ refers to concepts that combine descriptions with an evaluative context (Roberts, 2013). In THICK MDPs, each level attempts to represent the true decision process of the world while also encoding contextual information to interact with adjacent levels in the hierarchy.

inductive biases based on event cognition into the framework. For this, I will specify, and thereby constrain, the structure of the overall decision process. Refer to [Box 1](#) for a summary of the proposed properties of event representations.

1. Temporal persistence: It has been proposed that humans encode their experience in terms of event representations, which show stable activations over extended time segments ([Zacks et al., 2007](#)). To integrate event representations into the state belief of the agent, we can split the state $\mathbf{s}_{t^l}^l$ into two parts, an *event-encoding context* $\mathbf{c}_{t^l}^l \in \mathcal{C}^l$ and the residual^{3.2} state information $\mathbf{z}_{t^l}^l \in \mathcal{Z}^l$. Thus, $\mathbf{s}_{t^l}^l = (\mathbf{c}_{t^l}^l, \mathbf{z}_{t^l}^l)$ is a tuple.

Now we can align the event context $\mathbf{c}_{t^l}^l$ with what has been proposed for human event representations. Activations of event models seem to persist over time. Along similar lines, I propose that at a certain level l the event code $\mathbf{c}_{t^l}^l$ can remain constant over multiple time steps. However, $\mathbf{c}_{t^l}^l$ should also change systematically and somewhat predictably. Formally, we can state that for the MDP \mathcal{M}^l , we want the next context changes for a given state and policy to be semi-Markovian (detailed in [Suppl. A.3.1](#)). This means that if we ignore the time points without context changes, the process still fulfills the Markov property. These sparse but systematic transitions in contexts are a crucial prerequisite for the following assumptions because this allows us to define a higher-level process based on lower-level context transitions.

2. Nested time scales: Experimental evidence suggests that events are encoded in a partonomy with nested time scales ([Zacks & Tversky, 2001](#)). How can we realize this property for the decision making process? Assumption 1 constrains that the context $\mathbf{c}_{t^l}^l$ at level l can remain constant for multiple time steps t^l . I propose using the changes in context $\mathbf{c}_{t^l}^l$ as an adaptive time scale for the next level $l + 1$ up the hierarchy, illustrated in [Fig. 3.2](#). In other words, a transition in \mathcal{M}^{l+1} only occurs when the event context $\mathbf{c}_{t^l}^l$ of \mathcal{M}^l changes. Thus, the time scale t^{l+1} at level $l + 1$ is recursively defined over a time scale t^l of the lower level l . This results in a nested partonomy of contexts ([Fig. 3.2](#)).

How can we formalize this relationship between levels? At a certain level l we can determine the next point in time t^l for which the context changes, using a function τ with

$$\tau(t^l) = \min(\{\tau \mid \tau > t^l \wedge \mathbf{c}_{\tau}^l \neq \mathbf{c}_{t^l}^l\}). \quad (3.1)$$

Thus, $\tau(t^l)$ maps the time t^l to the next point in time $\tau(t^l)$ for which the event context $\mathbf{c}_{t^l}^l$ changes. [Figure 3.2](#) illustrates τ as a [red](#) arrow. The level $l + 1$ transitions only at the time steps defined by $\tau(t^l)$. According to this rule, we can define the time scales t^{l+1} in

^{3.2}The residual state information could encode everything not captured by $\mathbf{c}_{t^l}^l$ such that \mathcal{M}^l is still Markov. This could be static information about the current scene or aspects irrelevant to the ongoing event.

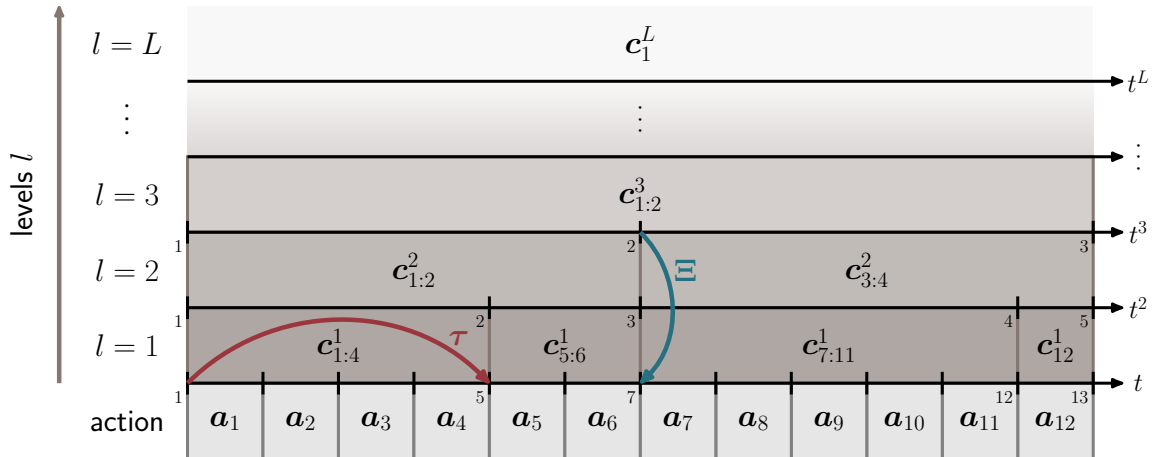


Figure 3.2: Context partitioning in THICK MDPs: Each level l has a different time scale t^l , with slower time scales at higher levels. The clock function Ξ (blue) maps level-dependent time scales t^l to the objective time t . On the objective time scale the contexts form a partitioning: While on level l one context persists, on a lower level $l - 1$ multiple context changes may occur. The function τ determines the next point in time with a context change (red).

terms of the objective time scale as

$$\Xi(t^{l+1} + 1) \doteq \Xi(\tau(t^l)), \quad \forall l : \Xi(t^{l+1}) = \Xi(t^l), \quad (3.2)$$

with the clock function Ξ determining the objective time (blue arrow in Fig. 3.2). At the lowest level, we set $t^1 = t$ to the objective time of the POMDP.

Informally, this means that the process at level $l + 1$ transitions at the time scale of context changes or event boundaries of the lower level l . For example, at level l the event `<grab teapot>` may take two steps, namely the events `<reaching>` and `<grasping>` of the lower level $l - 1$, as shown in Fig. 3.3. This automatically fulfills a handful of desiderata: For example, coarsely segmented event boundaries (i.e. context changes on a higher level $l + 1$) per definition must fall onto finely segmented event boundaries (i.e. context changes on a lower level $\leq l$). Furthermore, for a given (objective) period of time, events at a lower level change more frequently than events at a higher level.

3. Model-based predictions: An important property of event models is their prediction of future perceptions (Zacks et al., 2007). How can we integrate model-based predictions into the overall framework? As usual when modeling POMDPs via MDPs (Kaelbling et al., 1998), we can split the transition function T^l into two functions U^l and

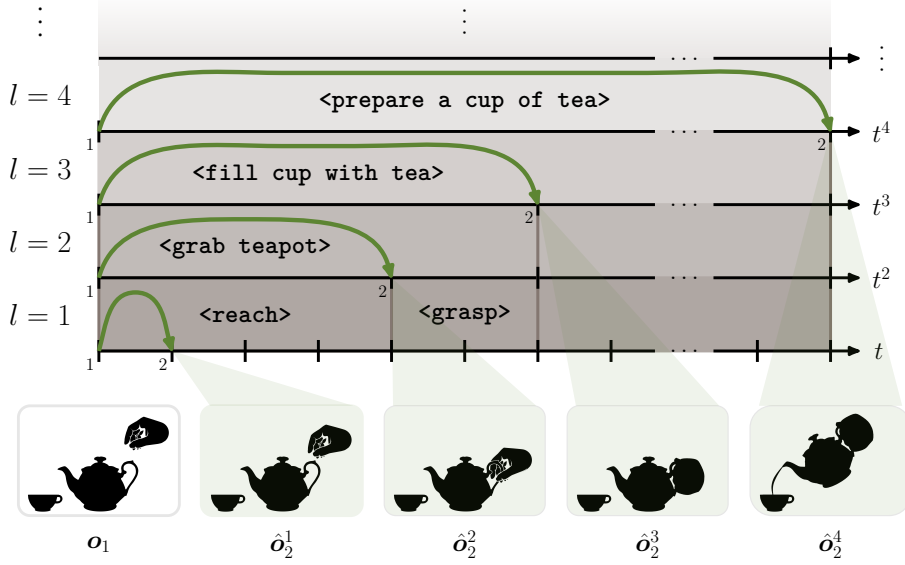


Figure 3.3: Models in THICK MDPs: When using models $f_{\phi}^l(\mathbf{o}_{t^l+1} | \mathbf{s}_{t^l}^l, \mathbf{a}_{t^l}^l)$ the model at level l predicts the observation $\hat{\mathbf{o}}_{t^l+1}^l$ at the next time $t^l + 1$ when the context of level $l - 1$ changes next. The lowest level simply predicts the next objective time step $t + 1$. Here this is illustrated for the example of preparing a cup of tea (cf. Fig. 2.1).

M^l , with

$$T(\mathbf{s}_{t^l+1}^l | \mathbf{s}_{t^l}^l, \mathbf{a}_{t^l}^l) = \sum_{\mathbf{o}_{t^l}^l \in \Omega^l} U^l(\mathbf{s}_{t^l+1}^l | \mathbf{s}_{t^l}^l, \mathbf{a}_{t^l}^l, \mathbf{o}_{t^l}^l) M^l(\mathbf{o}_{t^l+1}^l | \mathbf{s}_{t^l}^l, \mathbf{a}_{t^l}^l). \quad (3.3)$$

For each level l , the function U^l updates the belief, whereas M^l mirrors the true observation function (O in Sec. 2.2.3) of the POMDP conditioned on beliefs $\mathbf{s}_{t^l}^l$. We can approximate M^l by learning a forward model $f_{\phi}^l(\mathbf{o}_{t^l+1}^l | \mathbf{s}_{t^l}^l, \mathbf{a}_{t^l}^l)$ with parameters ϕ . Through this model, the agent is able to predict the next observation $\mathbf{o}_{t^l+1}^l$. Thus, model-based predictions of future observations can seamlessly be integrated into our framework.

Note that, due to Assumption 2, the level-dependent time t^{l+1} increases at the rate of context changes at the lower level l . This means that at a certain level l , the model f_{ϕ}^l predicts the observation at the next context change or at the event boundaries of the lower level. For example, on a level l the event <fill cup with tea> may take three steps, which correspond to the events <grab teapot>, <lift teapot>, and <pour tea into cup> of lower level $l - 1$. A model at level l would predict at the same temporal resolution. Thus, it would first predict an observation where the hand grasps the teapot (transition from <grab teapot> to <lift teapot>), then an observation of the teapot over the cup

(transition from <lift teapot> to <pour tea into cup>), etc. As a result, models on higher levels learn to predict sub-goal-like situations that need to be accomplished to trigger an event transition, e.g. having grasped the teapot is the first step towards filling tea in a cup. Model-based predictions for the tea example are visualized in Fig. 3.3.

4. Action-Goal Association: There is an increasing line of evidence for theories that equate the representations involved in event perception with the representations used for action planning (Hommel et al., 2001; Butz, 2016; Kuperberg, 2021; Cooper, 2021). For THICK MDPs, I propose that possible next event codes $\mathbf{c}_{t^{l+1}+1}^l \in \mathcal{C}^l$ on a certain level l can serve as actions $\mathbf{a}_{t^{l+1}}^{l+1} \in \mathcal{A}^{l+1}$ on the next higher level $l+1$, i.e. $\mathcal{A}^{l+1} = \mathcal{C}^l$. Crucially, only contexts to which the lower level can directly transition from the current situation are available as actions (detailed in Suppl. A.3.2). As a special case, on the lowest level $l=1$ we set $\mathcal{A}^1 = \mathcal{A}$ to the action set of the POMDP. Of course, this creates a series of inter-dependencies between the levels. For example, a policy π^l for level l outputs a probability over next event codes $\mathbf{c}_{t^{l+1}}^{l-1} \in \mathcal{C}^{l-1}$. Similarly, the transition function T^l is conditioned on event codes $\mathbf{c}_{t+1}^{l-1} \in \mathcal{C}^{l-1}$.

It may seem counterintuitive that the state information of one MDP serves as the action of another MDP. However, whenever the contexts encode temporally extended processes that can be triggered by the agent, this not only aligns with the everyday notion of actions but also fits more formal criteria of what constitutes an action representation in artificial agents (Zech et al., 2019). For example, the event <pour tea into cup> is an action of the higher level event <prepare a cup of tea>. This nested relationship even holds for events triggered by an agent in which no motor commands are involved: For example, the event <the ball flies into the basket> is one “action” of the higher level event <score a basket ball shot>.

Summary: I have outlined the properties that contribute to a hierarchically nested decision process. Each level l transitions at a level-dependent time scale t^l . During transitions the state $\mathbf{s}_{t^l}^l$ changes, however, parts of the state, the event context $\mathbf{c}_{t^l}^l$ tends to remain constant over extended segments in time. The time scales of the hierarchy are nested, and a transition at level $l+1$ occurs only when the context $\mathbf{c}_{t^l}^l$ of the lower level changes. The contexts $\mathbf{c}_{t^l}^l$ serve as the actions at the level $l+1$ that determine the transitions. Thus, for higher level up the hierarchy, temporally extended processes are more and more compressed into context codes. We also indirectly provided an algorithm to develop such a hierarchy of MDPs, starting from a single MDP in three steps: (1.) Separate event-specific contextual knowledge from residual state information of every state. (2.) Recursively define the time scale of the next level of the hierarchy based on changes in the event codes, and (3.) replace the actions of the next level with event codes of the lower level. This process can be repeated L times to form a hierarchy. We call this algorithm **Temporal Hierarchies from Invariant Context Kernels (THICK)**.

3.2 Hierarchical Planning

Temporal abstractions have the potential to drastically help agents to plan complex behavior over long time horizons. How could an agent plan their behavior within a THICK MDP? Let us assume finite-horizon planning, where each level l plans up to a level-dependent horizon K^l . Hierarchical action planning typically assumes a top-down scheme (Cooper, 2021), where a higher level affects the planning process at lower levels. Thus, starting at the highest level L we simply set the policy optimization objective to J^L with

$$J^L(\mathbf{s}_{t^L}^L, \pi^L, K^L) = \mathbb{E}_{\pi^L} \sum_{t=t^L}^{K^L} R^L(r_t^L \mid \mathbf{s}_t^L, \mathbf{a}_{t+1}^L, \mathbf{s}_{t+1}^L). \quad (3.4)$$

Note that this corresponds to the finite horizon planning objective, defined in Eq. 2.3. The only notable difference is that, in this case, the actions \mathbf{a}_t^L correspond to a desired next event code of the next lower level, i.e. $\mathbf{a}_t^L = \mathbf{c}_{t+1}^{L-1}$.

At any lower level $l < L$, we need to define a planning objective J^l that takes into account the plans of the higher levels. One potential way to express this is to guide each level l to produce event codes $\mathbf{c}_{t^l}^l$ that align with the policy π^{l+1} of the next level $l+1$ up the hierarchy. We can formulate this, for example, using the Kullback-Leibler divergence (KL), a prominent measure for distribution divergence (Hafner et al., 2020b). This gives us the objective J^l for every level l as

$$J^l(\mathbf{s}_{t^l}^l, \pi^l, \pi^{l+1}, K^l) = \mathbb{E}_{\pi^l} \underbrace{\sum_{t=t^l}^{K^l} R^l(r_t^l \mid \mathbf{s}_t^l, \mathbf{a}_t^l, \mathbf{s}_{t+1}^l)}_{\text{external rewards}} - \underbrace{\kappa \text{KL}(\pi^{l+1}(\mathbf{c}_{t+1}^l \mid \mathbf{s}_t^{l+1}) \parallel T^l(\mathbf{c}_{t+1}^l \mid \mathbf{s}_t^l, \mathbf{a}_{t+1}^l))}_{\text{intrinsic rewards}}, \quad (3.5)$$

with $T^l(\mathbf{c}_{t+1}^l \mid \mathbf{s}_t^l, \mathbf{a}_{t+1}^l)$ the transition function of contexts.^{3.3} Here, the objective is composed of two terms: The first term encourages the policy π^l to maximize external rewards provided through the environment. The second term punishes π^l for producing event codes $\mathbf{c}_{t^l}^l$ that diverge from the event codes planned by the higher-level policy π^{l+1} . The new hyperparameter κ controls the trade-off between extrinsic and intrinsic rewards. With such a scheme, an agent can plan behavior in a top-down matter, with the higher-level policy selecting events that serve as goals for the lower level. The policy π^1 on the lowest level translates this into concrete actions.

^{3.3}This is the transition function T^l at level l marginalized over the next states, i.e. $T^l(\mathbf{c}_{t+1}^l \mid \mathbf{s}_t^l, \mathbf{a}_{t+1}^l) = \sum_{\mathbf{z}_{t+1}^l \in \mathcal{Z}} T^l(\mathbf{c}_{t+1}^l, \mathbf{z}_{t+1}^l \mid \mathbf{s}_t^l, \mathbf{a}_{t+1}^l)$.

Our event-based planning scheme can also be seamlessly integrated with active inference (see Sec. 2.2.4), as previously proposed by Butz (2016). In active inference, an agent infers its policy by attempting to minimize expected free energy (EFE) (cf. Eq. 2.8). To compute EFE, the agent requires a generative model f_ϕ and a recognition density q_θ . As outlined before, THICK MDPs can learn generative models f_ϕ^l along every level l of the hierarchy. The same is true for recognition densities q_θ^l . In our planning objective in Eq. 3.5, we can simply substitute the reward function R^l on each level l with the EFE for this particular level. When minimizing EFE, the agent not only strives to minimize divergence from desired states but also aims to minimize uncertainty about future observations on all levels of the hierarchy.

3.3 Examples

I have outlined THICK MDPs, a theoretical framework for hierarchical decision making. As this integrative framework is inspired by multiple theories of cognition, I hypothesize that THICK MDPs are not only **useful for hierarchical problem solving**, but are **suitable to model the behavior of humans** and other biological agents. More concretely, THICK MDPs allow for both planning complex goal-directed behavior and showing human-like behavioral tendencies for which a non-hierarchical agent would require a very long planning horizon. In the following, I provide two examples to support this claim by illustrating (1.) how complex action sequences can be planned and (2.) how human-like behavior can emerge.

Example 1: Planning to drink tea Let us assume that an agent sits at a table with a full teapot and an empty cup and wants to drink the tea. In a classic POMDP setting, a reward could be +1 once tea is consumed. In the active inference formulation, the goal could be expressed as a homeostatic state without any thirst and sufficient warmth (via its prior preference g). Furthermore, let us assume that the agent is equipped with a THICK MDP with four levels l as shown in Fig. 2.1 or Fig. 3.3.

To achieve its goal of drinking tea, the agent first plans at the highest level $l = 4$. Here, the agent could come up with a policy π^4 , that plans to reach the event $\mathbf{c}^4 = \langle \text{prepare a cup of tea} \rangle$. The first action, or the lower level event \mathbf{c}^3 , within this event is $\langle \text{fill cup with tea} \rangle$. Thus, at the level $l = 3$, the agent plans to trigger this event. The policy π^3 might output $\langle \text{grab teapot} \rangle$ as a suitable first action, or the desired next low-level event \mathbf{c}^2 . This process continues recursively, such that $\mathbf{c}^1 = \langle \text{reach for teapot} \rangle$ is selected by policy π^2 . Ultimately, at level $l = 1$, the policy π^1 generates actions \mathbf{a}_t that align with the $\langle \text{reach for teapot} \rangle$ -event and move the agent’s hand towards the teapot.

By repeating this planning process, the agent is able to plan and conduct the complex behavior of filling tea in a cup and drinking from it. The policy π^1 of level $l = 1$ needs

to output a new action \mathbf{a}_t at each time step t . At all other levels l , the agent only needs to replan once the lower-level context \mathbf{c}_t^{l-1} changes. For example the <fill cup with tea>-event is active until the <pour tea in cup>-event on the lower level is over, either by successfully completing it or accidentally transitioning to the <drop teapot>-event.

Other reward formulations are also possible. For example, the active inference formulation additionally uses a term that aims at minimizing the expected uncertainty about the future. Including this aspect in the planning objective could result in more risk-aware behavior. For example, let us assume that the agent lifts the teapot and it is much lighter than expected. This creates a high uncertainty, for example, on level $l = 3$, because there might not be enough tea in the pot to <fill cup with tea>. As a result, the agent could plan to first execute the <shake teapot>-event at level 2, to receive auditory or proprioceptive feedback on how much tea is left in the pot.

Could such behavior also be achieved by optimizing a single-level policy? Although in principle it is possible, the planning horizon of the policy optimization would need to take all steps into account to reach the final goal. Thus, a very long planning horizon is needed. Although reinforcement learning (RL) aims to maximize reward over an infinite horizon (cf. Eq. 2.2), it is infamous for requiring large amounts of training data (Lake et al., 2017). For model-predictive control (MPC), long planning horizons are often impossible, since model errors can accumulate very quickly, leading to suboptimal or erroneous behavior. In the case of THICK MDP, the low-level policy has a much simpler objective: In the absence of external rewards, the lowest level needs to only achieve the subgoal proposed by level 2 (see Eq. 3.5). At all other levels l , the goal can be achieved by planning just a few steps, i.e. planning with a short horizon K^l .

Example 2: Anticipatory gaze behavior Next, let us consider whether THICK MDPs can be used to model human behavior. More specifically, we focus on gaze behavior. One robust effect of human gaze behavior is the *goal-anticipatory gaze shift*^{3,4} (Gredebäck & Falck-Ytter, 2015). Goal-anticipatory gaze shifts describe the phenomenon that when observing a movement, e.g. a reach, humans tend to shift their gaze from the moving entity, e.g. the hand, to the goal of a movement, e.g. the target of a reach, before the motion is concluded. This gaze behavior reliably arises in various settings (reviewed in Chap. 4), e.g. when performing goal-directed behavior (Land et al., 1999; Johansson et al., 2001) or when observing the behavior of others (Flanagan & Johansson, 2003).

How can such behavior be modeled using THICK MDPs? Consider an agent, illustrated in Fig. 3.4a, equipped with a two-level THICK MDP. For simplicity, the only actions \mathcal{A}

^{3,4}Synonyms in the literature are *anticipatory gaze shift* (Paulus, 2011), *predictive gaze shift* (Gredebäck & Falck-Ytter, 2015), or *goal-predictive gaze shifts* (Elsner & Adam, 2021). The act of looking at a location in anticipation is also called *proactive gaze* (Flanagan & Johansson, 2003), in contrast to *reactive gaze* in which the gaze follows a moving entity.

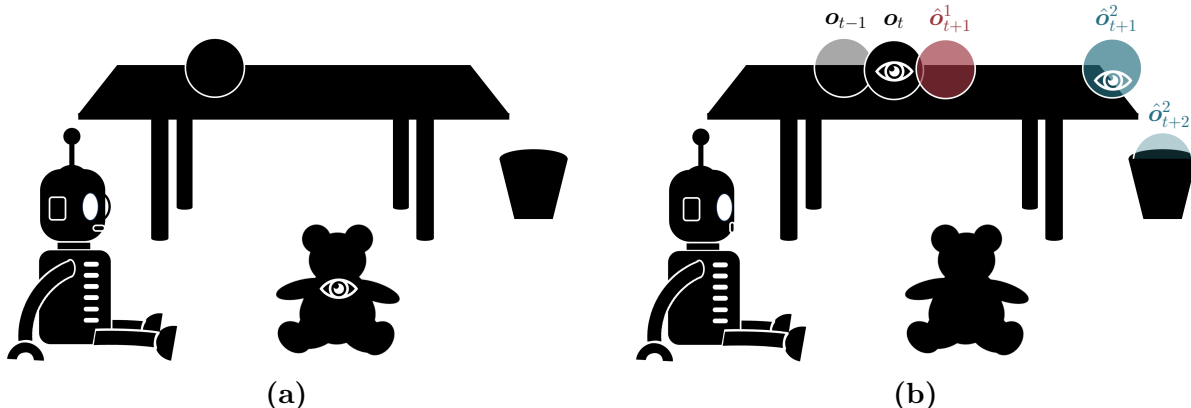


Figure 3.4: Goal-anticipatory gaze in THICK MDPs through active inference: Usually, the agent will infer its gaze to seek out desirable observations, e.g. by looking at a teddy bear (a). However, under high uncertainty, e.g. when a ball unexpectedly starts to move, the agent attempts to minimize uncertainty on multiple levels l of the model hierarchy (b). On a low level $l = 1$ (red) the agent may predict immediate ball positions in \hat{o}_{t+1}^1 . On a higher level $l = 2$ (blue), the agent might predict ball positions at the next event boundaries, e.g. that the ball goes from rolling to falling in \hat{o}_{t+1}^2 . In order to minimize uncertainty about these event boundaries, the agent might shift its gaze towards the location of the next event boundary, i.e. the edge of the table.

available to the agent are eye movements. Low-level events c^1 include gaze behavior such as <track a moving hand> or <look at a teddy bear>. Various aspects of human gaze behavior and eye movement have been modeled using active inference (Da Costa et al., 2020). Thus, let us use the active inference formalism as our planning objective.

In the absence of any uncertainty, the agent will strive for desirable states. For example, if it finds that teddy bears are visually pleasing, it will simply look at the teddy bear (Fig. 3.4a). Now consider the scenario in which a ball in the periphery of the agent unexpectedly starts to move, as illustrated in Fig. 3.4b. This creates a high uncertainty about future predictions. To minimize the expected uncertainty, the highest level $L = 2$ might select the $c^1 = \langle \text{look at the ball} \rangle$ as the next action. This guides the gaze behavior of the lower level, resulting in the agent looking at the ball. After accumulating sufficient evidence, we can assume that the agent has inferred that the ball is rolling and the agent is experiencing the overall event c^1 of <track a rolling ball>. Level 2 attempts to minimize the expected uncertainty about the next transitions at t^2 . Because the time scale t^2 depends on the context changes at level 1, π^2 aims at minimizing uncertainty about when, where, and how the event c^1 concludes and the next event begins. The ball

is likely to stop rolling and start to fall once it reaches the edge of the table. Therefore, to minimize uncertainty about the next event boundary, π^2 might select the <look at the edge of the table>-event as its next action. As a result, the low-level policy π^1 could direct the gaze toward the edge of the table. The resulting gaze behavior, i.e. looking at the goal of a movement in anticipation, describes exactly the goal-anticipatory gaze shift.

Again, we can ask ourselves if such a behavior could also emerge in a single low-level policy. Using expected uncertainty as a reward, a low-level policy would attempt to minimize uncertainty about every observation along a predicted trajectory. To minimize uncertainty about future ball positions, tracking the ball is the most suitable gaze behavior. Through tracking gaze, the agent gets the clearest information about the velocity and rolling direction of the ball. Thus, to even consider that the ball might change its trajectory when reaching the end of the table, a long planning horizon is needed. On the other hand, to model proactive gaze behavior in THICK MDPs, a horizon of $K^l = 1$ for every level l is sufficient.^{3.5}

3.4 Conclusion

This chapter introduced THICK MDPs, a theoretical framework that integrates insights about event cognition and hierarchical sequential decision making. Based on two examples, I sketched out how THICK MDPs not only facilitate long-horizon problem solving, but also enable modeling of behavioral tendencies of biological agents, such as proactive gaze behavior. Of course, there are still many open puzzle pieces. Here, I briefly outline relevant questions about the overall approach and describe how these questions are addressed in this thesis.

1. Can THICK MDPs be used to model aspects of cognition? THICK MDPs were motivated and partially derived from different theories of cognitive science. However, useful models of cognition explain behavior beyond narrative descriptions and generate testable predictions. In Chap. 4, we^{3.6} demonstrate that THICK MDPs fulfill these requirements using the example of goal-anticipatory gaze behavior. Based on THICK MDPs and active inference we derive a simple Bayesian model of event cognition that develops hierarchical predictions of event dynamics and event boundaries. When the model plans hierarchically to minimize uncertainty, it generates behavior that matches various findings of eye-tracking studies in infants. Crucially, the behavior develops with training experience, similarly to how goal-anticipatory gaze develops when infants mature.

^{3.5}Chapters 4 and 6 show this empirically.

^{3.6}For chapters Chap. 4-7 first person plural (we) is used instead of first person singular (I), to acknowledge the contributions of my collaborators on the presented hypotheses, methods, and results.

2. How can sparsely changing event codes be learned from scratch? The heavylifting that enables the development of the hierarchy in THICK MDPs come from the event contexts \mathbf{c}_t^l . Their stable temporal persistence gives rise to the overall hierarchy with nested time scales. In realistic scenarios, events are not simply provided to the system but need to develop from scratch based on interactions with the environment. Chap. 5 outlines how sparsely changing latent states can be learned within a recurrent neural network (RNN). Thereby, we not only show that the resulting latent states tend to encode simple events, but also demonstrate that the resulting RNN can be employed to improve state-of-the-art planning and reinforcement learning agents.

3. Can temporal abstractions develop from sparsely changing event codes? In THICK MDPs one level of the hierarchy operates at the time scale of event context changes of the next lower level. We hypothesized that such hierarchical organization could give rise to temporal abstractions. In Chap. 6, we investigate this claim by training a high-level neural network to only predict situations in which a latent state of a low-level sensorimotor network is updated. We illustrate that on the basis of this training scheme, meaningful temporal abstractions develop on the higher level.

4. Do THICK MDPs enable hierarchical planning? In Chap. 7, we combine all the insights previously gathered to design a two-level hierarchical world model, following the principles of THICK MDPs. We show in rich environments with image-based observation that the resulting THICK world model learns categorical, temporal abstractions on a high level, while at the same time making precise predictions on a low level. Furthermore, we demonstrate that the developing hierarchical predictive model can enhance the abilities of model-based reinforcement learning and planning methods.

5. Which aspects are still missing? I set out to create a cognition-inspired framework for event-based hierarchical decision making. However, many aspects of event cognition and human action planning have been simplified. In Chap. 8, the shortcomings of this approach and the developed methods are discussed. Additionally, I provide an outlook on how future research could overcome these limitations and build on the methods developed in this thesis.

4

Developing Event-Based Goal Anticipations^{4.1}

My conceptual framework, THICK MDP, is inspired by theories on how humans encode their experience and employ learned representations for prediction and planning. If the framework indeed captures aspects of humans' goal-directed planning, it should be useful to explain, or model, findings on how anticipation and goal-directed behavior arise in humans. This chapter presents CAPRI, a simple implementation of a THICK MDP, which is capable of modeling a variety of experimental findings on the development of goal-anticipatory gaze behavior in infants. Importantly, gaze behavior in CAPRI emerges purely from inferring gaze according to active inference using the hierarchical nested structure of THICK MDPs; no further assumptions are needed for modeling the experimental data.

^{4.1}This chapter is based on the publication:

Gumbsch, C., Adam, M., Elsner, B., & Butz, M. V. (2021). Emergent Goal-Anticipatory Gaze in Infants via Event-Predictive Learning and Inference. *Cognitive Science*, 45:e13016.

The text has been heavily revised to better suit the context of this thesis. The figures have been expanded and modified cosmetically. In accordance with the publication and to recognize the contributions of my coauthors, this chapter is written in first person plural (we).

4.1 Introduction

Already during the first year of life, infants appear to develop a rudimentary understanding that human actions are directed towards goals. An associated paradigm investigates the development of **goal-anticipatory gaze shifts**. In eye tracking studies, infants watch videos showing action events, e.g. a hand reaching for an object. If an infant looks at the goal of the shown event, e.g. a to-be grasped object, before the movement, e.g. a reach, is completed, the infant successfully anticipated the goal of the event (Elsner & Adam, 2021). The development of this ability appears to be supported by various factors, such as familiarity with the event and the agent involved (Cannon & Woodward, 2012; Kanakogi & Itakura, 2011), the motor ability to perform the movement themselves (Kanakogi & Itakura, 2011), behavioral cues indicating agency (Adam et al., 2017), and the saliency of the produced effect (Adam & Elsner, 2018, 2020; Adam et al., 2021). Despite the rather large conglomerate of findings, the internal computational mechanisms have been characterized only descriptively so far (see Gredebäck & Falck-Ytter, 2015 for a review).

We propose that goal-anticipatory gaze shifts emerge in infants from two interplaying factors: (1.) internally developing hierarchical predictions based on models of events, and (2.) the objective to minimize uncertainty in all currently activated models.^{4.2}

In support of this proposal, we modeled the emergence of goal anticipations in infants in a simplified object-interaction simulation. The modeling system first learned about simple events, such as reaching for an object or lifting an object. For each event, it learned **event schemata**, i.e. predictive models, which encode the typical dynamics and event boundary conditions in a probabilistic manner. Importantly, these event schemata allow for hierarchical predictions on two time scales: They can predict ongoing dynamics, e.g. a reaching hand moves towards an object, and they can predict upcoming event boundaries, e.g. hand-object contact at the end of reaching.

Throughout training, we put our system in experimental conditions, similar to how goal-anticipatory gaze shifts are tested in infants. The system was shown familiar action events (reaching) performed by agents that typically perform this kind of action (hand) or by unfamiliar agents (mechanical claw). We demonstrate that when the system chose its gaze to minimize the predicted uncertainty in its hierarchical predictions, it showed behavioral tendencies similar to the goal-anticipatory gaze behavior found in infants.

The remainder of this chapter is structured as follows. We first give an overview on goal-anticipatory gaze shifts and motivate our modeling approach. Next, we provide the algorithmic details of our model. In Sec. 4.4 we evaluate the model. Finally, we outline related computational models and discuss our results and their implications.

^{4.2}In other words, goal-anticipatory gaze should emerge from the gaze selection according to active inference (Sec. 2.2.4) in a THICK MDP (Chap. 3) as motivated and exemplified in Sec. 3.3

4.2 Development of Goal Anticipations

4.2.1 Goal-Anticipatory Gaze Behavior in Infants

Looking behavior is one of the first behaviors in human development and as such has been widely used to investigate how infants form expectations about observed goal-directed actions (e.g. Fantz, 1958; Gredebäck et al., 2010). The research paradigm to study goal anticipation *online*, i.e. while observing an action, is based on a seminal eye tracking study in which adult participants tended to shift their gaze to a to-be-attained goal before it was achieved in a block-stacking task (Flanagan & Johansson, 2003). Falck-Ytter et al. (2006) applied this paradigm to infants in a study in which 6- and 12-month-old infants, as well as adults, observed how toys moved into a container. The toys moved by being grasped and transported by a human or floating into the container. The data revealed anticipatory gaze behavior in the human agent condition for 12 months and adults, but not in the self-propelled condition and not for the 6-month-olds.

Several studies replicated and extended these findings. At around 7 months of age, infants start to show goal-anticipatory gaze shifts when observing human grasping actions, while they reactively track unfamiliar actions or unfamiliar agents, e.g. mechanical claws (Adam et al., 2016; Cannon & Woodward, 2012; Gredebäck & Melinder, 2010; Kanakogi & Itakura, 2011; Krogh-Jespersen & Woodward, 2014; Adam & Elsner, 2020). Furthermore, infant motor skills and the amount of experience with certain actions are positively correlated with the anticipation of goals during the observation of the respective actions (Cannon et al., 2012; Gredebäck & Melinder, 2010; Kanakogi & Itakura, 2011). In addition to familiarity, bottom-up cues can also boost goal anticipations, e.g. salient action effects or salient goals enable goal anticipations for unfamiliar agents at a younger age (Biro, 2013; Adam et al., 2017; Adam & Elsner, 2018, 2020).

4.2.2 Developing Anticipations via Event-Predictive Models

As detailed in Sec. 2.1, different theories suggest that humans tend to organize their experience through hierarchically structured predictive models of events (Butz, 2016, 2017; Butz et al., 2021; Zacks et al., 2007; Zacks & Tversky, 2001). A crucial characteristic of an event is that it is perceived to have a beginning and an end, i.e. an *event boundary* (Zacks & Tversky, 2001). Event boundaries tend to coincide with goals or subgoals of behavioral routines (Levine et al., 2017), e.g. the grasp of an object.

Our modeling approach, i.e. THICK MDP (see Chap. 3), follows the idea that event models are hierarchically organized in a partonomy (Zacks & Tversky, 2001) in which each level of the hierarchy transitions at the time scale of event boundaries of the next lower level. According to our framework, by learning to encode events, humans simultane-

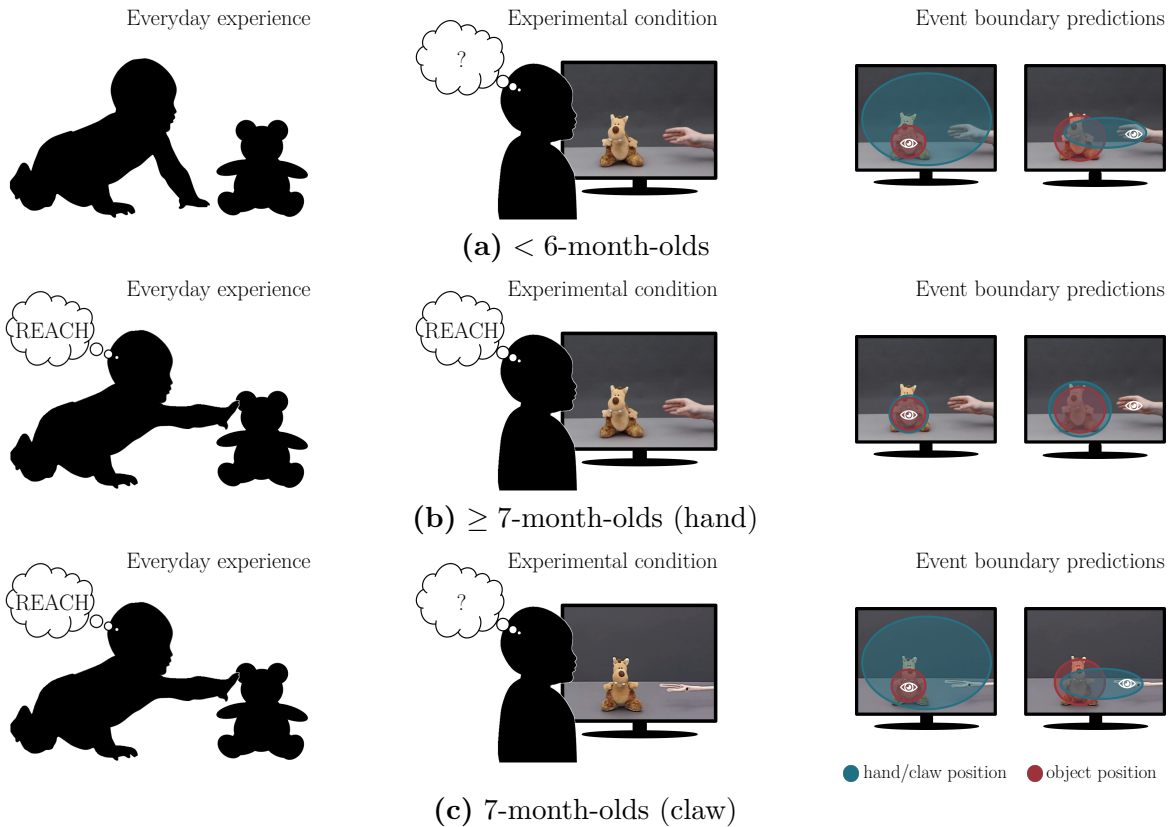


Figure 4.1: Modeling hypotheses for the development of goal anticipation for (a) infants younger than 6 months, (b) for older infants watching reaching motions done by hands, and (c) for 7-month-olds watching reaching motions by mechanical claws. White eye symbols visualize gaze. Colored circles visualize x -, y - position predictions for the next event boundary and their confidence (blue for the reaching hand or claw, red for the position of the object). The screenshots are taken from Adam & Elsner (2020).

ously learn to make hierarchical predictions about the exact low-level event dynamics and temporal abstract predictions about upcoming next event boundaries. Thus, encoding a certain event, e.g. a reach, allows the direct prediction of its goal by predicting its event boundary.

Hierarchical model-based predictions in the brain have been studied within the framework of the Free Energy principle and active inference (Kiebel et al., 2008; Pezzulo et al., 2018, 2015; Friston et al., 2018). Thus, our event-predictive models can be seamlessly merged with this line of work (Butz, 2016). According to active inference (Friston et al.,

2015, 2016; Parr et al., 2022), actions are partially inferred to minimize uncertainty about current predictions of the future (see Sec. 2.2.4). In THICK MDP predictions can occur simultaneously at hierarchical levels with nested time scales. Thus, when our approach is combined with active inference, actions are inferred to minimize uncertainty across multiple event-based time scales.

We hypothesize that merging active inference with event-predictive learning can explain a variety of experimental findings on the development of goal-anticipatory gaze behavior in infants, illustrated in Fig. 4.1. Let us assume a setting in which infants of different ages observe simple reach-and-grasp events performed by different agents (e.g. as in Kanakogi & Itakura, 2011; Adam & Elsner, 2020). We assume that infants younger than 6 months do not have a sufficiently learned event encoding for <reaching>. As a result, when they watch videos of reaching-events, they keep their gaze on the moving agent to better predict its trajectory. From about 7 months, when infants begin to show a goal-anticipatory gaze (Kanakogi & Itakura, 2011; Adam & Elsner, 2020), we hypothesize that they have developed a sufficiently well-predicting encoding for the observed event. Visual cues, e.g. appearance of the hand or movement-based information, activate the event encoding for <reaching>. This allows them to predict not only the unfolding dynamics, but also the next event boundary, which occurs for a reach when the hand touches the target object. Therefore, older infants will look at the target object to reduce the uncertainty about when, where, and how exactly the <reaching>-event will end. However, when 7-month-olds observe a reaching movement by an unknown agent, e.g. a mechanical claw, infants tend to track the claw with their gaze (Adam & Elsner, 2020).^{4.3} Our model assumes that, even though these children have learned a model for <reaching>, this event encoding will not be activated because some of the associated start conditions are not met.

We investigate the validity of our considerations by implementing and testing the proposed computational model, which we name Cognitive Action PRediction in Infants (CAPRI). CAPRI essentially implements a two-level THICK MDP (detailed in Suppl. B.1) which learns schematic models for different events such as <reaching for an object>. CAPRI continuously aims to decrease uncertainty through learning, inference, and action. As a result, it develops anticipatory epistemic gaze behavior similar to the goal-anticipatory gaze shifts generated by infants.

4.3 Cognitive Action Prediction in Infants (CAPRI)

CAPRI implements a model f_ϕ with learnable parameters ϕ and interacts with problems that can be modeled as a POMDP (detailed in Sec. 2.2.3). We assume that the system is

^{4.3}Assuming that the unfamiliar agent does not exhibit salient agency-related cues or action effects.

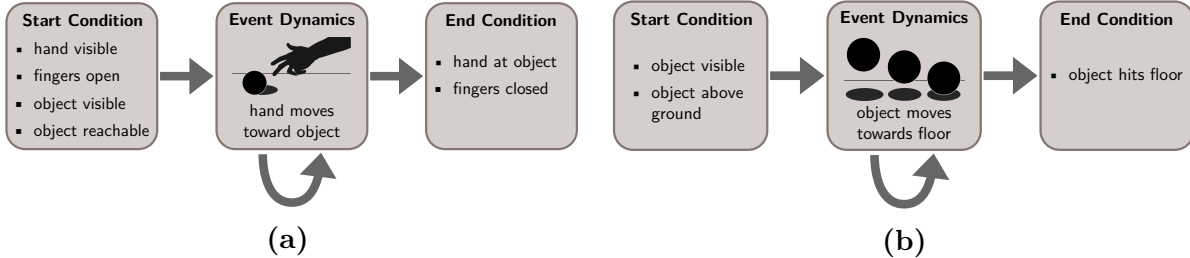


Figure 4.2: Event schemata in CAPRI are composed of three components: A start condition, an event dynamics model, and an end condition. We exemplify potential encodings of two events: `<reaching>` (a) and `<falling>` (b).

equipped with a set of policies π^i . In every time step t , the system receives an observation \mathbf{o}_t and samples an action according to the active policy π_t . In our scenario, the agent is an observer, and thus policies correspond to gaze behavior.

4.3.1 Learning of Event Schemata

CAPRI encodes its interactions with the world in terms of events. For this, CAPRI learns **event schemata**. Each event schema is a separate model composed of three components: a **start condition**, the **event dynamics model**, and an **end condition** (see Fig. 4.2). All components are encoded as probabilistic models, that is, conditional likelihood distributions over observations.

The **start condition** $P_{e^i}^{\text{start}}$ encodes the particular preconditions necessary for an event e^i to begin via the likelihood $P_{e^i}^{\text{start}}(\mathbf{o}_t | \pi_{t-1})$, with observation \mathbf{o}_t and the last executed policy π_{t-1} at time t . For example, a `<reaching>`-event might typically start with the observation \mathbf{o}_t of a hand that started to move towards a reachable object (cf. Fig. 4.2a).

The **event dynamics model** $P_{e^i}^{\text{event}}$ encodes how an event e^i typically unfolds. More specifically, we model the likelihood of observation \mathbf{o}_t given the last observation \mathbf{o}_{t-1} and the policy π_{t-1} : $P_{e^i}^{\text{event}}(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1})$. For example, at a certain time t during a `<reaching>`-event the position of the hand, described by observation \mathbf{o}_t , is expected to be closer to the object than one time step before, described by observation \mathbf{o}_{t-1} (Fig. 4.2a).

The **end condition** $P_{e^i}^{\text{end}}$ encodes the typical end state of an event e^i . We model the end condition through the likelihood $P_{e^i}^{\text{end}}(\mathbf{o}_t | \mathbf{o}_{t-\iota}, \pi_{t-1})$. We assume that the end of an event is predictable from an arbitrary point during the event. We incorporate this idea through a retrospective time horizon ι with $1 \leq \iota \leq T$ that spans over the duration T of the event e^i . Thus, the end condition $P_{e^i}^{\text{end}}$ models the likelihood of observation \mathbf{o}_t at the end of event e^i given the last policy π_{t-1} and some previous observation $\mathbf{o}_{t-\iota}$, which lies ι

time steps in the past within the same event. For example, when `<reaching>` (Fig. 4.2a) ends at time t , the final position of the hand, captured by observation \mathbf{o}_t , is close to the reachable object, which can be predicted from some previous observation \mathbf{o}_{t-l} .

Thus, for every event e^i CAPRI learns three separate likelihood distributions encoding the *dynamics* of an event and its *boundaries*. Based on these encodings, CAPRI is able to make predictions about the future on two levels of abstraction. Let us assume that the system has an event model e^{fall} , encoding `<object is falling>` (see Fig. 4.2b). When CAPRI perceives that a ball that just rolled over the edge of a table, the start condition $P_{e^{\text{fall}}}^{\text{start}}$ would predict a high likelihood for this particular observation, activating the event schema e^{fall} . CAPRI can either predict the immediate next observation \mathbf{o}_{t+1} via $P_{e^{\text{fall}}}^{\text{event}}$, which predicts the ball mid-air depending on its past position and velocity. CAPRI can also make a temporal abstract prediction about the end of this event through $P_{e^{\text{fall}}}^{\text{end}}$, predicting some future observation \mathbf{o}_τ of the ball when falling ends, e.g. upon hitting the floor.

All distributions are modeled as multivariate Gaussians. Each distribution is parameterized by a separate neural network (details in Suppl. B.2.2). During training, all event schemata are learned in a supervised setting (details in Suppl. B.2.3). Thus, during training CAPRI is informed about the underlying events. However, in the absence of explicit labels about the ongoing events, the likelihood estimates can be used for Bayesian inference to infer the probability of an event.

4.3.2 Event Inference

During *training* CAPRI receives supervised information about which event is currently unfolding. In contrast, during *testing* at time t CAPRI infers probabilities about which event e^i currently unfolds from past sensorimotor information, i.e. the sequence of all observations $\mathbf{o}_{1:t} = (\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_t)$ and past policies $\pi_{1:t-1} = (\pi_1, \pi_2, \dots, \pi_{t-1})$ of the current episode. CAPRI infers the probability $P(e_t^i | \mathbf{o}_{1:t}, \pi_{1:t-1})$ for every possible event e^i being active. This event inference is performed iteratively every time step t after executing an action based on π_{t-1} and receiving a new observation \mathbf{o}_t via

$$P(e_t^i | \mathbf{o}_{1:t}, \pi_{1:t-1}) = \sum_{e^j} P(e_t^i | \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j) \cdot P(e_{t-1}^j | \mathbf{o}_{1:t-1}, \pi_{1:t-2}). \quad (4.1)$$

At every time step t , CAPRI first computes the probability of an event transition $P(e_t^i | \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)$ for every combination of e^i and e^j and then updates its current event estimate. We compute $P(e_t^i | \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)$ as

$$P(e_t^i | \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j) = \frac{P(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(e_t^i | e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \cdot P(e_t^h | e_{t-1}^j)}. \quad (4.2)$$

We provide the full derivation in Suppl. B.2.4. We set the event transition prior $P(e_t^i | e_{t-1}^j) = 0.9$ for staying in the same event, i.e. $e^i = e^j$ and $P(e_t^i | e_{t-1}^j) = \frac{0.1}{N}$ for $e^i \neq e^j$ where N is the number of available events. This corresponds to the assumption that events tend to persist over time with a prior probability of 90%.

To compute the Bayesian posterior $P(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j)$, we use the event schemata distributions, which were outlined in Sec. 4.3.1. There are **two cases** for computing this likelihood depending on e_t^i and e_{t-1}^j : Either the world remains in the same event, or an event boundary occurred from time $t - 1$ to time t . When remaining in the **same event**, i.e. $e^i = e^j$, we compute the likelihood of \mathbf{o}_t by

$$P(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) = P_{e^i}^{\text{event}}(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}), \quad (4.3)$$

with $P_{e^i}^{\text{event}}$ encoding the event-respective dynamics distribution (see Sec. 4.3.1). For the **event boundary** case, where $e^i \neq e^j$, the likelihood is computed by

$$P(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) = P_{e^i}^{\text{start}}(\mathbf{o}_t | \pi_{t-1}) \cdot P_{e^j}^{\text{end}}(\mathbf{o}_t | \mathbf{o}_{t-1}, \pi_{t-1}), \quad (4.4)$$

using the start and end conditions $P_{e^i}^{\text{start}}$ and $P_{e^j}^{\text{end}}$ (see Sec. 4.3.1).

The event inference process is initialized at time $t = 1$ with

$$P(e_1^i | \mathbf{o}_1, \pi_1) = \frac{P_{e^i}^{\text{start}}(\mathbf{o}_1 | \pi_1)}{\sum_{e^h} P_{e^h}^{\text{start}}(\mathbf{o}_1 | \pi_1)}, \quad (4.5)$$

using only the start conditions $P_{e^i}^{\text{start}}$ (see Sec. 4.3.1).

4.3.3 Active Inference

Besides passively inferring events, CAPRI can actively gather sensory observations by selecting (gaze) policies. For this, the system infers its policy based on computing Expected Free Energy (EFE) (see Eq. 2.8). EFE is composed of two terms: predicted divergence from desired states and predicted uncertainty. Since we are interested in gaze and assume that the attractiveness of the visual stimuli is well controlled, we only focus on predicted uncertainty (second term in Eq. 2.8)^{4.4}. At t we can compute EFE for a candidate policy π^i and sensorimotor history $\mathbf{o}_{1:t}$ and $\pi_{1:t-1}$ with

$$\text{EFE}(\pi^i, K, \mathbf{o}_{1:t}, \pi_{1:t-1}) = \frac{1}{K} \sum_{k=t}^{t+K} \mathbb{E}_{\mathbf{o}_{t:k}, e_{t:k} \sim f_\phi} (\mathcal{H}[f_\phi(\mathbf{o}_{k+1} | e_{1:k}, \mathbf{o}_{1:k}, \pi_{1:t-1}, \pi_{t:k}^i)]), \quad (4.6)$$

^{4.4}To align with our notation we make minor changes to Eq. 2.8. Since CAPRI only interacts with the world in terms of policies π , we omit actions \mathbf{a}_k replacing them with policies π_k . Furthermore, CAPRI infers the state of the world in terms of events e^i . Thus, we replace states \mathbf{s}_k with events e_k .

with CAPRI as f_ϕ , entropy \mathcal{H} and a time horizon K . Thus, EFE is calculated by predicting the execution of policy π^i for a time horizon K expanding from the present into the future.

As described in Sec. 4.2.2, CAPRI can predict the future on two levels of abstraction. It can make (1.) *fine-grained* predictions about the next observation within the current event or (2.) *temporal abstract* predictions about upcoming event boundaries. We replace the prediction horizon K with the two types of prediction:

$$\begin{aligned} \text{EFE}'(\pi^i, \mathbf{o}_{1:t}, \pi_{1:t-1}) = & \underbrace{\sum_{e^j} \mathcal{H}[P_{e^j}^{\text{event}}(\mathbf{o}_{t+1} | \mathbf{o}_t, \pi^i)] P(e_t^j | \mathbf{o}_{1:t}, \pi_{1:t-1})}_{\text{uncertainty of event dynamics}} \\ & + \underbrace{\sum_{e^j} \sum_{e^h} \mathcal{H}[P_{e^h}^{\text{start}}(\mathbf{o}_\tau | \pi^i) P_{e^j}^{\text{end}}(\mathbf{o}_\tau | \mathbf{o}_t, \pi^i)] \cdot P(e_t^j | \mathbf{o}_{1:t}, \pi_{1:t-1})}_{\text{uncertainty of event boundaries}}, \end{aligned} \tag{4.7}$$

with τ marking the time of the next event boundary. In the modified EFE', CAPRI attempts to minimize predicted entropy over the currently estimated event dynamics and over all possible next event boundaries encoded by the end- and start conditions.

Assuming a set of policies, the next policy can then be selected via

$$\pi_t = \arg \min_{\pi^i} \text{EFE}'(\pi^i, \mathbf{o}_{1:t}, \pi_{1:t-1}). \tag{4.8}$$

We hypothesize that active inference based on the two terms of Eq. 4.7 leads to goal-anticipatory gaze shifts.^{4.5} Inferring gaze policies to **minimize predicted uncertainty about current dynamics encourages tracking gaze**, because tracking the velocity of a moving entity helps to predict its immediate trajectory. **Minimizing predicted uncertainty about future event boundaries encourages proactive gaze towards goals**, because knowledge about the exact location of the goal reduces uncertainty about when and where the next event boundary will occur. The combination could enable goal-predictive gaze shifts, i.e. disengaging the gaze from a moving entity to its goal location before it is reached.

4.4 Experiments

We evaluate CAPRI in a simple three-dimensional agent-patient interaction simulation, mimicking the probable action experiences of infants. In the following, we first detail the simulation and training procedure. We then evaluate the behavior of the system and show that goal-anticipatory gaze behavior indeed emerges over the course of learning.

^{4.5}We provide a more detailed example of how this might develop in the case of a rolling ball in Sec. 3.3.

4.4.1 Simulation Setup

Our simulation always contains **two entities** – an **agent** \mathbf{a} , that is the *active* part of an interaction, and a **patient** \mathbf{p} , that is the *passive* part.^{4.6} At every time step t , CAPRI received an observation $\mathbf{o}_t \in [-1, 1]^{18}$, containing absolute and relative positions of the entities, their velocities, and their distance. Furthermore, for all entities a one-dimensional appearance of the entity was observed. The appearance serves as a simple cue to distinguish entities, e.g. distinguishing hands from claws, but does not affect dynamics.

Our system acted by choosing one of **three different policies** that evoke gaze behavior. We model gaze as a simplified entity-based focus, influencing the standard deviation σ of normally distributed sensory noise. When **focusing on the agent** (π^a), large sensory noise ($\sigma = 0.1$) is added for all patient-related components of \mathbf{o}_t , but small sensory noise ($\sigma = 0.01$) was added to agent-based information. When **focusing on the patient** (π^p), the sensory noise scheme was reversed. When **focusing on neither** of the entities (π^n), normally distributed noise with $\sigma = 0.1$ was added to the full observation \mathbf{o}_t . Thus, in our simplified entity-based gaze, the actual physical distances between entities are ignored.

Four possible events e^i were simulated, mimicking events that may be encountered and produced by infants:

- During a **<standing still>**-event e^{still} the agent and the patient remained motionless for a fixed number of time steps, mimicking stationary objects.
- During a **<randomly directed motion>**-event e^{rand} the agent moved constantly in a fixed but randomly generated direction with decreasing velocity. This event mimics the observation of rolling or sliding entities, such as a toy car.
- In a **<reaching>**-event e^{reach} a hand agent moved towards the patient with a randomly set, constant velocity. The event ended when the hand reached the patient.
- A reach is always followed by a **<transporting>**-event e^{tran} , where hand and patient moved together to a random goal location with a randomly set constant velocity.

By combining these events, three possible event sequences E were generated for training (cf. Fig. 4.3): In E^{grasp} a hand agent reached for the patient (e^{reach}), transported it (e^{tran}), let go, and randomly moved away (e^{rand}). E^{rand} started with a randomly directed motion (e^{rand}) followed by both entities standing still (e^{still}). E^{still} showed both entities motionless (e^{still}). For testing, we considered an additional sequence E^{test} identical to E^{grasp} , but also allowed claw agents. Event sequences took roughly 100-300 time steps.

The experiments were divided into training and testing phases. Each **training phase** consisted of 100 event sequences, in which the system was informed of the observed events and gaze policies were randomly chosen per sequence. Each training phase was followed

^{4.6}The agent could also be called the *subject* and the patient the *object* of an interaction.





sequence	events	visualization	agents
E^{still}	e^{still}		all types
E^{rand}	$e^{\text{rand}} \rightarrow e^{\text{still}}$		all types
E^{grasp}	$e^{\text{reach}} \rightarrow e^{\text{tran}} \rightarrow e^{\text{rand}}$		only hands
E^{test}	$e^{\text{reach}} \rightarrow e^{\text{tran}} \rightarrow e^{\text{rand}}$		hands or claws

Figure 4.3: Simulation environment of CAPRI showing event sequences, their individual events, corresponding exemplar visualizations rendered from a bird’s-eye view (z -dimension shown via entity size), and potential agents.

by a **testing phase**, mimicking eye-tracking studies in infants (Kanakogi & Itakura, 2011; Adam et al., 2016, 2017, 2021; Adam & Elsner, 2020). During testing, we recorded the gaze of CAPRI while the system was shown grasping sequences (E^{test}). In contrast to training, CAPRI had to infer which event was currently observed (cf. Sec. 4.3.2), select its gaze policy using active inference (cf. Sec. 4.3.3), and no model updates were performed.

Similarly to Adam & Elsner (2020), we distinguished between **two testing conditions**: In the **hand**-condition, the system was shown a grasping sequence E_{test} performed by a hand agent, as during training. In the **claw**-condition, the same event sequence E^{test} was shown with a claw agent. Claw agents only differed in the appearance component of the observation \mathbf{o}_t . Each test phase consisted of 10 E_{test} event sequences for each condition.

We conducted 20 randomly initialized experiments by training CAPRI for 30 training phases. We provide further simulation details in Suppl. B.3.

4.4.2 System Behavior

We first analyze CAPRI’s internally estimated **event probabilities** to quantify the system’s beliefs about the unfolding events. Figure 4.4 (bottom) shows the inferred event probabilities for two exemplary event sequences E^{test} for a fully trained system and either

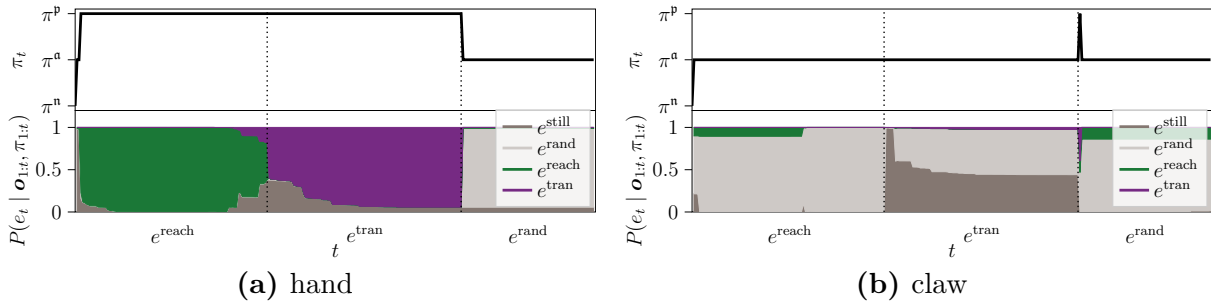


Figure 4.4: Exemplary event and policy inference of CAPRI after full training over the course of one event sequence E^{test} for a hand agent **(a)** and a claw agent **(b)**. The top rows show the active policy π_t . The bottom rows show the inferred event probability estimates. Event boundaries are marked by dotted lines.

(a) hand or (b) claw agents.^{4.7} For a hand agent, CAPRI tends to assign high probabilities to the correct event quickly after an event started. For example, after roughly 20 time steps for e^{reach} and only two steps for e^{rand} CAPRI assigned almost 100% probability to the correct event. Inferred event probabilities differ drastically for claws (Fig. 4.4b). For claw agents, CAPRI infers a high probability of e^{rand} during a reach and assigns high probabilities to e^{rand} and e^{still} during a transportation event.

These **differences in event inference affect gaze behavior**, exemplified in Fig. 4.4 (top). For the hand agent (Fig. 4.4a), CAPRI started looking at the patient, i.e. activated π^p , once it was certain of observing a reach ($\gtrsim 90\%$ probability). Thus, for hands, we observe a goal-anticipatory gaze: the system looked at the reaching target before it was reached. For the claw agent (Fig. 4.4b), the system tracked the agent, i.e. chose π^a , throughout the reach. Thus, for the claw, CAPRI did not show goal-anticipatory gaze.

How are event inference and gaze behavior affected by experience? Figure 4.5 shows the mean inferred event probability during test phases over training experience for hand (left) and claw agents (right)^{4.8}. During the course of training, CPARI learned to correctly infer progressively higher probabilities for the actual underlying events when the hand executed them. In contrast, when CAPRI observed a claw it incorrectly inferred e^{still} and e^{rand} with a much higher probability, even when actually observing e^{reach} or e^{tran} .

As a result, **gaze behavior develops differently for hands or claw agents**. We quantify gaze behavior by analyzing at what time CAPRI on average activated the policy

^{4.7}We provide more exemplary sequences in Suppl. B.4.1.

^{4.8}For every event $e^i \in E^{\text{test}}$ starting at time t^s and ending at time t^e , we compute the mean event probabilities as $\bar{P}(e^j | \mathbf{o}_{1:t}, \pi_{1:t}) = \sum_{t=t^s}^{t^e} P(e^j | \mathbf{o}_{1:t}, \pi_{1:t}) \frac{1}{t^e - t^s}$, for all events e^j .

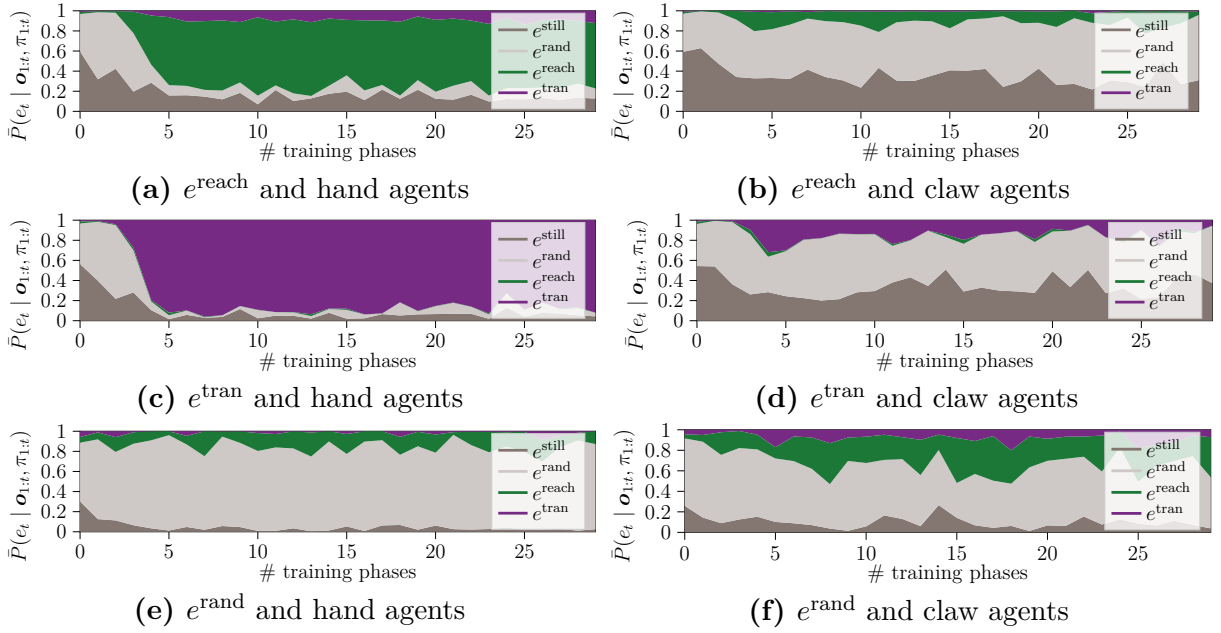


Figure 4.5: Mean event inference of CAPRI plotting $\bar{P}(e_t^i | \mathbf{o}_{1:t}, \pi_{1:t})$ for $e^i \in \{e^{\text{still}}, e^{\text{rand}}, e^{\text{reach}}, e^{\text{tran}}\}$ during E^{test} over the course of learning for hand subjects (a,c,e) and claw agents (b,d,f). (a)–(b) show the event probabilities during e^{reach} , (c)–(d) during e^{tran} , and (e)–(f) during e^{rand} .

π^p within an event sequence E^{test} .^{4.9} Figure 4.6a plots the average time of first gazes at the patient over training experience. Early during training, the system on average looked at the patient after it was grasped, i.e. marked by t^0 . After the fourth training phase, the system began to systematically activate π^p in the beginning of reaching events e^{reach} in the case of hand agents. This goal-anticipatory gaze did not develop for claw agents.

These results indicate that over the course of training, CAPRI develops goal-anticipatory behavior that is **comparable with the gaze behavior that develops in infants** (Adam & Elsner, 2020). Figure 4.6b shows the mean gaze arrival time for infants of different age groups watching videos of hand or claw agents reaching for and lifting a toy (Adam & Elsner, 2020). The videos started with a 1000 ms still frame depicting a toy, and then showed a hand or claw entering from the right part of the screen and moving linearly toward the toy (≈ 2500 ms), lifting the toy and placing it down again (≈ 2800 ms), followed by a still frame (≈ 2400 ms). To measure gaze arrival times, an area of interest (AOI) was created to cover the toy. Mean gaze arrival times were calculated by

^{4.9}Simulations where π_p was not activated counted as looking at the patient at the end of the sequence.

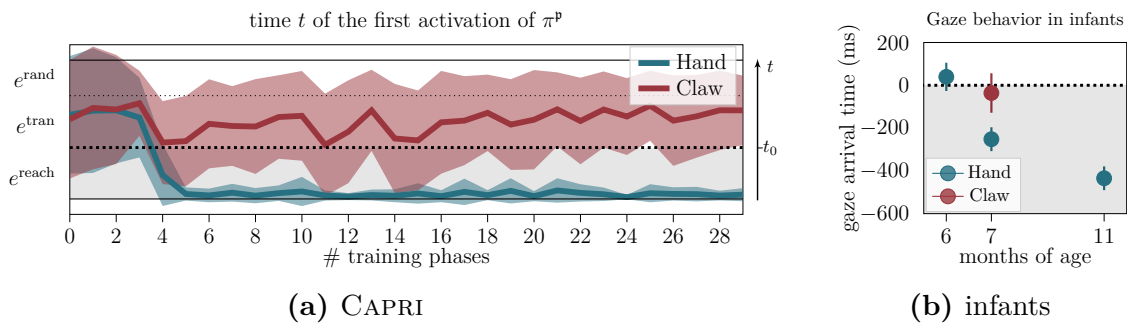


Figure 4.6: Anticipatory gaze behavior of CAPRI and in infants when watching reach-grasp-lift (E^{test}) sequences performed by hands (blue) or claws (red). We plot **(a)** the mean time t during E^{test} when CAPRI first activated π^p , i.e. it looked at the patient, and **(b)** the mean gaze arrival time at the reaching target for infants of different age from Adam & Elsner (2020). The dotted horizontal line mark event boundaries with agent-patient-contact at $t = t^0$ in **(a)** or at $t = 0$ in **(b)**. Thus, in both plots data points below this contact line mark an anticipatory gaze. Shaded areas show standard deviation across simulations **(a)** and error bars show standard error over participants **(b)**.

subtracting the time the agent entered the AOI from the time of the first AOI fixation. Thus, a negative gaze arrival time corresponds to goal-anticipatory gaze. As shown in Fig. 4.6b, from 7 months onward, when the infants observed reaching movies done by hands, they tended to look at the AOI before the hand arrived. For claw agents, 7-month-olds tended to first look at the AOI upon arrival of the agent. Similar behavior developed in CAPRI (cf. Fig. 4.6a and Fig. 4.6b).

4.4.3 Discussion of results

When did CAPRI show goal-anticipatory gaze shifts? CAPRI was neither trained nor preprogrammed to generate goal-anticipatory gaze behavior. Instead, the behavior emerged purely from the system estimating the ongoing event via Eq. 4.1 and choosing its actions by means of active inference, aiming to decrease the expected uncertainty about the ongoing event and the upcoming event boundaries according to Eq. 4.7. Apparently, for reaching events e^{reach} CAPRI could minimize uncertainty best by choosing policy π^p , likely because less noisy information about the patient’s position informed the system where reaching would end.^{4.10} On the other hand, when the current event is uncertain or a random motion event e^{rand} is unfolding, CAPRI could minimize the uncertainty best by

^{4.10}Additional evaluations in Suppl. B.4.2 confirm that when uncertainty about event boundaries was not considered for gaze selection, the system did not show goal-anticipations for hands.

looking at the agent, i.e. choosing gaze policy π^a , in order to gain clear information about the agent’s appearance, position, and velocity.

How did goal-anticipatory gaze shifts develop selectively with training? During the beginning of training, CAPRI did not show goal-anticipatory gaze shifts during E^{test} because the system (1.) did not recognize the underlying events, inferring a low probability for e^{reach} , and (2.) had not yet learned through which policy expected uncertainty could be decreased. Once its models were sufficiently precise, CAPRI learned that fixating the target object served to decrease uncertainty best for e^{reach} . As a result, goal-anticipatory gaze behavior emerged during the course of training when observing a reaching hand. For claw agents, the system produced incorrect inferences. This is mainly due to the system learning that e^{reach} and e^{tran} events are typically performed by agents with a certain hand-like appearance, encoded in their observation \mathbf{o}_t . As a result, for claws e^{rand} or e^{still} were mostly inferred during a reach, resulting in tracking gaze, i.e. π^a .

Could this explain experimental findings for infants? We propose that similar processes are involved when infants exhibit goal-anticipatory gaze behavior (e.g. Adam & Elsner, 2020). For most events involving a moving agent, tracking the agent with one’s gaze can give the most information about the future, e.g. the agent’s future position. However, for goal-directed events, e.g. a reach, information about the goal can also improve predictions of the future. If the event is familiar, its event boundary tells us where to look for the goal. When observing reaching actions, the anticipatory gaze of infants depends on their action experience and on the familiarity with the agent (Adam et al., 2016; Falck-Ytter et al., 2006; Kanakogi & Itakura, 2011). In CAPRI, goal-anticipatory gaze depends on reaching experience and the recognition of the event.^{4.11} We assume that infants also associate perceptual features, e.g. visuospatial features of a hand together with a graspable object, with the encoding of an event. Unfamiliar situations, e.g. unfamiliar agents, may not activate the correct event schema for generating goal-anticipatory gaze behavior.

4.5 Related Work

Event and context inference Similar to CAPRI, the Structured Event Model (SEM) (Franklin et al., 2020) predicts event dynamics using likelihood distributions generated by separate neural networks. In SEM, events are inferred via nonparametric Bayesian clustering. SEM was able to produce human-like event segmentation for complex video stimuli (Bezdek et al., 2022). However, unlike CAPRI, SEM does not explicitly model event boundaries, which prohibits hierarchical predictions. The Contextual Inference MOdel (COIN) (Heald et al., 2021) is a Bayesian model closely related to CAPRI. COIN

^{4.11}We further investigate how experience with E^{grasp} and difficulty to recognize an agent affects the goal-anticipatory gaze of CAPRI in additional experiments in Suppl. B.4.3– B.4.4.

infers contexts which encode temporal persistent dynamics and guide sensory predictions (Heald et al., 2021), akin to events. COIN was able to model a variety of findings on sensorimotor learning and memory formation (Heald et al., 2021, 2023). However, so far, all experiments have been heavily simplified using one-dimensional sensory observations.

Modeling the development of goal-anticipations Goal-anticipation in infants was previously modeled by Copete et al. (2016) using open-loop sensorimotor forward simulations. In their model, the goals of reaching motions were detected through salient changes in predicted tactile simulations. Although the model worked well in a complex robotic reaching scenario with high-dimensional inputs, the simplified goal detection confines the approach to scenarios in which the event boundaries involve tactile feedback.

4.6 Discussion

In this chapter, we have proposed how goal-anticipatory gaze shifts in infants may emerge. We introduced and implemented CAPRI, a computational model that learns event schemata, which predict the dynamics as well as the start and end conditions of an event. Thus, CAPRI implements a simple two-level THICK MDP (see Chap. 3) allowing both low-level dynamics predictions and high-level event boundary predictions. CAPRI constantly infers which event is unfolding based on incoming observations. Additionally, CAPRI chooses gaze behavior aiming solely at minimizing expected future uncertainty of its hierarchical predictions about the current dynamics and the next expected event boundary.

In a simple agent-patient interaction scenario, CAPRI showed gaze behavior that qualitatively **models three findings from eye-tracking studies in infants** (Adam et al., 2016; Cannon & Woodward, 2012; Kanakogi & Itakura, 2011; Adam & Elsner, 2020). (1.) Early during training, CAPRI did not look at a reaching target before reaching was complete. Similarly, infants younger than 6 months do not show goal-anticipatory gaze behavior when observing reaching motions (Adam & Elsner, 2020). (2.) Later during training, when a familiar hand reached for an object, CAPRI looked at the target before it was reached, similar to 7-month-olds looking at the target of a hand-reaching movement in anticipation (e.g. Adam et al., 2016; Adam & Elsner, 2020). (3.) When an unfamiliar claw was observed reaching, CAPRI preferred to follow the agent with its gaze. Similarly, when 7-month-olds (Adam & Elsner, 2020), and in some studies also 11- or 12-month-olds (Adam et al., 2016, 2021), observe a mechanical claw reaching for an object, they tend to perform tracking gaze. None of these effects were explicitly programmed into the system. Instead, they emerged from combining the structure of event encodings (cf. Sec. 2.1) with the minimization of predicted uncertainty through active inference (cf. Sec. 2.2.4).

Developmental research has raised the question of which representations may enable goal-anticipatory gaze behavior in infants. Infants have been proposed to learn flexible

goal representations for each event (Cannon & Woodward, 2012). An alternative explanation suggests that infants rely on *trajectory-based information* to estimate how an observed movement will end (Ganglmayer et al., 2019). There seems to be experimental evidence to support both explanations (Cannon & Woodward, 2012; Ganglmayer et al., 2019). Our modeling results suggest that *both types of representation* are involved. Event dynamics models encode changes over a short period of time, essentially predicting exact trajectories. However, inferring goals through open-loop forward simulation is costly, prone to error accumulation, and might not elicit goal-anticipatory gaze (additional experiment in Suppl. B.4.2). Encoding event boundaries enables the direct anticipation of goals.

Nonetheless, various aspects and design considerations demand future modeling work. First, CAPRI never develops goal-anticipatory gaze for claw agents. However, starting around 11 months of age infants perform a goal-anticipatory gaze shifts for mechanical claws when they show cues of agency, such as self-propelled motion or salient action-effects (Adam et al., 2017, 2021). Thus, instead of directly conditioning on the agent’s appearance, more general agency cues may be encoded in the start conditions of the event schemata. Identifying agency cues on the fly, i.e. within a test phase, could then help to recognize events, e.g. reaching, thus enabling goal-anticipatory gaze.

Second, CAPRI selects its gaze to minimize predicted uncertainty over a fixed temporal horizon into the future. This is sufficient for the investigated scenario. However, the effects of goal anticipation might emerge even more strongly when the prediction horizon is dynamically determined based on the available cognitive resources (Butz, 2022; Lieder & Griffiths, 2020). For example, CAPRI could learn to only anticipate the next event boundary when it is certain about the ongoing dynamics. Future research could explore adaptive uncertainty- or resource-dependent prediction horizons.

Third, while CAPRI learns from its own experiences, we ignored the fact that motor signals are available when executing particular events. In future modeling work (see Chap. 6), we will enhance our system to learn from its own motor experience.

Finally, our experiments were heavily simplified. In more realistic applications there might be thousands of potential events and tracking the probability of all events at all steps is not feasible. In the current implementation, we provided supervised information on observed events during training. Additionally, we specified for each event which entity takes the role of agent and patient. Furthermore, each event schema components were modeled as a single-layered neural network, essentially only allowing for linear predictions. Clearly, this is all unrealistic. Humans learn to attend to relevant events and predict the potentially non-linear event dynamics of the involved entities purely from the continuous stream of sensorimotor information. Such self-supervised segmentation and encoding of events would allow us to apply our modeling approach to much more complex scenarios with unlabeled events, high-dimensional observations, and more complex dynamics. Thus,

as a next step in this thesis, we will attempt to tackle this challenge, by learning to segment activity into events.

5

Sparsely Changing Latent States for Prediction and Planning^{5.1}

So far, I have emphasized the importance of events for prediction and planning. However, how an agent could learn to compress its continuous stream of sensorimotor experience into representations of events has not been addressed. In this chapter, we introduce GATELORD, a recurrent neural network (RNN) that incorporates the inductive bias to maintain sparsely changing latent states. We show in various prediction and planning tasks that GATELORD tends to encode the underlying generative factors of the environment, ignores spurious temporal dependencies, and generalizes better than other RNNs. Importantly, GATELORD automatically discretizes time series into events with piecewise constant latent states and sparse event boundaries upon which its latent states are updated.

^{5.1}This chapter is based on the publication:

Gumbsch, C., Butz, M. V. & Martius, G. (2021). Sparsely Changing Latent States for Prediction and Planning in Partially Observable Domains. *Advances in Neural Information Processing Systems* (NeurIPS 2021), 17518–17531.

The text has been slightly revised to better fit within the storyline of this thesis. The figures have been adopted except for minor cosmetic changes. In accordance with the publication and to recognize the contributions of my coauthors, this chapter is written in first person plural (we).

5.1 Introduction

When does the meeting start? Where are my car keys? Is the stove turned off? Since many aspects of the world are not directly observable, humans need to memorize a lot of information to plan their behavior. To allow artificial agents to plan in partially observable domains (see Sec. 2.2.3), we need to give them the ability to develop suitable memory structures for decision-making.

Recurrent neural networks (RNNs) are often used to deal with partial observability (Hausknecht & Stone, 2015; Igl et al., 2018; Zhu et al., 2017; Hafner et al., 2019a). They encode past observations by maintaining latent states, which are iteratively updated at every time step. However, continuously updating the latent state causes past information to quickly “wash out”. Long-Short Term Memory networks (LSTM, Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRU, Chung et al., 2014) deal with this problem by using internal gates. However, they cannot leave their latent states completely unchanged because small amounts of information continuously leak through the sigmoidal gating functions. Additionally, inputs typically need to pass through the latent state to affect the output, making it difficult to disentangle observable information from unobservable information within their latent states.

In contrast to RNNs, humans seem to update latent beliefs about the world somewhat sparsely in time. For example, as discussed in more detail in Sec. 2.1, it has been argued that the activations of internal event encodings seem to persist over time (Zacks et al., 2007; Butz, 2016; Radvansky & Zacks, 2014; Schapiro et al., 2013; Shin & DuBrow, 2021; Butz et al., 2021). Event Segmentation Theory (EST) (Zacks et al., 2007) proposes that at every point in time a set of event models is active and provides additional event-relevant information to predict future perceptions. EST argues that the active set of event models is only updated sparsely in situations in which transient prediction errors are detected. Thus, humans seem to discretize sensorimotor activity into long segments with a stable activity of event-encoding beliefs, and sparse points in time, i.e. event boundaries, in which these beliefs are updated.

Our main hypothesis is that the latent states of planning agents do not need to be updated in every time step. We hypothesize that many latent generative factors in the physical world are constant over extended periods of time. Thus, there might not be a need to update latent states at every time step. For example, consider dropping an object: If the drop-off point as well as some latent generative factors, such as gravity and aerodynamic object properties, are known, iteratively predicting the fall can be reasonably accomplished by a non-recurrent process. Similarly, when an agent picks up a key, it is sufficient to memorize that the key is inside its pocket. However, latent factors typically change systematically at particular points in time. For example, the aerodynamic prop-

erties of a falling object change drastically when it shatters on the floor, and the location of the key changes systematically when the agent takes it out of their pocket.

These observations are related to assumptions used in causality research. A common assumption is that the generative process of a system is composed of autonomous mechanisms that describe causal relationships between the variables of the system (Peters et al., 2017; Schölkopf, 2019; Schölkopf et al., 2021). When focusing on decision making, it has been proposed that these mechanisms tend to interact sparsely in time and locally in space (Pitis & Garg, 2020; Seitzer et al., 2021). Causal models aim at creating dependencies between variables only when there exists a causal relationship between them, in order to improve generalization (Schölkopf, 2019). Updating the latent state of a model in every time step, on the other hand, induces the prior assumption that the generative latent state typically depends on all previous inputs. Thus, by suitably segmenting the dependencies of the latent variables over time, one can expect improved generalization across spurious temporal dependencies.

In accordance with EST and our sparsely changing latent factor assumption, we introduce Gated L_0 Regularized Dynamics (GATELORD). GATELORD is a novel RNN that employs L_0 -regularized gates for latent state updates, inducing an inductive learning bias to encode piecewise constant latent state dynamics. GATELORD learns to compress dynamics, e.g. sensorimotor events, into temporally stable latent states. We show that planning with such sparsely changing latent states is beneficial in several ways, for example, by improving generalization abilities across temporal dependencies, enhancing long-horizon planning, and boosting explainability.

The main contributions of this chapter can be summarized as follows.

- We introduce a **stochastic, rectified gating function for controlling latent state updates**, which we regularize towards sparse updates using the L_0 norm.
- We demonstrate that our network performs as good or better than state-of-the-art RNNs for **prediction or control in various partially observable problems** with piecewise constant dynamics.
- We also show that the inductive bias leads to **better generalization under distributional shifts**.
- Lastly, we illustrate that the **latent states tend to compactly encode the dynamics of events** and can be easily interpreted by humans.

5.2 L_0 Regularization of Latent State Changes

Let $f_\phi : \mathcal{X} \times \mathcal{H} \rightarrow \mathcal{Y} \times \mathcal{H}$ be a recurrent neural network (RNN) with learnable parameters ϕ mapping inputs $\mathbf{x}_t \in \mathcal{X}$ and $\mathbf{h}_{t-1} \in \mathcal{H}$ the latent (hidden) state to the output $\hat{\mathbf{y}}_t \in \mathcal{Y}$

and updated latent states \mathbf{h}_t . We assume that the RNN is trained on a training dataset \mathcal{D} which consists of episodes of input-output pairs $[(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_T, \mathbf{y}_T)]$ of length T .

We want the RNN f_ϕ to learn to solve a task while maintaining piecewise constant latent states over time. The network creates a dynamics of latent states \mathbf{h}_t when applied to a sequence: $(\hat{\mathbf{y}}_t, \mathbf{h}_t) = f_\phi(\mathbf{x}_t, \mathbf{h}_{t-1})$ starting from some \mathbf{h}_0 . The most suitable measure to determine how much a time series is piecewise constant is the L_0 norm applied to temporal changes. With the change in latent state as $\Delta\mathbf{h}_t = \mathbf{h}_{t-1} - \mathbf{h}_t$, we define the L_0 -loss as

$$\mathcal{L}^{L_0}(\Delta\mathbf{h}) = \|\Delta\mathbf{h}\|_0 = \sum_{j=1} \mathbb{I}(\Delta h^j \neq 0), \quad (5.1)$$

which penalizes the **number of non-zero entries** of the vector of latent state changes $\Delta\mathbf{h}$. This regularization loss in Eq. 5.1 can be combined with a task objective to yield the overall learning objective \mathcal{L} of the network:

$$\mathcal{L}(\mathcal{D}, \phi) = \mathbb{E}_{d \sim \mathcal{D}} \left[\sum_t \mathcal{L}^{\text{task}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) + \beta \mathcal{L}^{L_0}(\Delta\mathbf{h}_t) \right] \quad (5.2)$$

with $(\hat{\mathbf{y}}_t, \mathbf{h}_t) = f_\phi(\mathbf{x}_t, \mathbf{h}_{t-1})$. The task-dependent loss $\mathcal{L}^{\text{task}}(\cdot, \cdot)$ can be, for instance, the mean-squared error for regression or cross-entropy loss for classification. The hyperparameter β controls the trade-off between the task-based loss and the desired latent state regularization. Thus, when training the network to minimize Eq. 5.2, the first term pushes the network to generate outputs according to the task at hand while the second term fosters the development of latent states that are piece-wise constant over time.

Unfortunately, we cannot directly minimize this loss using gradient-based techniques, such as stochastic gradient descent (SGD), due to the non-differentiability of the L_0 -term. Louizos et al. (2018) proposed a way to learn L_0 regularization of the learnable parameters of a neural network with SGD. They achieve this by using a set of stochastic gates that control the usage of the parameters. Each learnable parameter ϕ^j that is subject to the L_0 loss is substituted by a gated version $\phi'^j = \Theta(u^j)\phi^j$ where $\Theta(\cdot)$ is the Heaviside step function ($\Theta(u) = 0$ if $u \leq 0$ and 1 otherwise) and \mathbf{u} is determined by a distribution $q_\nu(\mathbf{u})$ with learned parameters ν . Thus, ϕ'^j is only nonzero if $u^j > 0$. This allows us to rewrite the L_0 loss (Eq. 5.1) for ϕ' as:

$$\mathcal{L}^{L_0}(\phi', \nu) = \|\phi'\|_0 = \sum_j \Theta(u^j) \quad \text{with } \mathbf{u} \sim q_\nu(\mathbf{u}), \quad (5.3)$$

where parameters ν influence sparsity and are affected by the loss.

To tackle the problem of non-differentiable binary gates, we can use a smooth approximation as a surrogate (Maddison et al., 2017; Louizos et al., 2018; Jang et al., 2017) or based on samples as in the REINFORCE algorithm (Williams, 1992). Alternatively, we can substitute its gradients during the backward pass, for example, using the straight-through estimator (Bengio et al., 2013), which treats the step function as a linear function during the backward pass. Here, we will employ the straight-through estimator.

To transfer this approach to regularize the latent state dynamics in an RNN, we require an internal gating function $\Lambda(\cdot) \in [0, 1]$, which controls whether the latent state is updated or not. For instance:

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \Lambda(\mathbf{u})\Delta\tilde{\mathbf{h}}_{t-1} \quad \text{with } \Delta\tilde{\mathbf{h}}_{t-1} = \tilde{\mathbf{h}}_t - \mathbf{h}_{t-1} \quad (5.4)$$

where $\tilde{\mathbf{h}}$ is the proposed new latent state and u is a stochastic variable depending on the current input and previous latent state and the parameters, i.e. $\mathbf{u}_t \sim q_\nu(\mathbf{u}_t | \mathbf{x}_t, \mathbf{h}_{t-1})$. For brevity, we merge the parameters ν into the overall parameter set, i.e. $\nu \subset \phi$. For computing Eq. 5.2 we need to binarize the gate by applying the step function $\Theta(\Lambda(u))$. Thus we can rewrite Eq. 5.2 as

$$\mathcal{L}(\mathcal{D}, \phi) = \mathbb{E}_{d \sim \mathcal{D}} \left[\sum_t \mathcal{L}^{\text{task}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) + \beta \mathbb{E}_j \Theta(\Lambda(u_t^j)) \right]. \quad (5.5)$$

Thus, we replace the L_0 norm on latent state changes with a penalty on latent state updates. Additionally, we average the sparsity loss over latent state dimensions which is an optional step and identical to using a different β and omitting the expectation.^{5.2}

LSTMs and GRUs use deterministic sigmoidal gates for Λ in Eq. 5.4 to determine how to update their latent state. However, it is not straightforward to apply this approach to them (detailed in Suppl. C.1). Thus, we instead introduce a novel RNN, that merges components from GRUs and LSTMs, to implement the proposed L_0 regularization of latent state changes while still allowing the network to make powerful computations. We name our network **Gated L_0 Regularized Dynamics (GATELORD)**.

5.3 GATELORD

The core of GATELORD implements the general mapping $(\hat{\mathbf{y}}_t, \mathbf{h}_t) = f_\phi(\mathbf{x}_t, \mathbf{h}_{t-1})$ using three functions, or subnetworks: (1) a *recommendation network* r_ϕ , which proposes a new candidate latent state, (2) a *gating network* g_ϕ , which determines how the latent state is

^{5.2}Averaging over latent state dimensions lets β scale the fraction of hidden state changes instead of their sum, yielding more natural ranges for the hyperparameter β . In our experiments, we use values of $\beta \in [0.0001, 0.1]$ that would require roughly $\beta \in [10^{-6}, 0.01]$ when summing the dimensions instead.

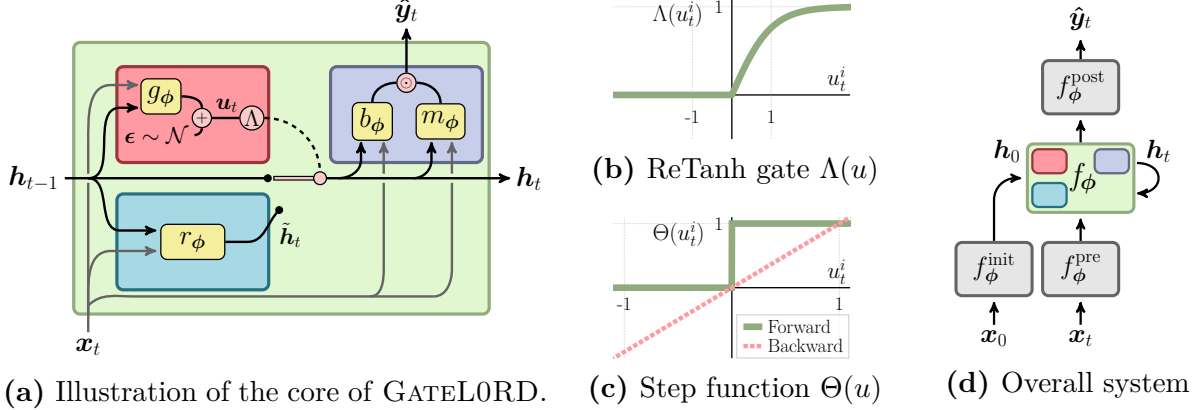


Figure 5.1: GATELORD architecture (a) GATELORD with its three subnetworks. The *gating function* controls the latent state update (red), the *recommendation function* computes a new latent state (blue) and the *output function* computes the output (purple). (b) Gate-activation function Λ (ReTanh). (c) Heaviside step function Θ and its gradient estimator. (d) Overall architecture.

updated, and (3) an *output function*, which computes the output based on the updated latent state and the input. The network is systematically illustrated in Fig. 5.1a.

The overall processing is described by the following equations:

$$\mathbf{u}_t \sim \mathcal{N}(g_\phi(\mathbf{x}_t, \mathbf{h}_{t-1}), \Sigma) \quad (\text{sample gate input}) \quad (5.6)$$

$$\Lambda(\mathbf{u}) := \max(0, \tanh(\mathbf{u})) \quad (\text{new gating function}) \quad (5.7)$$

$$\mathbf{h}_t = \mathbf{h}_{t-1} + \Lambda(\mathbf{u}_t) \odot (r_\phi(\mathbf{x}_t, \mathbf{h}_{t-1}) - \mathbf{h}_{t-1}) \quad (\text{update or keep latent state}) \quad (5.8)$$

$$\hat{\mathbf{y}}_t = b_\phi(\mathbf{x}_t, \mathbf{h}_t) \odot m_\phi(\mathbf{x}_t, \mathbf{h}_t), \quad (\text{compute output}) \quad (5.9)$$

where \odot denotes element-wise multiplication (Hadamard product).

Latent state updates We start with the control of the latent state in Eq. 5.8. Following Eq. 5.4, a new latent value is proposed by the *recommendation function* $r_\phi(\mathbf{x}_t, \mathbf{h}_{t-1})$ and the update is “gated” by $\Lambda(\mathbf{u})$. Importantly, if $\Lambda(\mathbf{u}) = \mathbf{0}$ no change in latent state occurs. Note that the update in Eq. 5.8 is, in principle, equivalent to the latent state update in GRUs (Chung et al., 2014), for which it is typically written as $\mathbf{h}_t = \Lambda(\mathbf{u}) \odot r(\mathbf{x}_t, \mathbf{h}_{t-1}) + (1 - \Lambda(\mathbf{u})) \odot \mathbf{h}_{t-1}$ with $\Lambda(\mathbf{u})$ a deterministic sigmoidal gate.

Update gate activation Because we aim for piecewise constant latent states, the activation function of the update gate Λ defined in Eq. 5.7 needs to be able to output exactly zero. A potential choice would be the Heaviside function, i.e. either copy the new latent state or keep the old one. However, this does not allow for any multiplicative

computation. So a natural choice is to combine the standard sigmoid gate of RNNs with the step-function: $\Lambda(\mathbf{u}) = \max(0, \tanh(\mathbf{u}))$ which we call ReTanh (rectified tanh)^{5.3}. Figure 5.1b shows the activation function Λ depending on its input. The gate is closed ($\Lambda(u^i) = 0$) for all inputs $u^i \leq 0$. A closed gate results in a latent state that remains constant in dimension i , i.e., $h_t^i = h_{t-1}^i$. On the other hand, for $u^i > 0$ the latent state is interpolated between the new value proposed and the old one.

Update gate inputs Motivated from the L_0 regularization in Eq. 5.3 we use stochastic inputs for the update gates. However, in our RNN setting, it should depend on the current situation. Therefore, we use a Gaussian distribution from which we sample \mathbf{u}_t with the mean determined by the *gating network* $g_\phi(\mathbf{x}_t, \mathbf{h}_{t-1})$ as defined in Eq. 5.6. We chose a fixed diagonal covariance matrix Σ , which we set to $\Sigma^{i,i} = 0.1$. To train our network using backpropagation, we implement the sampling using the *reparametrization trick* (Kingma & Welling, 2014). We introduce a noise variable ϵ and compute the gate activation as

$$\mathbf{u}_t = g(\mathbf{x}_t, \mathbf{h}_{t-1}) + \epsilon \quad \text{with } \epsilon \sim \mathcal{N}(\mathbf{0}, \Sigma). \quad (5.10)$$

During testing, we set $\epsilon = \mathbf{0}$ to achieve maximally accurate predictions.

RNN cell output Finally the output $\hat{\mathbf{y}}$ is computed from the inputs and the new latent state \mathbf{h}_t in Eq. 5.9. Inspired by LSTMs (Hochreiter & Schmidhuber, 1997), the output is determined by a multiplication of a normal branch ($b_\phi(\mathbf{x}_t, \mathbf{h}_t)$) and a sigmoidal gating branch ($m_\phi(\mathbf{x}_t, \mathbf{h}_t)$). We thus enable both additive as well as multiplicative effects of \mathbf{x}_t and \mathbf{h}_t on the output, enhancing the expressive power of the piecewise constant latent states.

Subnetwork implementations In our implementation, all subnetworks are Multi-Layer Perceptrons (MLPs). r_ϕ, b_ϕ use a tanh output activation; m_ϕ uses a sigmoid; g_ϕ has a linear output. b_ϕ, m_ϕ are one-layer networks. By default, r_ϕ, g_ϕ are also one-layer networks. However, when comparing against deep (stacked) RNNs, we increase the number of layers of r_ϕ and g_ϕ to up to three (cf. Suppl. C.2).

Differentiability We use the loss of Eq. 5.5 that is fully differentiable except for the Heaviside step function Θ . A simple approach to deal with discrete variables is to approximate the gradients by a differentiable estimator (Bengio et al., 2013; Jang et al., 2017; Maddison et al., 2017). We employ the straight-through estimator (Bengio et al., 2013), which substitutes the gradients of the step function Θ by the derivative of the linear function (see Fig. 5.1c).

Overall architecture We use GATELORD as a memory module of a more general architecture illustrated in Fig. 5.1d. The network input is preprocessed by a feed-forward network $f_\phi^{\text{pre}}(\mathbf{x}_t)$, which is an MLP or a Convolutional Neural Network (LeCun et al.,

^{5.3}Note that $\tanh(u) = 2 \cdot \text{sigmoid}(2u) - 1$.

1989) for image-like inputs. Similarly, its output is postprocessed by an MLP $f_\phi^{\text{post}}(\hat{\mathbf{y}}_t)$ (i.e. a read-out layer) before computing the loss. The latent state \mathbf{h}_0 of GATELORD could be initialized by $\mathbf{0}$. However, improvements can be achieved if the latent state is instead initialized by another network f_ϕ^{init} , a shallow MLP that sets \mathbf{h}_0 based on the first input (Mohajerin & Waslander, 2017; Ba et al., 2015).

Ablations In the Supplementary Material, we ablate various components of GATELORD, such as the gate activation function Λ (Suppl. C.3.1), the gate stochasticity (Suppl. C.3.2), the hidden state initialization through f_ϕ^{init} (Suppl. C.3.3), the multiplicative output branch m_ϕ (Suppl. C.3.4), and compare against L_1/L_2 -variants (Suppl. C.3.5).

5.4 Experiments

Our experiments offer answers to the following questions:

- **Does GATELORD generalize better to out-of-distribution inputs in partially observable domains than other commonly used RNNs?** We demonstrate both GATELORD’s ability to generalize from a 1-step prediction regime to autoregressive N -step prediction (Sec. 5.4.3) and its prediction robustness when facing action rollouts from different policies (Sec. 5.4.4).
- **Is GATELORD suitable for control problems that require (long-horizon) memory?** We reveal precise memorization abilities (Sec. 5.4.5), show that GATELORD is more sample efficient in various decision-making problems requiring memory (Sec. 5.4.6), and generalizes well across different durations of memorization (Sec. 5.4.7).
- **Are the developing latent states in GATELORD easily interpretable by humans?** Finally, we examine exemplary latent state codes and illustrate their explainability (Sec. 5.4.8).

5.4.1 Model Setup

In our experiments we compare GATELORD to LSTMs (Hochreiter & Schmidhuber, 1997), GRUs (Chung et al., 2014), and Elman RNNs (Elman, 1990). We use the architecture shown in Fig. 5.1d for all networks, only replacing the core f_ϕ . We examine the RNNs both as a prediction model for *model-predictive control* (MPC) as well as a memory module in a *reinforcement learning* (RL) setup. The networks were trained using Adam (Kingma & Ba, 2015), with learning rates and layer numbers determined through a grid search for each network type individually (cf. Suppl. C.2).

When used for prediction, the networks received input $\mathbf{x}_t = (\mathbf{o}_t, \mathbf{a}_t)$ with observations $\mathbf{o}_t \in \mathcal{O}$ and actions $\mathbf{a}_t \in \mathcal{A}$ at time t and were trained to predict the change in observation

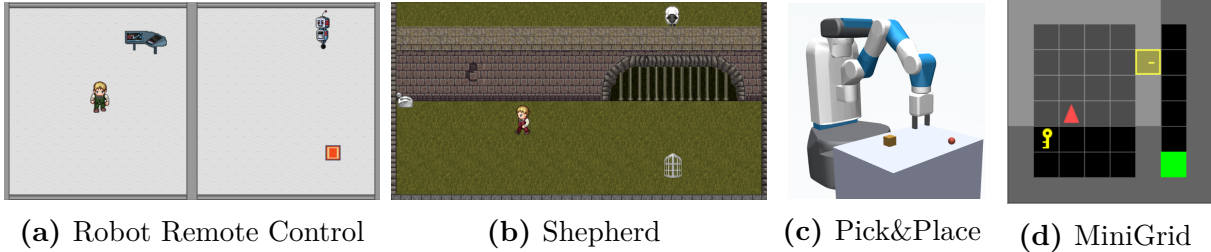


Figure 5.2: Simulation environments of GATELORD: (a) and (b) are continuous 2D-control tasks: (a) requires triggering the control of a robot by activating a computer; (b) needs memorization of the sheep’s position behind a wall to capture it later. (c) is the Fetch Pick&Place environment (Brockman et al., 2016) modified to become partially observable and (d) shows a problem (DoorKey-8x8) of the MiniGrid suite (Chevalier-Boisvert et al., 2018b).

i.e. $\mathbf{y}_t = \Delta \mathbf{o}_{t+1}$ (residual connections, detailed in Suppl. C.2.1). During testing, the next observational inputs were generated autoregressively as $\hat{\mathbf{o}}_{t+1} = \mathbf{o}_t + \hat{\mathbf{y}}_t$. Each experiment using predictive models was run with 20 random initializations.

In the RL setting (detailed in Suppl. C.2.6), the networks received as input $\mathbf{x}_t = \mathbf{o}_t$ the observation $\mathbf{o}_t \in \mathcal{O}$ and were trained as an actor-critic architecture to produce both policy and value estimations using Proximal Policy Optimization (PPO, Schulman et al., 2017). All RL experiments were performed with 10 random seeds per configuration.

5.4.2 Simulation Environments

We evaluated GATELORD in a variety of partially observable scenarios. In the **Billiard Ball** scenario a single ball, simulated in a realistic physics simulator, is shot on a pool table with low friction from a random position in a random direction with randomly selected velocity. The time series contain only the positions of the ball. This is the only considered scenario without actions purely analyzing predictions.

Robot Remote Control is a continuous control problem where an agent moves according to two-dimensional actions \mathbf{a}_t (Fig. 5.2a). Once the agent reaches a fixed position (terminal), a robot in another room is also controlled by the actions. The observable state \mathbf{o}_t is composed of the agent’s position and the robot’s position. Thus, whether the robot is controlled or not is not directly observable. When planning, the goal is to move the robot to a particular goal position (orange square).

Shepherd is a challenging continuous control problem that requires long-horizon memorization (Fig. 5.2b). Here, the agent’s actions \mathbf{a}_t are the two movement directions and a grasp action that controls whether to pick up or drop the cage. The sheep starts at the

top of the scene moving downward with a fixed randomly generated velocity. The sheep is then occluded by the wall, which masks its position from the observation. If the agent reaches the lever, the gate inside the wall opens, and the sheep appears again at the same horizontal position at the open gate. The goal is to get the sheep to enter the previously placed cage. The challenge is to memorize the sheep’s horizontal position exactly over a potentially long time to place the cage properly and to then activate the lever during mental simulation. The seven-dimensional observation \mathbf{o}_t is composed of the height of the occluder and the positions of all entities.

Fetch Pick&Place (OpenAI Gym v1, Brockman et al., 2016) is an RL benchmark task in which a robotic manipulator must move a randomly placed box (Fig. 5.2c). In our modified setting^{5.4}, the observable state \mathbf{o}_t is composed of the gripper- and object position and the relative positions of the object and fingers with respect to the gripper. The four-dimensional actions \mathbf{a}_t control the gripper position and the opening of the fingers.

MiniGrid (Chevalier-Boisvert et al., 2018b) is a gridworld suite with a variety of partially observable RL problems. At every time t , the agent (red triangle in Fig. 5.2d) receives an image-like, restricted, ego-centric view (gray area) as its observation \mathbf{o}_t ($7 \times 7 \times 3$ -dimensional). It can move forward, turn left, turn right, or interact with objects using its one-hot-encoded actions \mathbf{a}_t . The problems vary largely in their difficulty, typically contain only sparse rewards, and often involve memorization, e.g. remembering that the agent picked up a key. Suppl. C.2.7 details all MiniGrid environments examined.

5.4.3 Learning Autoregressive Predictions

First, we consider the problem of autoregressive N -step prediction in the **Billiard Ball** scenario. Here, during testing, the networks receive the first two ball positions as input and predict a sequence of 50 ball positions. We first train the RNNs using *teacher forcing*, whereby during training the real inputs are fed to the networks. Figure 5.3a shows the prediction error for autoregressive predictions. Only GATELORD with latent state regularization ($\beta = 0.001$) is capable of achieving reasonable predictions in this setup. The other RNNs seem to learn to continuously update their estimates of the ball’s velocity based on the real inputs. As a result, their prediction accuracy drops when during testing the real inputs are not available. Because GATELORD punishes continuous latent state updates, learning leads to updates of the estimated velocity only when required, i.e. upon collisions, improving its prediction robustness.

The problems of RNNs learning autoregressive prediction are well known (Bengio et al., 2015a; Lamb et al., 2016). A simple countermeasure is *scheduled sampling* (Bengio et al., 2015a), where each input is stochastically determined to be either the last output of the

^{5.4}We omit all velocities and the rotation of the object to make the scenario partially observable.

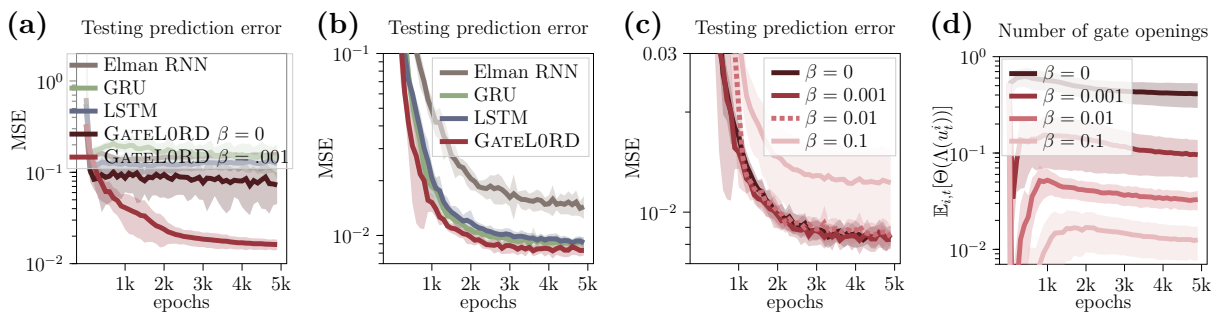


Figure 5.3: Billiard Ball prediction: prediction errors when trained using teacher forcing (a), or using scheduled sampling (b). GATELORD’s prediction error (c) and mean number of gate openings (latent state updates) (d) for different values of β . Shaded areas show \pm one standard deviation.

network or the real input. The probability of using the network output increases over the course of training. While the prediction accuracy of all RNNs improves when trained using scheduled sampling, GATELORD ($\beta = 0.001$) still achieves the lowest prediction errors (see Fig. 5.3b).

How does the regularization affect GATELORD? Figure 5.3c shows the prediction error for GATELORD for different settings of β . While a small regularization ($\beta = 0.001$) leads to the highest accuracy in this scenario, similar predictions are obtained for different strengths ($\beta \in [0, 0.01]$). Overly strong regularization ($\beta = 0.1$) degrades performance. Figure 5.3d shows the average gate openings, i.e. latent state updates, per sequence. As intended, β directly affects how often GATELORD’s latent state is updated: a higher value results in fewer gate openings, and thus fewer latent state changes. Note that even for $\beta = 0$ GATELORD learns to use fewer gates over time. We describe this effect in more detail in Suppl. C.4.1.

5.4.4 Generalization Across Policies

Traditionally, in model-based control, an agent learns directly from its own interactions with the environment (Sutton & Barto, 2018). In offline RL (Levine et al., 2020), data that was previously generated using some policy, is later reused to train a new model or policy. This is a great challenge, mainly due to the distributional shift: While the data was generated under one distribution, the model will be evaluated under a different one (Levine et al., 2020). In particular, when priorities change or a data-generating policy switches behavior, different spurious temporal correlations can easily occur in the resulting sensorimotor data. This makes training models or policies based on offline data highly

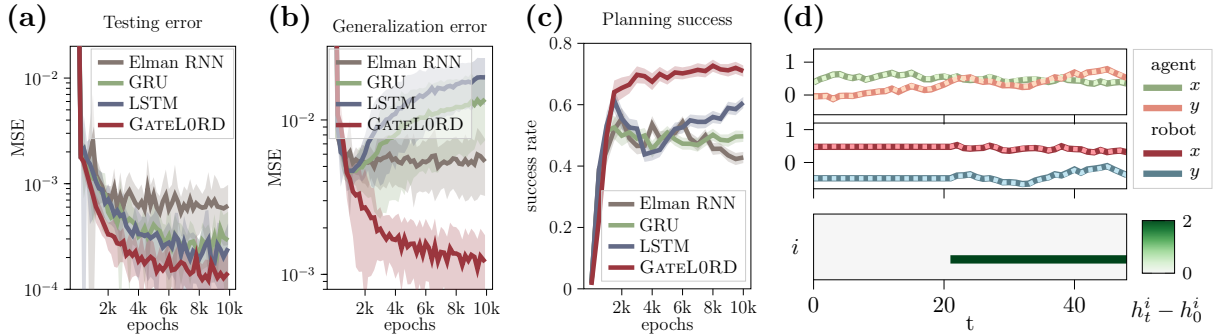


Figure 5.4: Generalization in Robot Remote Control: prediction error on the test set (a) and on the generalization set (b). Success rate for MPC (c). Shaded areas show standard deviation (a – b) or standard error (c). Exemplary generalization sequence (d) showing the agent’s positions (top), the robot’s positions (middle) with GATELORD’s predictions shown as dots, and GATELORD’s latent states (bottom).

challenging. Consequently, systems that generalize across such correlations are needed. We investigate this aspect in **Robot Remote Control**.

In Robot Remote Control the training data was generated by performing rollouts with 50 time steps of a policy that produces random but linearly magnitude-increasing actions. The magnitude of the actions in the training data is positively correlated with time, which is a spurious correlation that does not alter the underlying transition function of the environment in any way. We train different RNNs as models to predict the sequence of observations given the initial observation and a sequence of actions. We test the networks using data generated by the same policy (*test set*) and generated by a policy that samples uniformly random actions (*generalization set*). Additionally, we employ the trained RNNs for model-predictive control (MPC) using iCEM (Pinneri et al., 2021a), a random shooting method based on CEM (Rubinstein & Davidson, 1999, see also 2.2.1) that iteratively optimizes its actions to move the robot to the given goal position.

As shown in Fig. 5.4a, GATELORD ($\beta = 0.001$) outperforms all other RNNs on the test set. When tested on the generalization data (Fig. 5.4b), the prediction errors of the GRU and LSTM networks even increase over the course of training. Thus, they likely overfit to the spurious correlations in the training data. Only GATELORD is able to maintain a low prediction error. Figure 5.4c shows the MPC performance. GATELORD yields the highest success rate and outperforms all other RNNs.

Note that the other RNNs’ failure to generalize is not primarily caused by the choice of hyperparameters: Even when the learning rate of the other RNNs was optimized for the generalization set, GATELORD still outperformed them (additional experiment in

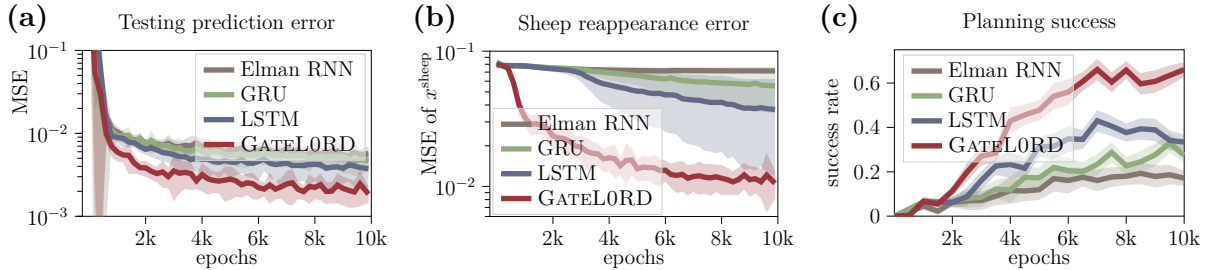


Figure 5.5: Precise memorization in Shepherd: prediction error for 100-step predictions (a) and 1-step prediction errors of the sheep’s x -position at the time step of its reappearance behind the gate (b). Success rate for capturing the sheep using MPC (c). Shaded areas show standard deviation (a-b) or standard error (c).

Suppl. C.4.3). Instead, GATELORD’s better performance is likely because it mostly encodes unobservable information within its latent state \mathbf{h}_t . This is shown exemplarily in Fig. 5.4d (bottom row) and is further analyzed in Suppl. C.4.2. The latent state remains constant, and only one dimension changes once the agent controls the robot’s position (middle row) through its actions. Because the other RNNs also encode observable information, e.g. actions, within their latent state, they are more negatively affected by distributional shifts and spurious temporal dependencies.

GATELORD’s improved generalization across temporal dependencies also holds for more complicated environments. In an additional experiment in Suppl. C.4.4 we show similar effects for the **Fetch Pick&Place** environment when trained on reach-grasp-and-transport sequences and tested to generalize across grasp timings.

5.4.5 Memorization

We hypothesized that GATELORD’s latent state update strategy fosters the exact memorization of unobservable information, which we examine in the **Shepherd** task. We tested the RNNs when predicting sequences of 100 observations given the first two observations and a sequence of actions. Again, we used the trained models for MPC using iCEM (Pinneri et al., 2021a), with the aim of catching sheep by first placing a cage and then pulling a lever. This is particularly challenging to plan because the sheep’s horizontal position needs to be memorized exactly before it is occluded for quite some time (> 30 steps) in order to accurately predict, and thus place the cage at the sheep’s future position.

Figure 5.5a shows the prediction errors during training. GATELORD ($\beta = 0.0001$) continuously achieves a lower prediction error than the other networks. Apparently, it is able to accurately memorize the sheep’s future position while occluded. To investigate the

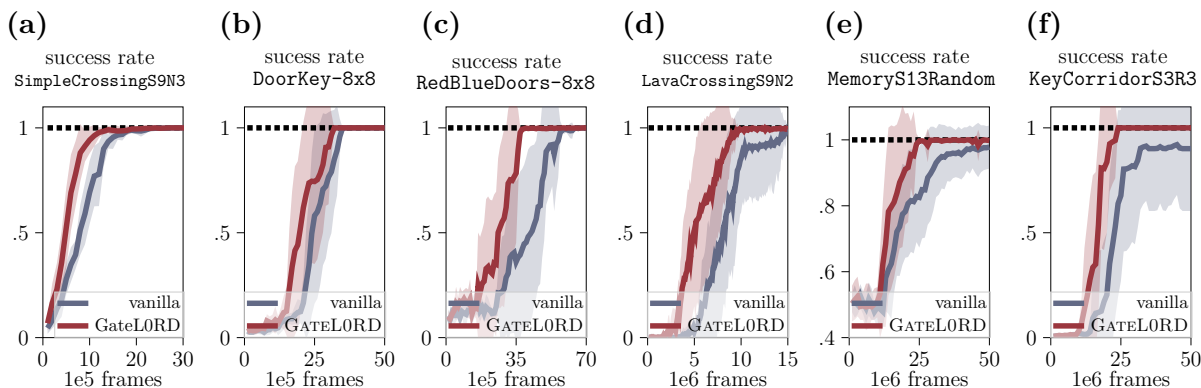


Figure 5.6: Sample efficient RL in MiniGrid: success rate in solving various tasks when GATELORD replaces an LSTM (vanilla) in a PPO architecture. Shaded areas depict the standard deviation.

memorization, we consider the situation occurring during planning: the sequence of (past) observations is fed into the network, and the prediction error of the sheep’s horizontal position at the time of reappearance is evaluated (Fig. 5.5b). Only GATELORD reliably learns to predict where the sheep will appear when the lever is activated. GRU and Elman RNNs do not noticeably improve in predicting the sheep’s position. LSTMs need more training to improve their predictions and do not reliably reach GATELORD’s level of accuracy. This is also reflected in the success rate when the networks are used for MPC (Fig. 5.5c). Only GATELORD manages to solve this challenging task with a mean success rate greater than 50%.

5.4.6 Sample Efficiency in Reinforcement Learning

Now that we have outlined some of GATELORD’s strengths in isolation, we want to analyze whether GATELORD can improve existing RL-frameworks when it is used as a memory module for POMDPs. To do so, we consider various problems that require memory in the **MiniGrid** suite (Chevalier-Boisvert et al., 2018b). Previous work (Chevalier-Boisvert et al., 2018a; Goyal et al., 2021b; Madan et al., 2021) used Proximal Policy Optimization (PPO) (Schulman et al., 2017) to solve MiniGrid problems. We took an existing architecture based on Chevalier-Boisvert et al. (2018a) (denoted as *vanilla*, detailed in Suppl. C.2.6) and replaced the internal LSTM module with GATELORD ($\beta = 0.01$). Note that we left the other hyperparameters unmodified.

As shown in Fig. 5.6 the architecture containing GATELORD achieves the same success rate or higher than the vanilla baseline in all the tasks considered. Additionally,

GATELORD is more sample efficient, i.e. it is able to reach a high success rate (Fig. 5.6) or a high reward level faster (Suppl. C.4.6). The difference in sample efficiency tends to be more pronounced for problems that require more training time. It seems that the inductive bias of sparsely changing latent states enables GATELORD to quicker learn to encode task-relevant information, e.g. key pick-up, within its latent states. Additionally, unchanged latent states enable longer backpropagation of information in time with less vanishing gradients. This could boost learning of long-horizon dependencies.

5.4.7 Zero-Shot Policy Transfer

The amount of training an RL agent needs to solve a task scales with the complexity of the problem. A simple approach to reduce training time for complex tasks is to train the agent in simpler task variants and hope that the learned policy transfers well to the more complex versions of the task. However, this requires architectures that generalize well across problem complexity, e.g. environment size. Previous experiments revealed GATELORD’s precise memorization and its robustness towards spurious temporal correlations. Thus, we hypothesize that GATELORD can generalize well across task horizons.

We investigate this aspect in **MiniGrid** by training an RL agent (as in 5.4.6) containing an LSTM (vanilla) or GATELORD on two problems that require memory, i.e. **DoorKey-8x8** (Fig. 5.2d) and **MemoryS13Random**. We evaluate the learned policies on the same problems in larger environments, i.e. **DoorKey-16x16** and **MemoryS17Random**. Figure 5.7 plots the success rate in solving the more complex problems over training time in the simpler variants. For both problems GATELORD achieves a higher success rate. The better generalization performance cannot simply be explained by GATELORD’s better performance on these types of problem. In training tasks, both architectures reach approximately the same success rate after sufficient training (cf., Fig. 5.6b). Instead, the higher success rate during zero-shot policy transfer is likely due to GATELORD generalizing better across task horizons.

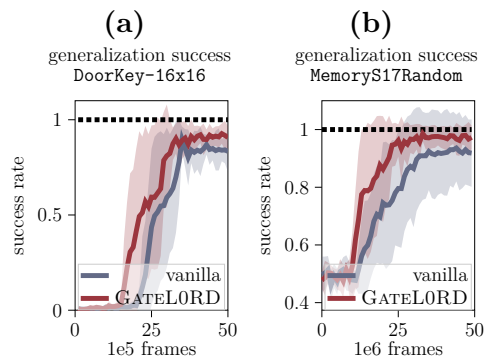


Figure 5.7: Zero-shot policy transfer in MiniGrid: We plot zero-shot success rate upon policy transfer to new problems. Shaded areas depict the standard deviation.

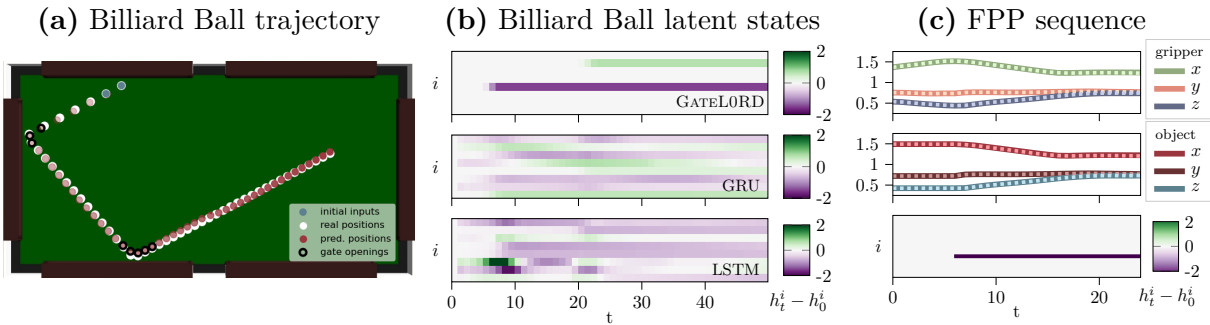


Figure 5.8: Explainability of the latent states: (a) Billiard Ball trajectory for GATELORD ($\beta = 0.01$) with real positions (white), provided inputs (blue), and predicted positions (red, saturation increasing with time). The inputs for which at least one gate opened are outlined in black. (b) The latent states for the depicted trajectory for GATELORD, GRU, and LSTM (cell states). (c) Fetch Pick&Place (FPP) sequence with real (solid) and predicted (dotted) positions of gripper (top) and object (middle) and GATELORD’s latent states (bottom). Latent states are shown relative to initialization, i.e. $\mathbf{h}_t - \mathbf{h}_0$.

5.4.8 Explainability of the Latent States

Lastly, we analyze the latent representations of GATELORD, starting with **Billiard Ball**. Figure 5.8a shows one exemplary ball trajectory in white and the prediction in red. Inputs for which at least one gate opened are outlined in black. Figure 5.8b shows the corresponding latent states \mathbf{h}_t relative to the initial latent state \mathbf{h}_0 . GATELORD updates two dimensions of its latent states around the points of collisions to account for the changes in x - and y -velocity of the ball. For $\beta = 0.01$ we find on average only two latent state dimensions change per sequence (see Suppl. C.4.1), which hints at a tendency to encode x - and y -velocity using separate latent dimensions. In contrast, the exemplary latent states of the GRU and LSTM networks shown in Fig. 5.8b are not as easily interpretable.

For **Robot Remote Control**, GATELORD ($\beta = 0.001$) updates only its latent state once it controls the robot (exemplary shown in Fig. 5.4d). Thus, the latent state clearly encodes control over the robot. We use the **Fetch Pick&Place** scenario as a higher-dimensional problem to investigate latent state explainability when training on grasping sequences (detailed in Suppl. C.2.5). Here, GATELORD updates the latent state typically when the object is grasped (exemplary shown in Fig. 5.8c). This hints at an encoding of ‘object transportation’ using one dimension. Other RNNs do not achieve such a clear representation, neither in Robot Remote Control nor in Fetch Pick&Place (see Suppl. C.4.2 and C.4.4 for more examples).

5.5 Related Work

Structural regularization of latent updates: Pioneering work on regularizing latent updates was done by Schmidhuber (1992) who proposed the Neural History Compressor, a hierarchy of RNNs that autoregressively predict their next inputs. Thereby, the higher-level RNN only becomes active and updates its latent states, if the lower-level RNN fails to predict the next input. To structure latent state updates, the Clockwork RNN (Koutnik et al., 2014) partitions the hidden neurons of an RNN into separate modules, where each module operates at its own predefined frequency. Along similar lines, Phased LSTMs (Neil et al., 2016) use gates that open periodically. The update frequency in Clockwork RNNs and Phased LSTMs does not directly depend on the world state, but only on a predefined time scale.

Loss-based regularization of latent updates: Krueger & Memisevic (2015) have proposed using an auxiliary loss term that punishes the change in L_2 -norms of the latent state, which results in piecewise constant norms but not dynamics of the hidden states.

Binarized update gates: Closely related to our ReTanh, Skip RNNs (Campos et al., 2018) use a binary gate to determine latent state update decisions. In Skip RNNs, the gating depends only on the previous latent state and not the input. Thus, given a particular latent state, the time required until the update is performed is fixed. Similarly, Gumbel-Gate LSTMs (Li et al., 2018) replace sigmoid input gates and forget gates with stochastic binary gates, approximated by a Gumbel-Softmax estimator (Jang et al., 2017). However, sparsity of the latent state updates is not incentivized. Selective-Activation RNNs (SA-RNNs) (Hartvigsen et al., 2020) modify a GRU by masking the latent state with deterministic, binary gate. Along similar lines, Variational Sparse Gating (VSG) (Jain et al., 2022) samples latent state updates in a GRU from a Bernoulli distribution. Both SA-RNNs and VSG incentivize sparse updates via an auxiliary loss term. However, for GRUs the network output corresponds to the networks’ latent state, thus, a piecewise constant latent state will result in piecewise constant outputs.

Attention-based latent state updates: Sparse latent state updates can also be achieved using attention (Graves et al., 2014; Bahdanau et al., 2015; Vaswani et al., 2017). Neural Turing Machines (Graves et al., 2014) use an attention mechanism to update an external memory block. Thereby, the attention mechanism can focus and only modify particular locations within the memory. Recurrent Independent Mechanisms (RIMs) (Goyal et al., 2021b) use a set of recurrent cells that only interact sparsely with the environment and one another through competition and a bottleneck of attention. Recent extensions explore the update of the cells and the attention parameters at different time scales (Madan et al., 2021). For RIMs, the sparsity of the latent state changes is predefined via a hyperparameter that sets the number of active cells. In contrast, our L_0

loss implements a soft constraint.

Event-encoding latent states Inspired by studies on event cognition, a number of RNNs have been proposed to compress events into stable latent codes. Reynolds et al. (2007) employed an update gate that modifies the latent state of an RNN when prediction errors exceed a threshold. Along similar lines, gated surprise (Humaidan et al., 2020) or surprise in combination with counterfactual regularization (Humaidan et al., 2021) was used to update parts of an RNN latent state. In REPRISE (Butz et al., 2019), latent states were inferred to retrospectively explain recent sensorimotor experiences. All of these approaches managed to develop latent states encoding events. However, so far, each technique has been used only on a family of highly related problems. Thus, it still needs to be shown whether these approaches transfer to other environments and scale to more complex problems.

Transformers: Transformers (Vaswani et al., 2017) omit memory altogether, processing a complete sequence for every output at once using query-key-based attention. While this avoids problems arising from maintaining a latent state, their self-attention mechanism comes with high computational costs. Transformers have shown breakthrough success in natural language processing. Recently, RL and planning systems have also started to include transformers (Parisotto et al., 2020; Chen et al., 2022; Robine et al., 2023; Micheli et al., 2023); however, it remains challenging to train them for these applications (Parisotto et al., 2020).

Structured State Space Sequence: Concurrently to our work, the problem of long-horizon memory was tackled by the Structured State Space Sequence (S4) models (Gu et al., 2021). S4 builds on the state space model (SSM), a classical model from control theory for continuous-timed inputs (Bertsekas, 2012). S4 learns the involved matrices of a linear SSM and an appropriate step size for handling discrete inputs through deep learning. Crucially, S4 is initialized with a specialized, structured state matrix (HiPPO matrix, Gu et al., 2020), which has been shown to capture long-ranging dependencies (Gu et al., 2021). In contrast to other RNNs, S4 brings the computational benefit that it offers parallel computation, i.e. S4 can process a full time series at once by applying a convolutional filter (Gu et al., 2021; Deng et al., 2023). S4 has been shown to learn dependencies over thousand of time steps (Gu et al., 2021). Although originally the S4 layer was designed to operate on one-dimensional inputs and outputs, extensions transferred the approach to multi-input-output layers (Smith et al., 2022). Recent work showed that S4-based models are well suited for model-based RL (Deng et al., 2023).

5.6 Discussion

We have introduced a novel RNN architecture (GATELORD), which implements an inductive bias to develop sparsely changing latent states. The bias is realized by a gating mechanism, which minimizes the L_0 norm of latent updates. In several empirical evaluations, we quantified and analyzed the performance of GATELORD in various prediction and control tasks. The results support our hypothesis that networks with piecewise constant latent states can generalize better to distributional shifts of the inputs, ignore spurious time dependencies, and enable precise memorization. This translates into improved performance for both model-predictive control (MPC) and reinforcement learning (RL). Moreover, we demonstrated that the latent space becomes interpretable, which is important for explainability reasons.

Our approach introduces an additional hyperparameter which controls the trade-off between the task at hand and latent space constancy. When chosen in favor of explainability, it can reduce the in-distribution performance while improving its generalization abilities. When the underlying system has continuously changing latent states, our regularization is counterproductive. As demonstrated by an additional experiment in Suppl. C.4.5, the unregularized network performs well in such cases.

Our sparsity-biased gating mechanism segments sequences into chunks of constant latent activation. These latent dynamics can show a striking similarity to what has been proposed for the temporally persistent activity of event models of human cognition (see Sec. 2.1). For example, while the robot reaches for an object in the Fetch Pick&Place simulation the latent state remains unchanged. Once the object is grasped by the gripper, the latent state is updated but remains unchanged while the object is carried. This suggests event codes for <reaching for an object> or <lifting an object> that develop within GATELORD. Thus, when embedded in a suitable sensorimotor learning architecture, GATELORD develops latent states that compactly encode sensorimotor events.

The general approach of this thesis, THICK MDPs outlined in Chap. 3, proposes that hierarchical models with nested time scales can develop based on temporally persistent event codes. In this chapter, we demonstrated that GATELORD is a practical deep learning method to learn such codes with minimal supervision.^{5.5} Thus, GATELORD may allow us to develop hierarchical predictions based on the discrete dynamics of its latent state – a direction we will explore in the next chapter.

^{5.5}The only explicit supervision needed is correctly setting the hyperparameter β .

6

Hierarchical Predictions from Discrete Latent Dynamics^{6.1}

Events and their boundaries seem like ideal representations for the segmentation and hierarchical compression of activity. With GATELORD we now have a practical deep learning method for encoding events as sparsely updated latent states. In this chapter, we use GATELORD to learn hierarchical predictions in a self-supervised way. Our system trains a high-level model based on the sparse latent state changes of a low-level model. For simulated robotic manipulations, we show that the high level learns to make meaningful temporal abstract predictions. Furthermore, when applying the gaze selection strategy of CAPRI, our system develops goal-anticipatory gaze behavior as found in eye-tracking studies with infants thereby overcoming simplifications of our previous model.

^{6.1}This chapter is based on the publication:

Gumbsch, C., Adam, M., Elsner, B., Martius, G. & Butz, M. V. (2022). Developing Hierarchical Anticipations via Neural Network-based Event Segmentation. *IEEE International Conference on Development and Learning (ICDL 2022)*, 1-8.

The text has been slightly revised and shortened, mainly to avoid repetitions from previous chapters. The figures have been adopted except for minor cosmetic changes. In accordance with the publication and to recognize the contributions of my coauthors, this chapter is written in first person plural (we).

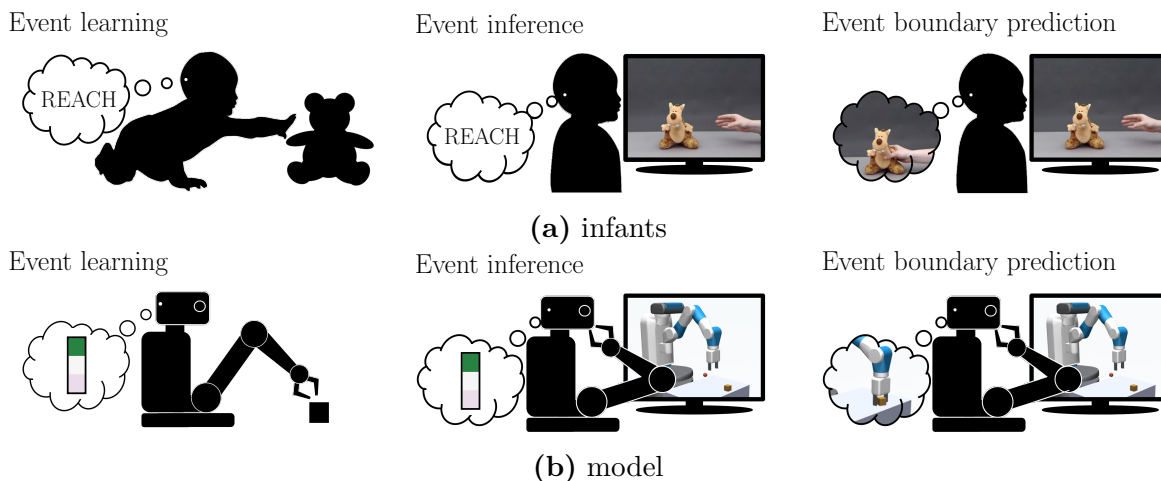


Figure 6.1: Hypothesis for modeling goal anticipations via event codes: (a) Infants first learn event encodings from their interaction with the world (left). When they see familiar movements, these encodings get activated (middle), enabling the anticipation of the next event boundary, i.e. the action consequences (right). (b) We model this process in artificial systems, which learn event codes in a self-supervised manner. Screenshots are taken from Adam & Elsner (2020).

6.1 Introduction

Humans are able to generate hierarchical predictions on various time scales. This ability seems to develop already during the first year of life, as indicated by findings on goal-anticipatory gaze shifts (reviewed in detail in Sec. 4.2.1): Infants start to look at the goal of an action event before it is finished. Goal-anticipatory gaze shifts start to arise with approximately six months of age (Kanakogi & Itakura, 2011; Adam & Elsner, 2020). These goal anticipations seem to be to some degree temporal abstract, seeing that they do not depend on the duration of the event as much as on other factors, e.g. agent familiarity, action familiarity, the ability to execute the action themselves, agency cues, and action effects (Gredebäck et al., 2010; Kanakogi & Itakura, 2011; Adam et al., 2016; Adam & Elsner, 2018, 2020; Adam et al., 2021; Elsner & Adam, 2021).

In Chap. 4 we have introduced Cognitive Action PRediction in Infants (CAPRI), a computational model based on theories of hierarchical event-predictive learning (Zacks & Tversky, 2001; Zacks et al., 2007; Butz, 2016; Butz et al., 2021; Elsner & Adam, 2021) and active inference (Friston et al., 2016; Parr et al., 2022) to explain these findings. We briefly review the model here and illustrate the modeling hypothesis in Fig. 6.1. Our main

assumption is that infants learn to hierarchically compress their sensorimotor experiences into event schemata. Via these event schemata, they not only learn to predict how an event typically unfolds, but also learn to make temporal abstract predictions about upcoming event boundaries. Once learned, the observation of a known event, e.g., a hand moving towards an object, results in the activation of the matching event schema. Through this event schema, infants can directly anticipate the next event boundary, i.e. the hand touching the object. This activated schema leads to the generation of goal-anticipatory gaze shifts, striving to minimize uncertainty about when, where, and how the event will end.

Although CAPRI was able to model a variety of findings, the existing implementation has several limitations (see Sec. 4.6). Most notably, CAPRI learns event models based on supervised segmentations of sequences. Since event segmentation is a crucial aspect in the development of event schemata (Zacks et al., 2007), this could raise doubts about the cognitive plausibility of the model. Additionally, this prevents applying CAPRI to other scenarios in which events are unknown.

In Chap. 5, we have introduced Gated L_0 Regularized Dynamics (GATELORD), a novel RNN that only sparsely updates its latent state when the task demands it. We saw that for some of the investigated tasks, GATELORD developed temporally stable latent states that seem to encode simple sensorimotor events, such as reaching or lifting. Our general approach THICK MDPs (Chap. 3) proposes that hierarchical predictions can develop from temporally stable event codes. Thus, through GATELORD we could improve the modeling approach CAPRI by giving the model the ability of self-supervised event segmentation and developing hierarchical predictions from scratch.

In this chapter we introduce a hierarchical recurrent neural network architecture, which can be trained end-to-end. The architecture learns to compress sensorimotor sequences into sparsely changing latent states, developing event-encoding latent states from scratch. Based on these event codes, the system learns temporal abstract event boundary predictions, based on the principles of THICK MDPs (see Chap. 3), that anticipate the observations encountered upon latent state changes. Striving to minimize uncertainty across its hierarchical predictions, the system develops goal-anticipatory behavior similar to the way it develops in infants.

The main contributions of this chapter are the following:

- We develop a hierarchical architecture in which a higher level learns to predict the situations in which the latent state of a lower level changes. This is the **first proof of concept** that our algorithm for learning **Temporal Hierarchies from Invariant Context Kernels** (THICK, Chap. 3) works in practice.
- We show for different interactions of a simulated robotic manipulator that **meaningful event boundary predictions** develop at the high level.

- We use our system to **model the development of goal-anticipatory gaze behavior** in infants. Thereby, we **improve our previous modeling approach** in several dimensions, namely, self-supervised event segmentation, predicting nonlinear dynamics, and mental simulations of actions.

6.2 Learning Hierarchical Predictions

We present a neural network architecture that learns to process time series with sparsely changing latent states, modulating predictive forward-inverse models. Furthermore, it learns to predict situations where latent states tend to change. These predictive abilities enable goal-anticipatory behavior. We now detail our architecture and describe (1.) how its inductive biases enable it to maintain sparsely changing latent states, (2.) how we embed it in a model such that these states encode the sensorimotor dynamics of events, (3.) which allows learning event boundary predictions, and (4.) enable goal-anticipatory gaze behavior.

6.2.1 Sparsely Changing Latent States

We want our system to encode the dynamics of events via latent temporally persistent states $\mathbf{h}_t \in \mathbb{R}^H$. Thus, to develop such sparsely changing codes, the core of our system is implemented by a GATELORD cell f_ϕ^{core} (green in Fig. 6.2) Here, we briefly review crucial aspects of GATELORD and our notation. We refer to Chap. 5 for a more detailed review.

GATELORD uses three functions, or subnetworks, a *recommendation function* r_ϕ , a *gating function* g_ϕ , and an *output function* o_ϕ , which together route sensorimotor prediction dynamics and maintain an inner latent state \mathbf{h}_t , as follows:

$$\hat{\mathbf{h}}_t = r_\phi(\mathbf{x}_t, \mathbf{h}_{t-1}) \quad (\text{latent state proposal}) \quad (6.1)$$

$$\mathbf{\Lambda}_t = \max(\mathbf{0}, \tanh(g_\phi(\mathbf{x}_t, \mathbf{h}_t))) \quad (\text{update gate}) \quad (6.2)$$

$$\mathbf{h}_t = \mathbf{\Lambda}_t \odot \hat{\mathbf{h}}_t + (\mathbf{1} - \mathbf{\Lambda}_t) \odot \mathbf{h}_{t-1} \quad (\text{update or keep latent state}) \quad (6.3)$$

$$\hat{\mathbf{y}}_t = o_\phi(\mathbf{x}_t, \mathbf{h}_t) \quad (\text{compute output}) \quad (6.4)$$

with \odot the Hadamard product. The recommendation function r_ϕ proposes a new latent state $\hat{\mathbf{h}}_t$ (Eq. 6.1). The gating function g_ϕ computes an *update gate* $\mathbf{\Lambda}_t \in [0, 1]$ (Eq. 6.2) with a rectified tanh activation function. If $\Lambda_t^i > 0$ the latent state is updated at dimension i (Eq. 6.3). The network output is then determined using the output function o_ϕ (Eq. 6.4).

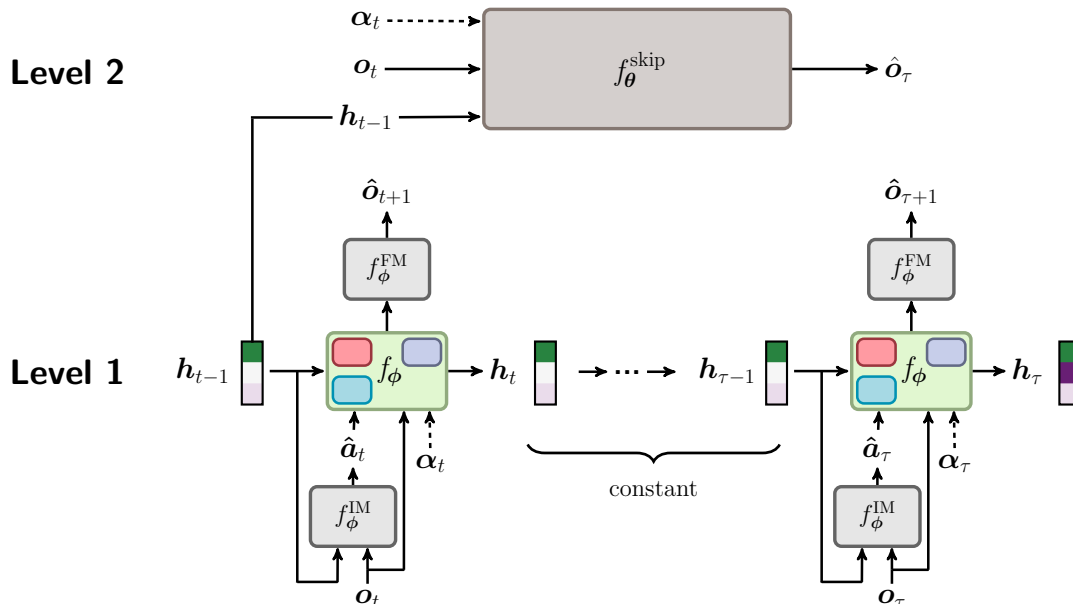


Figure 6.2: Skip Network Architecture: The forward-inverse model f_ϕ and its latent states \mathbf{h}_t are unrolled over time (**Level 1**, bottom). The skip network f_θ^{skip} (**Level 2**, top) predicts input observation \mathbf{o}_τ for which the latent states change. The gaze input α_t (dashed lines) is only used for modeling infants’ goal anticipation (see Sec. 6.2.4).

To enforce sparse latent state updates, GATELORD is trained using the loss:

$$\mathcal{L}(\mathcal{D}, \phi) = \mathbb{E}_{\mathcal{D}} \left[\sum_t \mathcal{L}^{\text{task}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) + \beta \mathcal{L}^{\text{sparse}}(\Lambda_t) \right] = \mathbb{E}_{\mathcal{D}} \left[\sum_t \mathcal{L}^{\text{task}}(\hat{\mathbf{y}}_t, \mathbf{y}_t) + \beta \mathbb{E}_i \Theta(\Lambda_t^i) \right], \quad (6.5)$$

where $\mathcal{L}_{\text{task}}$ is the task-based loss, e.g. Mean Squared Error, Θ is the Heaviside step function, \mathcal{D} a training data set, and ϕ are trainable parameters of the network. The sparsity loss $\mathcal{L}^{\text{sparse}}$ punishes latent state changes and thus fosters the tendency to develop piecewise constant latent states \mathbf{h}_t . The gradients of the Heaviside step function are approximated by the straight-through estimator (Bengio et al., 2013). The hyperparameter β modifies the influence strength of the regularization term.

6.2.2 Forward-Inverse Model

We use GATELORD as a memory module, which is embedded into a predictive forward-inverse model structure as shown in Fig. 6.2. In every time step, the model receives an observation \mathbf{o}_t and an action \mathbf{a}_t as its input. At time t the sensorimotor inputs $(\mathbf{a}_t, \mathbf{o}_t)$ are

fed into GATELORD and a network output $\hat{\mathbf{y}}_t$ and a new latent state \mathbf{h}_t are computed as $(\hat{\mathbf{y}}_t, \mathbf{h}_t) = f_\phi^{\text{core}}(\mathbf{o}_t, \mathbf{a}_t, \mathbf{h}_{t-1})$. The network outputs $\hat{\mathbf{y}}_t$ are processed by a forward model f_ϕ^{FM} , which predicts the next sensory observation $\hat{\mathbf{o}}_{t+1}$:

$$\hat{\mathbf{o}}_{t+1} = f_\phi^{\text{FM}}(\hat{\mathbf{y}}_t). \quad (6.6)$$

Based on the updated latent state \mathbf{h}_{t+1} and the next observation \mathbf{o}_{t+1} , an inverse model f_ϕ^{IM} predicts the next action $\hat{\mathbf{a}}_{t+1}$:

$$\hat{\mathbf{a}}_{t+1} = f_\phi^{\text{IM}}(\mathbf{o}_{t+1}, \mathbf{h}_{t+1}). \quad (6.7)$$

GATELORD’s initial latent state \mathbf{h}_0 is determined by an initialization model f_ϕ^{init} based on the first sensorimotor inputs:

$$\mathbf{h}_0 = f_\phi^{\text{init}}(\mathbf{a}_1, \mathbf{o}_1). \quad (6.8)$$

Our system implements a fully predictive sensorimotor model that attempts to constantly predict its next sensory input as well as the next action. However, in many situations, the prediction accuracy varies drastically across states of the environment. For example, depending on the agent’s gaze, different parts of the observation may be noisy or focused. Similarly, while in some state the model can be very certain about the agent’s next actions, in other states a variety of potential actions could follow. Thus, we want the sensorimotor predictions of our network to be probabilistic, reflecting the heteroscedastic uncertainty in the world. Instead of directly predicting the next observation $\hat{\mathbf{o}}_{t+1}$, the forward model f_ϕ^{FM} predicts a probability distribution $P_\phi(\hat{\mathbf{o}}_{t+1} | \mathbf{o}_t, \mathbf{a}_t, \mathbf{h}_t)$, from which an observation prediction $\hat{\mathbf{o}}_{t+1}$ is sampled. We model $P_\phi(\hat{\mathbf{o}}_{t+1} | \mathbf{o}_t, \mathbf{a}_t, \mathbf{h}_t)$ as a normal distribution with

$$P_\phi(\hat{\mathbf{o}}_{t+1} | \mathbf{o}_t, \mathbf{a}_t, \mathbf{h}_t) = \mathcal{N}(\mathbf{o}_t + \boldsymbol{\mu}_{t,\Delta o}, \boldsymbol{\Sigma}_{t,o}). \quad (6.9)$$

The mean $\boldsymbol{\mu}_{t,\Delta o}$ and a diagonal covariance matrix $\boldsymbol{\Sigma}_{t,o}$ are predicted by the forward model f_ϕ^{FM} . Equivalently, for the distribution over actions $P_\phi(\hat{\mathbf{a}}_{t+1} | \mathbf{o}_{t+1}, \mathbf{h}_t)$ the inverse model f_ϕ^{IM} outputs a mean $\boldsymbol{\mu}_{t,a}$ and diagonal covariance matrix $\boldsymbol{\Sigma}_{t,a}$. We model the probability distribution as

$$P_\phi(\hat{\mathbf{a}}_{t+1} | \mathbf{o}_{t+1}, \mathbf{h}_t) = \mathcal{N}(\boldsymbol{\mu}_{t,a}, \boldsymbol{\Sigma}_{t,a}). \quad (6.10)$$

We can train the overall architecture to minimize the negative log likelihood (NLL) loss. The overall loss of our forward-inverse model is defined as

$$\mathcal{L}(\mathcal{D}, \phi) = \mathbb{E} \left[\sum_t -\log (P_\phi(\mathbf{o}_{t+1} | \mathbf{o}_t, \mathbf{a}_t, \mathbf{h}_t)) \right. \\ \left. - \underbrace{\log (P_\phi(\mathbf{a}_{t+1} | \mathbf{o}_{t+1}, \mathbf{h}_t))}_{\text{prediction loss}} + \underbrace{\beta \mathcal{L}^{\text{sparse}}(\Lambda_t)}_{\text{regularization}} \right]. \quad (6.11)$$

This corresponds to the original loss from GATELORD (Eq. 6.5) with the task-based loss $\mathcal{L}^{\text{task}}$ being the summed NLL of actions and observations (prediction loss). In practice, we use the recently proposed β -NLL loss^{6.2} (Seitzer et al., 2022) ($\beta^{\text{NLL}} = 0.5$), to avoid instabilities during training and achieve high-quality predictions. We model all components apart from GATELORD, i.e. f_{ϕ}^{init} , f_{ϕ}^{FM} , and f_{ϕ}^{IM} , as feed-forward neural networks.

Can we expect that the resulting latent states \mathbf{h}_t will resemble event encoding, as we outlined them in Sec. 2.1? Event Segmentation Theory (Zacks et al., 2007) proposes that event models are only updated upon transient prediction errors, or surprise. Equation 6.11 essentially phrases this idea in the form of a loss function: the system is trained to minimize prediction errors of future sensorimotor inputs. In parallel, the regularization loss punishes changes in its latent state. Thus, the system learns to change the latent state only when transient prediction errors outweigh the regularization loss. According to the Theory of Event Coding (Hommel et al., 2001), event codes should encode both actions as well as sensory effects in a common code. The latent states \mathbf{h}_t will encode both sensory- and motor-predictive aspects, because they are used to predict both the next sensory states (Eq. 6.6) and the next actions (Eq. 6.7).

6.2.3 Event Boundary Predictions

A crucial characteristic of event encodings is that they seem to be hierarchically organized in a partonomy with nested time scales (Zacks & Tversky, 2001). THICK MDPs, our general framework of event-based decision making detailed in Chap. 3, proposes that this nested hierarchy may develop when a high level in the hierarchy is only updated when the next lower level adapts a temporally stable event code.

To implement this idea, we add a second-level neural network f_{θ}^{skip} with learnable parameters θ , which we call the *skip network*. The skip network is designed to predict the observation that will occur at the next event boundary, that is, at the end of the current event and at the start of a next event. For example, the sensation of one’s hand touching an object could characterize the ending of a <reaching>-event and the start of a <grasping>-event.

Let us assume that our system encodes an event with the same latent state \mathbf{h}_t . At an event boundary, the latent state changes, which is caused by the opening of an update gate $\Lambda_t^i > 0$ (Eq. 6.3). We can use this segmentation of time series to generate the training data for the skip network in a self-supervised manner, illustrated in Fig. 6.2. To generate the training data, we feed a sensorimotor sequence $((\mathbf{o}_1, \mathbf{a}_1), (\mathbf{a}_2, \mathbf{o}_2), \dots, (\mathbf{o}_T, \mathbf{a}_T))$ through our

^{6.2}The β -NLL loss scales the NLL gradients by a β^{NLL} -exponentiated per-sample variance for a hyper-parameter $\beta^{\text{NLL}} \in [0, 1]$. $\beta^{\text{NLL}} = 0$ corresponds to the normal NLL loss. For $\beta^{\text{NLL}} = 1$ the mean $\boldsymbol{\mu}$ is learned as when using MSE loss.

forward-inverse model f_ϕ to receive a sequence of latent states and update gate openings $((\mathbf{h}_1, \mathbf{\Lambda}_1), (\mathbf{h}_2, \mathbf{\Lambda}_2), \dots, (\mathbf{h}_T, \mathbf{\Lambda}_T))$. We define a function $\tau(\cdot)$ that maps every time step t to the next event boundary, with

$$\tau(t) = \min (\{ \tau \mid t < \tau \leq T \wedge \mathbf{\Lambda}_\tau > \mathbf{0} \}). \quad (6.12)$$

Thus, $\tau(t)$ determines the next time step after t for which at least one gate opens.^{6.3} We train the skip network to predict $\mathbf{o}_{\tau(t)}$ from the observation \mathbf{o}_t and latent state \mathbf{h}_t . In other words, the skip network is trained to predict the final observation of an event from an arbitrary starting point within this event. We treat the last time step T of a sequence also as an event boundary.

To enable the skip network to predict uncertainties, we model its output as a distribution of observations $P_\theta(\hat{\mathbf{o}}_{\tau(t)} \mid \mathbf{o}_t, \mathbf{h}_t)$ and use a multivariate Gaussian distribution parametrization

$$P_\theta(\hat{\mathbf{o}}_{\tau(t)} \mid \mathbf{o}_t, \mathbf{h}_t) = \mathcal{N}(\hat{\mathbf{o}}_{\tau(t)} \mid \boldsymbol{\mu}_{t, \mathbf{o}_{\tau(t)}}, \boldsymbol{\Sigma}_{t, \mathbf{o}_{\tau(t)}}) \quad (6.13)$$

with mean $\boldsymbol{\mu}_{t, \mathbf{o}_{\tau(t)}}$ and diagonal covariance matrix $\boldsymbol{\Sigma}_{t, \mathbf{o}_{\tau(t)}}$ predicted by the skip network. The skip network f_θ^{skip} is trained to minimize the loss $\mathfrak{L}(\mathcal{D}, \theta)$ with

$$\mathfrak{L}(\mathcal{D}, \theta) = \mathbb{E}_{\mathcal{D}} \left[-\log (P_\theta(\mathbf{o}_{\tau(t)} \mid \mathbf{o}_t, \mathbf{h}_t)) \right], \quad (6.14)$$

for its learnable parameters θ . As before, we use the β -NLL loss (Seitzer et al., 2022) ($\beta^{\text{NLL}} = 0.5$). We model the skip network as an MLP.

6.2.4 Modeling Goal Anticipations

As previously done in CAPRI (Chap. 4), we use the structure of events and event boundaries to model experimental findings on infants’ goal-anticipatory gaze. For modeling infant gaze (experiments in Sec. 6.3.5) we modify the inputs of the two networks. Here, the forward-inverse model f_ϕ and the skip network f_θ^{skip} receive a simplified gaze or attentional focus $\boldsymbol{\alpha}_t$ as an additional input. As in Chap. 4, we implement this focus as a noise mask on the input observations. When the system attends to an entity \mathbf{e} , e.g. the agent’s hand, all dimensions of the input observation $\mathbf{o}_t^{\mathbf{e}}$ concerning this entity receive no sensory noise, e.g. the position of the hand. All other dimensions of \mathbf{o}_t , for example, the position of an object on a table, are masked by sensory noise. In this way, we model simplified gaze behavior: The system can focus on entities, akin to looking at them, to gain clear sensory observations, whereas unattended entities receive sensory noise.

^{6.3}Note, that for the outlined architecture this implements the general function Eq. 3.1, that we previously defined for THICK MDPs.

In Chap. 4, we proposed that infants’ goal-anticipatory gaze emerges from selecting gaze behavior that minimizes uncertainty across hierarchical predictions. According to CAPRI, infants partially direct their gaze to minimize uncertainty within the ongoing event to better predict the next sensory observations (event dynamics). In addition, they attempt to minimize uncertainty about future event boundaries to better predict when, where, and how future events will follow (event boundary). In our system we can express this idea as

$$\boldsymbol{\alpha}_t = \arg \min_{\boldsymbol{\alpha}} \left[\underbrace{U(P_{\phi}(\hat{\boldsymbol{o}}_{t+1} \mid \boldsymbol{o}_t, \boldsymbol{a}_t, \boldsymbol{h}_t, \boldsymbol{\alpha}))}_{\text{event dynamics uncertainty}} + \underbrace{U(P_{\theta}(\hat{\boldsymbol{o}}_{\tau(t)} \mid \boldsymbol{o}_t, \boldsymbol{h}_t, \boldsymbol{\alpha}))}_{\text{event boundary uncertainty}} \right], \quad (6.15)$$

where U is an uncertainty measurement and P_{ϕ} and P_{θ} are the Gaussian distributions over observations generated by the forward-inverse-model f_{ϕ} and skip network f_{θ}^{skip} , respectively. We use the following simple uncertainty estimation

$$U(P) = \sum_{i \in I} \Sigma_{t,P}^{i,i}, \quad (6.16)$$

which is the sum of variances^{6.4}, with $\Sigma_{t,P}$ the predicted covariance matrix of distribution P , i.e. either P_{ϕ} or P_{θ} , and I a set of relevant dimensions. The set of observation indices I allows us to specify for which parts of the observation the system should aim to minimize uncertainty, e.g., focusing only on future hand positions.

6.3 Experiments

Our experiments set out to empirically answer the following questions:

- **Does our system learn to encode sensorimotor events within its sparsely changing latent states?** We evaluate the semantics of the learned event encoding for scripted sequences (Sec. 6.3.2) and on more challenging data (Sec. 6.3.4).
- **Does the skip network learn to make meaningful temporal abstract predictions?** We showcase the quality of the skip predictions in Sec. 6.3.3.
- **Can our system model the development of goal-anticipatory gaze behavior in infants?** Finally, we demonstrate the emergence of goal-anticipatory behavior in Sec. 6.3.5.

^{6.4}In CAPRI, entropy was used for the uncertainty U . Computing the entropy of Gaussian distributions involves multiplying variances. When some variances are very close to zero, we empirically found the sum of variances to be more stable.

6.3.1 Scenario and Data

We apply our system in the **Fetch Pick&Place** environment (OpenAI Gym v1 [Brockman et al., 2016](#)), a benchmark reinforcement learning (RL) problem in which a robotic manipulator should move a randomly placed box on a table to a random goal position. The “hand” of the robot is a gripper with two fingers.^{6.5} The action $\mathbf{a}_t \in \mathbb{R}^4$ position-controls the movement of the hand and the opening of the gripper. In our experiments the observation $\mathbf{o}_t \in \mathbb{R}^{11}$ is composed of the three-dimensional position of three entities, i.e., the hand, the object, and the goal, and the fingers’ opening. We modify the simulation such that the height of the table can vary across simulations.

The system is trained and evaluated on datasets \mathcal{D} of 9.2k sensorimotor sequences $((\mathbf{o}_1, \mathbf{a}_1), \dots, (\mathbf{o}_T, \mathbf{a}_T)) \in \mathcal{D}$ of length $T = 25$. We consider two types of sensorimotor sequences, *scripted* and *generated via model-based RL*.^{6.6} The dataset $\mathcal{D}_{\text{script}}$ contains three types of scripted movements. In **reach-grasp-transport sequences** the hand moves to the object until it is located between the fingers. Then the gripper closes, and, once the object is fully grasped, the robot lifts the object to the goal position where it is held until the sequence is over. In **pointing sequences** the robot “points” to the goal by moving its hand to the goal position. In **stretching sequences** the robot stretches its arm by repeatedly performing the same randomly generated motor command. In all sequences, goal-directed motions follow a fixed direction and decrease their velocity when approaching the target to avoid overshooting. All actions were perturbed by normally distributed motor noise ($\sigma = 0.05$).

The $\mathcal{D}_{\text{APEX}}$ dataset contains sequences that were generated by the policy-guided model-predictive control method APEX ([Pinneri et al., 2021b](#)) trained to move the object to the goal position. APEX finds various ways to interact with the object, including lifting, pushing, or flicking it. In $\mathcal{D}_{\text{APEX}}$ table height is fixed.

When modeling infant gaze behavior (experiment [Sec. 6.3.5](#)), we provide an additional attention focus $\boldsymbol{\alpha}_t$. This corresponds to attending to one of the three entities, i.e., hand, object, or goal. When attending to one entity, sensory information about the other entities’ positions is masked by normally distributed noise ($\sigma = 0.05$). During training, the attentional focus is randomly shifted 5 times per sequence.

The forward-inverse model and the skip network were trained independently using Adam ($\text{lr}_\phi = 0.0005, \text{lr}_\theta = 0.0001$). Each experiment was carried out with 10 different random seeds. We provide further details of the experiment in [Suppl. D.1](#).

^{6.5}In the remainder of this chapter we will refer to the endeffector of the robot as its *hand*. We do this to unify notation and simplify the comparison to CAPRI.

^{6.6}For both types of datasets ($\mathcal{D}_{\text{script}}$, $\mathcal{D}_{\text{APEX}}$) we generate three version with 9.2k sequences, for training, for validation (hyperparameter search), and for testing.

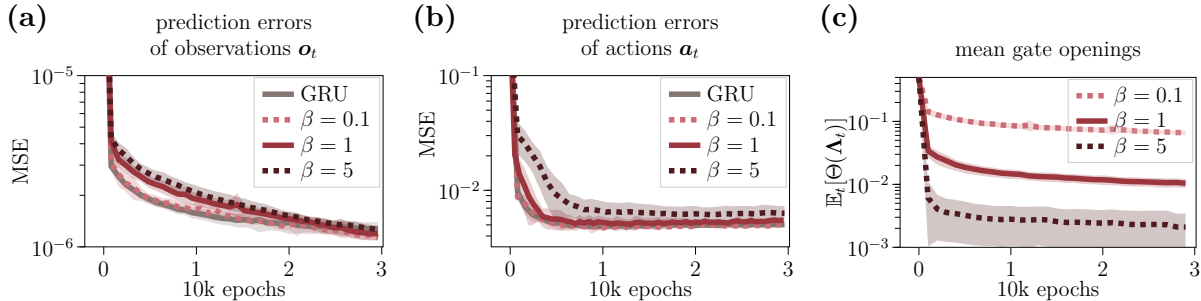


Figure 6.3: Prediction and gate regularization on $\mathcal{D}_{\text{script}}$: Test prediction error of observations \mathbf{o}_t (a) and actions \mathbf{a}_t (b) and mean number of gate openings (c), i.e., latent updates, for variants of our system with different regularization strengths β and a GRU ablation. Shaded areas show \pm one standard deviation.

6.3.2 Learned Event Segmentation

To analyze prediction accuracy and the learned latent state of the forward-inverse model, we first trained our model on the dataset $\mathcal{D}_{\text{script}}$ of scripted movements. We compare our system (with GATELORD) with different regularization strengths β to an ablated version without sparsity regularization that uses a GRU (Chung et al., 2014) as its internal RNN.

Fig. 6.3a and Fig. 6.3b show the prediction errors for the next observations and actions, respectively. The networks learn to improve their predictions over training. For a regularization of $\beta \leq 1$ the regularized versions reach the same prediction accuracy as the GRU ablation. Thus, the sparsity regularization via GATELORD, does not necessarily degrade performance. Figure 6.3c shows the mean number of gate openings, i.e., latent state changes. With a higher β , fewer latent states are updated. Our system with $\beta = 1$ learns to only adapt its latent state on average a handful of times during a sequence^{6,7}, with the same prediction abilities as less regularized variants. Thus, for further analysis we focus on our system with $\beta = 1$.

Figure 6.4 shows exemplary sequences for different interactions. The bottom row of each plot illustrates the latent states for one exemplary model. For the reach-grasp-transport sequence (Fig. 6.4a), the model changes its latent state when the hand has reached the object, when the object is fully grasped, and when hand and object arrive at the goal. Thus, the model seems to encode <reaching>, <grasping>, <transporting>, and <holding> the object using four unique latent states. For the pointing sequence (Fig. 6.4b), the model changes its latent state once the hand has reached the goal. Interestingly, the

^{6,7} $\mathbb{E}_t[\Theta(\mathbf{\Lambda}_t)] = 10^{-2}$ corresponds to changing one dimension of \mathbf{h}_t on average 4 times during a sequence (for $T = 25$ and 16-dimensional \mathbf{h}_t).

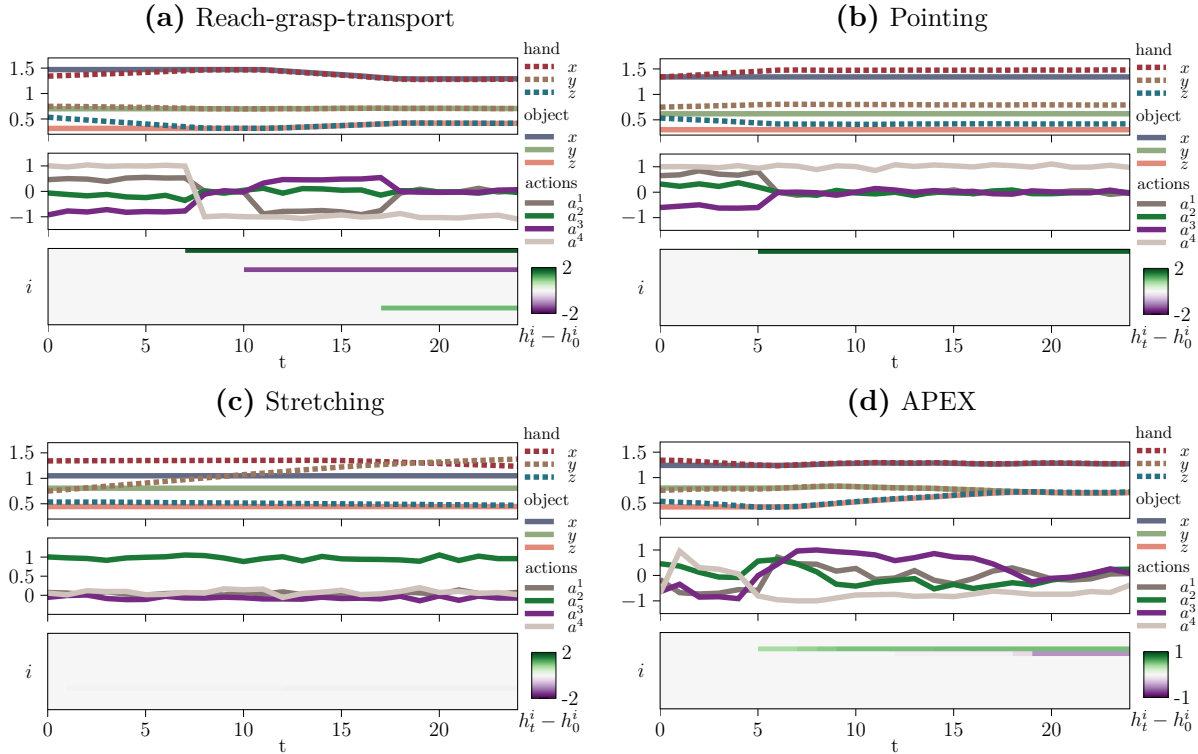


Figure 6.4: Exemplary sensorimotor sequences and latent states for reaching, grasping, and transporting an object **(a)**, pointing to the goal **(b)**, and stretching the arm **(c)** from $\mathcal{D}_{\text{script}}$, or an object interaction from $\mathcal{D}_{\text{APEX}}$ **(d)**. The upper row shows the positions of the hands and objects over time. The middle row shows actions ($a_t^{\{1,2,3\}}$ control hand movements, a_t^4 controls gripper closing). The bottom row shows the latent states relative to initialization, i.e. $\mathbf{h}_t - \mathbf{h}_0$. The same model was used for generating plots **(a)**-**(c)**, allowing us to compare how the same model encodes different events.

same dimension is updated when reaching the object (Fig. 6.4a) and the goal (Fig. 6.4b), indicating the similarity between the two events. For the stretching sequence (Fig. 6.4c), no clear latent update is visible. This implies that stretching is encoded as one event.

6.3.3 Skip Predictions

To systematically analyze the skip predictions, we fed scripted sequences into the forward-inverse model up to a fixed point in time at $t = 2$. We then used $(\mathbf{o}_2, \mathbf{h}_2)$ as inputs to the skip network to produce a skip prediction $\hat{\mathbf{o}}_{\tau(2)}$. Thus, we essentially queried the skip

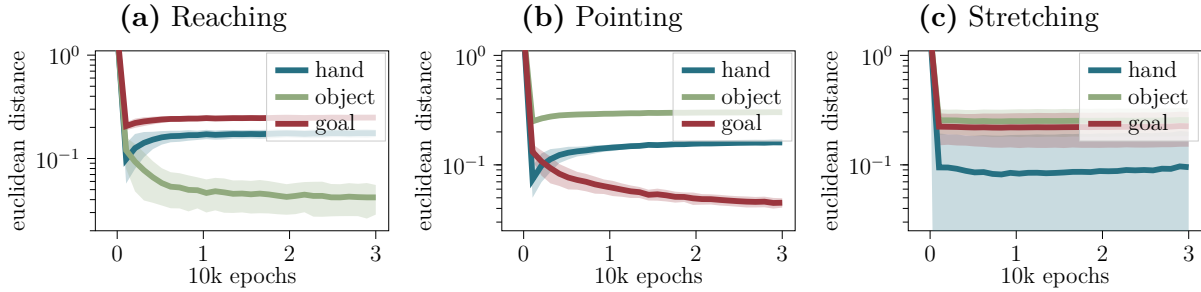


Figure 6.5: Skip predictions for $\mathcal{D}_{\text{script}}$: Euclidean distance between skip-predicted hand position in $\hat{\mathbf{o}}_{\tau(2)}$ to the position of all three entities in \mathbf{o}_2 for a skip at time $t = 2$. Shown for reach-grasp-transport (a), pointing (b), and stretching (c) sequences. Shaded areas denote standard deviation.

network at time $t = 2$ which observation are predicted at the next event boundary.

We compare the predicted position of the hand from $\hat{\mathbf{o}}_{\tau(2)}$ to the current position of the entities in \mathbf{o}_2 . Figure 6.5 shows the Euclidean distances between the hand position of the skip prediction and the current positions of all entities. For reaching sequences (Fig. 6.5a) the distance between the predicted hand position and the current object position decreases, while the other distances remain the same or increase. Thus, the skip network learns that the hand tends to move to the object, where the next gate opening, or event boundary, will occur. Similarly, for pointing sequences (Fig. 6.5b) the system learns that the hand will move toward the goal. For stretching sequences (Fig. 6.5c), on the other hand, the predicted distances between the hand and the other entities show high variances without any regularities.

Figure 6.6 shows the hand position predictions of all fully trained skip networks for three exemplary sequences. One blue circle marks the predicted position of the hand for one trained skip network. We show the predictions of all trained networks (10 random seeds) overlaid. For reach-grasp-transport sequences (Fig. 6.6a), the skip network predicts that the hand will be close to the object at the next event boundary. For pointing sequences (Fig. 6.6b), the skip network predicts that the hand will be at the goal location (red sphere). For both sequences, the predicted positions strongly overlap, implying a consistent segmentation and hierarchical prediction across different random initializations. For stretching sequences (Fig. 6.6c), the skip network predicts that the hand will be close to positions when the arm is fully stretched-out, or somewhere in-between the current position and the final position. We show skip predictions for other entities in Suppl. D.2.

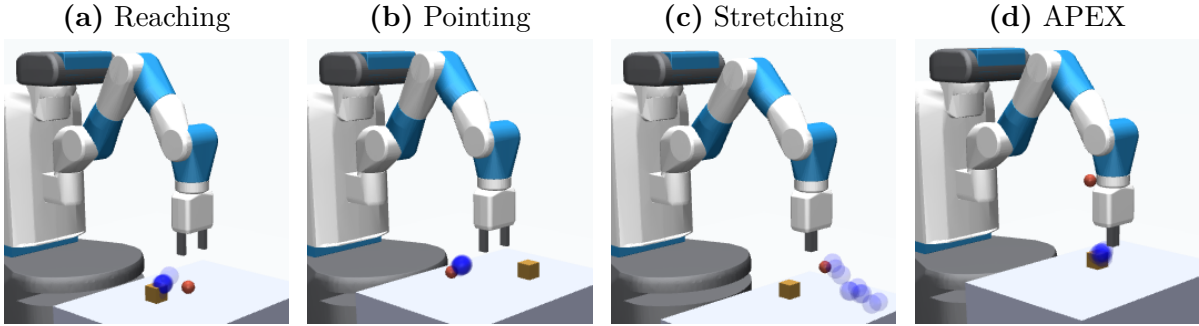


Figure 6.6: Exemplary event boundary predictions. Skip predictions of hand positions at $t = 2$ shown for reaching (a), pointing (b), and stretching (c) sequences of $\mathcal{D}_{\text{script}}$ and for reaching in $\mathcal{D}_{\text{APEX}}$ (d). One blue circle shows one skip-predicted hand position. Predictions of all 10 random seeds are shown together. The goal position is depicted as a red sphere.

6.3.4 Training on More Diverse Sequences

To investigate learning robustness, we trained our system on sequences of the $\mathcal{D}_{\text{APEX}}$ dataset, which contains various object interaction sequences. Again, we compare our system (with GATELORD, $\beta \in \{1, 5, 10\}$) to a GRU ablation.

Fig. 6.7a and Fig. 6.7b show the prediction errors for the observations and actions, respectively. Our system and the unregularized GRU ablation reach a similar level of accuracy for predicting observation. However, the GRU ablation learns to better predict actions. It seems that for predicting the unregularized outputs of a policy, strong regularization of the latent state updates is detrimental. For further evaluations, we focus on our system with $\beta = 5$.

Figure 6.4d shows an exemplary reach-grasp-and-transport sequence from $\mathcal{D}_{\text{APEX}}$. The latent states, depicted in the bottom row, change strongly at two points in the sequence. First, when the object is grasped, and later, when the object was successfully transported to the goal. Thus, the latent states seem to encode a similar event structure as for the scripted data (cf. Fig. 6.4a), even for the much noisier actions of $\mathcal{D}_{\text{APEX}}$.

To evaluate the system’s ability to make temporal abstract predictions, we input $(\mathbf{o}_2, \mathbf{h}_2)$ into the skip network and analyze the resulting predictions $\hat{\mathbf{o}}_{\tau(2)}$. Figure 6.7c shows the Euclidean distance between the skip-predicted hand position in $\hat{\mathbf{o}}_{\tau(2)}$ and the current positions of all entities at the time $t = 2$. During the course of training, the distance between the predicted hand position and the current object position decreases. Thus, with increasing experience the skip network learns that the hand will move to the object. Additionally, the increasing distance between the current and predicted hand position

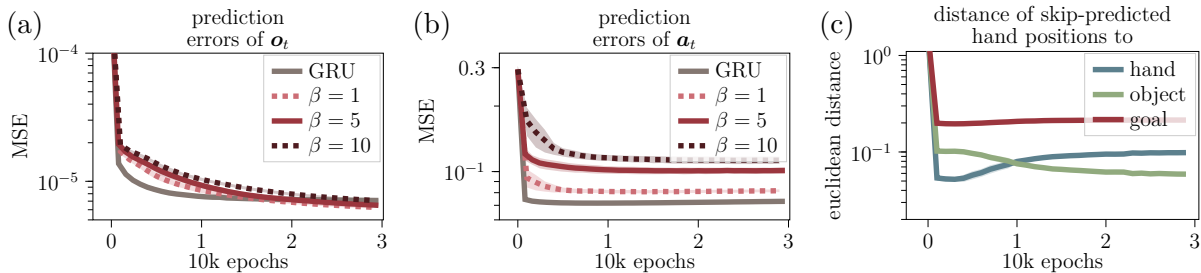


Figure 6.7: Prediction errors and skip predictions on $\mathcal{D}_{\text{APEX}}$: Test prediction error of observations (a) and actions (b) for variants of our system with different regularization strengths β and a GRU ablation. Euclidean distance between skip-predicted hand position to the current position of all three entities for a skip at time $t = 2$ (c).

suggests that with more training experience, skip predictions reach farther into the future. Figure 6.6d shows the skip-predicted hand positions for one exemplary sequence. Across all random seeds, the system reliably learns to predict that the hand will be close to the object position at the next event boundary.

6.3.5 Modeling Infants’ Goal Anticipations

Finally, we model the experimental findings of goal anticipations in infants. We train our system on $\mathcal{D}_{\text{script}}$ with an additional gaze focus α_t . We test the system similarly to eye-tracking studies on infant goal anticipation, e.g. Adam & Elsner (2020). We let the system “observe” sequences. During observation, only \mathbf{o}_t is available and the action inputs are generated by the inverse model f_ϕ^{IM} .^{6.8} We let the system choose its gaze α_t to minimize uncertainty about future positions of the hand (specified as indices I in Eq. 6.16). Similar to how developmental studies track the first gaze to a goal area, we track the time step t^ϵ , when the system first attended to entity ϵ (hand, object, or goal)^{6.9}. Akin to infants’ goal-anticipatory gaze shifts, we want to see if our system attends to the goal of an event, e.g., the to-be-grasped object, before the goal is reached.

We compare three different versions of attention selection for reaching events: Minimizing uncertainty within the current event (event dynamics uncertainty in Eq. 6.15), minimizing uncertainty about the next event boundary (event boundary uncertainty in Eq. 6.15), or minimizing both uncertainties (full Eq. 6.15). When minimizing uncertainty about the event dynamics for reaching (Fig. 6.8a), the system first attended to the hand (blue) long before hand-object-contact (dashed black line) with little variance

^{6.8}Only the first action \mathbf{a}_1 is provided to initialize the latent state \mathbf{h}_0 and start the process.

^{6.9}For each entity ϵ , we determine the first time step t^ϵ for which $\alpha_{t^\epsilon}^\epsilon = 1$, with α_t the one-hot encoded gaze focus. No focus on an entity is treated as $t^\epsilon = 25$, i.e., maximum length T .

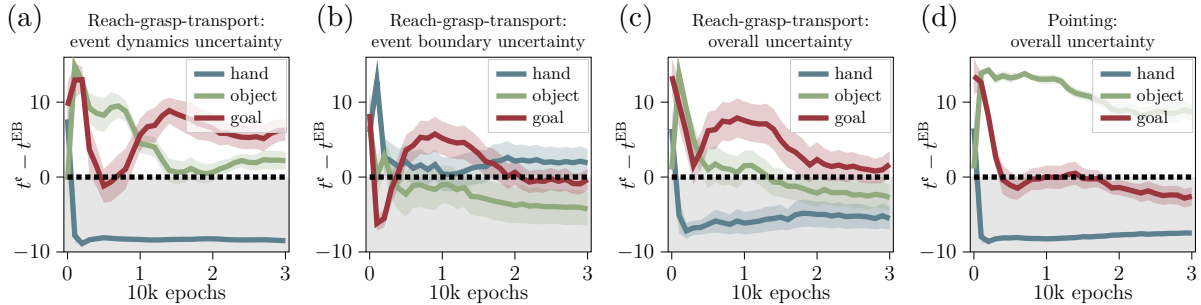


Figure 6.8: Goal-anticipatory gaze of our system: The y -axis shows the time t^ϵ of first attending to entity ϵ relative to the time t^{EB} of the first event boundary of a sequence, i.e. $t^\epsilon - t^{\text{EB}}$. Thus, negative values (gray area) for an entity indicate that the system attended to this entity before the first event was over. t^{EB} is marked as dashed black lines (reaching: hand at object; pointing: hand at goal). For reaching events, we evaluate gaze inference for minimizing uncertainty about event dynamics (a), about the next event boundary (b), or both (c). For pointing sequences, we only consider minimizing both uncertainties (d). Shaded areas show standard error.

across simulations. Naturally, attending to the hand helps to predict immediate hand movements during reaching. The system on average attended to other entities only after hand-object-contact with higher variance across simulations.

When minimizing uncertainty about the event boundary (Fig. 6.8b), the system on average first attended to the object (green) before it was reached by the hand (dashed black line). Apparently, the system has learned that attending to the object helps to minimize uncertainty about the end of reaching.

When putting both uncertainties together, the system exhibits goal-anticipatory gaze shifts: For reach-grasp-transport sequences (Fig. 6.8c), the system on average first attended to the hand (blue), followed by attending to the object (green). The focus on the object happened on average before hand-object-contact (dashed black line). Similarly, for pointing sequences (Fig. 6.8d) the system on average first attended to the hand (blue) and then to the goal (red) before the hand arrived there (dashed line). Thus, in both cases, the system shifted its gaze to the goal of the event before the event was over. Crucially, this behavior developed with training

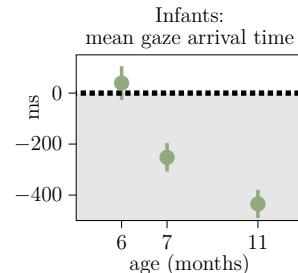


Figure 6.9: Goal-anticipatory gaze in infants from Adam & Elsner (2020). The dashed line depicts hand-object-contact. The gray area marks anticipatory gaze. Error bars show standard error.

experience.

This behavior mimics gaze behavior found in infants: Figure 6.9 shows the mean gaze arrival time for infants watching movies of a hand grasping and lifting a toy (Adam & Elsner, 2020) depending on age. The movies started with a still frame depicting a toy (0-1000ms), followed by a hand entering the screen and moving to the object (1000-3500 ms), lifting the toy and placing it back (3500-6300 ms). Mean gaze arrival times were calculated by subtracting the time when the hand entered the target object area of interest (AOI), from the time of the first fixation to that AOI. Negative gaze arrival times thus indicate goal-anticipatory gaze. Only trials in which the hand was first fixated (for a minimum of 200 ms) were considered. As shown in Fig. 6.9, 6-month-olds tend to follow the hand with their gaze. Older infants (7 and 11 months) shift their gaze from the hand to the object before it is reached. These results are qualitatively matched by our systems’ behavior (cf. Fig. 6.8c, green line).

6.4 Discussion

We introduced a hierarchical deep learning system that learns to encode its sensorimotor experience in event-compressing, latent states, which are used for probabilistic forward- and inverse predictions. We show that the learned codes can uncover suitable segmentations of processed sensorimotor sequences. Based on our general modeling approach of how nested temporal hierarchies of events may develop (THICK MDPs, Chap. 3), we trained a high-level model to only predict situations prompting latent state changes. We show that this hierarchical learning scheme enables the learning of meaningful temporal abstract predictions of event boundaries from scratch without explicit supervision. Thus, the system can be seen as the first proof-of-concept for deep learning THICK MDPs.

We tested our model similarly to how goal anticipation was studied in infants (Kanakogi & Itakura, 2011; Adam & Elsner, 2020). Infants learn to shift their gaze from a moving entity (hand) to the next goal of an event as they mature. In our system this developed with training time and better segmentations. Thus, hypotheses that event familiarity and motor experience enable goal-anticipatory gaze shifts in infants (Kanakogi & Itakura, 2011; Elsner & Adam, 2021) align with our model.

Previously, CAPRI used an explicit structure of events and their boundaries to model the development of goal-anticipatory gaze in infants. Our system follows the modeling approach of CAPRI but advanced several aspects: (1.) In CAPRI sequence segmentation was supervised. Our system learns event segmentation in a completely self-supervised fashion purely from prediction and latent state regularization. This allowed us to apply uncertainty-minimizing gaze inference strategies without providing any event information explicitly. (2.) In CAPRI event models were restricted to fully observable linear dynamics.

Our novel architecture is able to deal with more realistic scenarios, where motions are boundedly complex but non-linear and information can be hidden. (3.) CAPRI ignored the motor commands involved in the production of an event. Our new model is able to simulate motor commands through its inverse model while observing a motion.

Apart from general hyperparameters related to neural networks, our model only has one hyperparameter that needs to be set: the regularization strength β , controlling the trade-off between prediction accuracy and sparsity in latent state changes. The parameter is important for developing robust event segmentation and thus the emergence of meaningful goal-predictive attention shifts. While GATELORD works rather robustly under a range of values for β , in the future it may be interesting to investigate a battery of gates with different values of β and automatically determine the best-suited value.

Seeing that our model learns event segmentation best when there is clear sensorimotor information available to separate events, our model predicts that salient segmentation cues, e.g., a sound at the event boundary, should ease event segmentation, and thus boost goal anticipations. Future infant studies could test whether goal anticipation for newly learned event sequences occurs at an earlier age if during learning of sequences salient segmentation cues are provided.

Our system hierarchically selects its gaze behavior while passively observing event sequences. Could the same system be used to hierarchically plan motor actions, e.g. grasping or lifting an objects, while interacting with its environment? Unfortunately, in the current state, one major restriction prevents universally applying the system for hierarchical model-based planning: The high level only predicts the most likely next event boundary. It cannot predict fundamentally different outcomes for the same situation.^{6.10} For example, it would predict either moving the hand to an object or moving the hand to the goal, depending on whether it judges reaching or pointing to be the likely event. For goal-directed planning, the high level would either need to predict multimodal distributions that encode multiple possible event boundaries or learn a form of high-level action such that it can distinguish between different next event boundaries. In the next chapter, we will overcome this limitation and present a fully hierarchical generative world model, where event boundaries are encoded as categorical distributions and action abstractions enable planning on multiple time scales.

^{6.10}The system is able to predict how gaze behavior affects the distribution for the next event boundary. However, the distribution is unimodal and typically encodes one outcome, such as a grasp, and not different event boundaries.

7

Hierarchical World Models^{7.1}

Hierarchical predictions can substantially improve model-based reinforcement learning (MBRL) and planning by enabling reasoning across multiple time scales. In this chapter, we combine, enhance, and scale our previously developed methods to learn a hierarchy of world models from scratch according to the principles of THICK. Our THICK world models maintain sparsely changing latent states at the low level and learn to predict situations prompting such latent state changes at its high level. Crucially, the high-level model develops categorical high-level action abstraction in a self-supervised way to disambiguate different temporal abstract outcomes. Based on these high-level actions the system can simulate and plan behavior on an adaptive event-based time scale. We show that the emerging hierarchical model seamlessly enhances the abilities of MBRL and planning methods in challenging problems with pixel-based inputs.

^{7.1}This chapter is based on the following manuscript, accepted for publication:

Gumbsch, C., Sajid, N., Martius, G. & Butz, M. V. (in press, 2024). Learning Hierarchical World Models with Adaptive Temporal Abstractions from Discrete Latent Dynamics. *The Twelfth International Conference on Learning Representations (ICLR 2024)*

The text has been slightly revised to avoid repetitions and to fit better within this thesis. The figures have been adopted unchanged from the manuscript. In accordance with the manuscript and to recognize the contributions of my coauthors, this chapter is written in the first person plural (we).

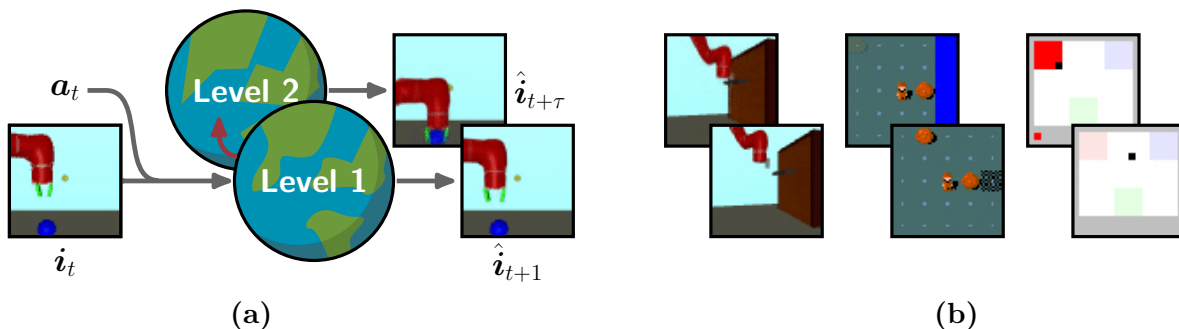


Figure 7.1: THICK world models predict on two levels. (a) Level 1 predicts the next input ($t + 1$). Level 2 predicts a future state ($t + \tau$) expected to change an otherwise constant latent state. (b) Exemplary low- (bottom) and high-level predictions (top) for opening a door (Multiworld-Door), pushing a boulder into water (Minihack-River), or activating a pad (VisualPinPadThree).

7.1 Introduction

There is a conglomerate of evidence that various representations formed by the human brain to encode their sensorimotor experience are hierarchically structured (Lee & Mumford, 2003; Rougier et al., 2005; Botvinick & Weinstein, 2014; Rohe & Noppeney, 2015; Lake et al., 2017; Friston et al., 2018, 2021; Butz, 2016; Tomov et al., 2020). These hierarchical abstractions seem to allow predictions and planning on different time scales and levels of abstraction. Humans, for example, can plan their behavior on various time scales and flexibly switch between them. For example, when organizing a birthday party, a temporally distant event planned on a rather high-level of abstraction, one might pick up a pen to write an invitation, a short-horizon, low-level sensorimotor event. The integration of hierarchical models in model-based reinforcement learning (MBRL) and model-predictive control (MPC) could bolster planning efficiency and effectiveness in complex, long-horizon, real-world tasks (Schmidhuber, 1992; LeCun, 2022).

Recent research has made tremendous advancements in equipping MBRL agents with the capacity to learn world models, i.e., generative forward models that encode an agent’s interaction with its environment from high-dimensional inputs (Ha & Schmidhuber, 2018; Hafner et al., 2019b,a, 2020a, 2023). However, these models lack a hierarchical structure. Consequently, they are restricted to predictions on predefined time scales, hampering their capability for long-horizon planning. The main challenge lies in formalizing suitable methods to learn higher-level abstractions (Sutton et al., 1999b; Pateria et al., 2021; Precup, 2000; van Seijen et al., 2014). Fixed hierarchical time scales are inadequate since different events, and even instances of the same event, typically have different durations.

Conversely, learning temporal abstractions from rewards like in hierarchical RL (Sutton et al., 1999b; Pateria et al., 2021) binds the abstractions to a specific task, whereas world models ideally should maintain a degree of task-agnosticism (Friston et al., 2021).

Throughout this thesis, I have argued for and developed the necessary tools for a novel approach to learn hierarchical world models. This approach is called **Temporal Hierarchies from Invariant Context Kernels (THICK)**. Motivated by theories of event cognition (Sec. 2.1) and our prior work on sparsely changing latent states (Chap. 5), THICK assumes that temporally persistent contexts or events can be discovered by guiding a lower-level world model to update parts of its latent state only sparsely in time. The high-level model is then trained to predict scenarios involving changes in these low-level latent states (as in Chap. 6). Driven by the objective of disentangling temporal abstract outcomes, the THICK world models additionally develop high-level action abstraction in a self-supervised way. Thus, we distill a high-level world model from discrete low-level latent state updates. A depiction of THICK world models can be found in Fig. 7.1a.

In this chapter make the following key contributions:

- We introduce the Context-specific Recurrent State Space Model (C-RSSM), which enhances Dreamer’s (Hafner et al., 2019b, 2020a) Recurrent State Space Model (RSSM) by **encoding context-sensitive dynamics** via sparsely changing latent factors, labeled *context*. Thus, we scale our previous models to complex problems with high-dimensional inputs.
- We provide an implementation of THICK, an algorithm that learns a **hierarchy of world models**. The high-level runs at an adaptive time scale developing categorical representation of **higher-level actions** that anticipate lower-level context changes.
- We demonstrate the effectiveness of THICK in two **planning** scenarios: (1.) using THICK’s hierarchical predictions to enhance **MBRL in long-horizon tasks**, and (2.) using THICK’s high-level predictions to set sub-goals for **hierarchical model-predictive planning** (MPC).

7.2 THICK World Models

THICK world models are composed of a low-level world model that predicts next observations at the time scale of an underlying POMDP, and a high-level world model that predicts on an abstract, event-based time scale. First, we detail the low-level world model (Sec. 7.2.1), before outlining the design and training scheme of the high-level world model (Sec. 7.2.2).

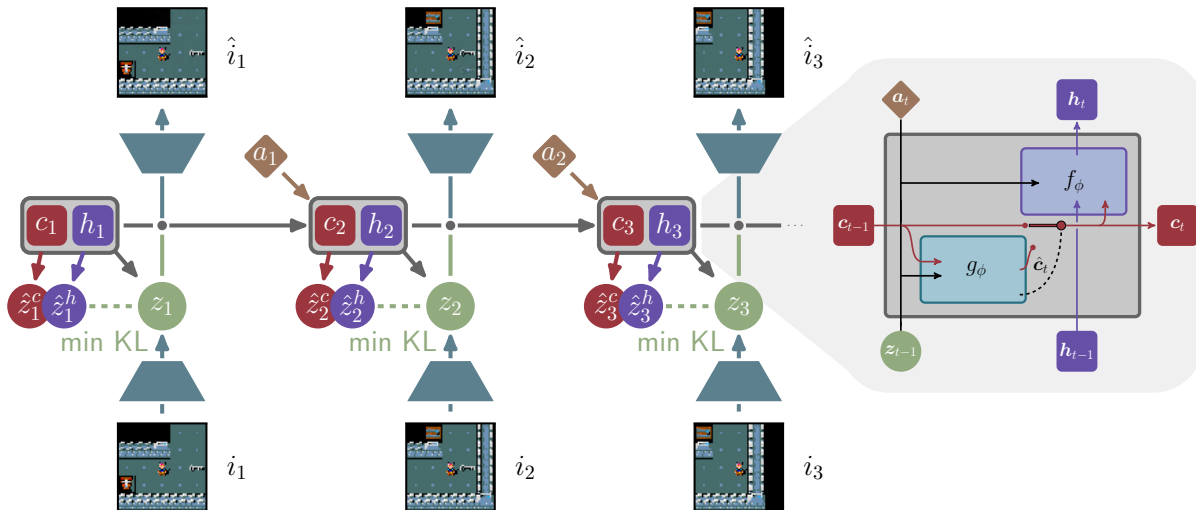


Figure 7.2: C-RSSM world model. Left: The C-RSSM encodes dynamics within latent states with a stochastic part z_t , a high-dimensional deterministic part h_t , and a low-dimensional, sparsely changing context c_t . The network predicts the next stochastic state z_t via two pathways: It makes *coarse* predictions \hat{z}_t^c based mainly on c_t and *precise* predictions \hat{z}_t^h based on h_t . The posterior z_t is updated given also the new input image i_t . Right: Internally, the sparsely changing context c_t is updated through a GATELORD cell g_ϕ with a sparsely operated update gate. A GRU cell f_ϕ is used to continuously change h_t .

7.2.1 C-RSSM World Model

The RSSM proposed in Hafner et al. (2019b) is a recurrent neural network (RNN) that is commonly used for model-predictive control (Hafner et al., 2019b; Sajid et al., 2021b) or model-based reinforcement learning (Hafner et al., 2019a, 2020a, 2023; Sekar et al., 2020; Mendonca et al., 2021). The RSSM embeds sequences of input images i_t and actions a_t into a latent state s_t and predicts dynamics exclusively within this latent state. All aspects of the latent state evolve continuously. We require sparse latent state changes to establish hierarchical world models. We propose **Context-specific RSSM (C-RSSM)**, a modification of the RSSM that (1.) integrates a sparsely changing latent state c_t as context and (2.) adds a coarse prediction pathway. Figure 7.2 illustrates the C-RSSM. Our C-RSSM world model w_ϕ with trainable parameters ϕ is computed by:

$$\mathbf{s}_t \leftarrow [\mathbf{c}_t, \mathbf{h}_t, \mathbf{z}_t] \quad (\text{full latent state}) \quad (7.1)$$

$$\mathbf{c}_t = g_\phi(\mathbf{a}_{t-1}, \mathbf{c}_{t-1}, \mathbf{z}_{t-1}) \quad (\text{coarse dynamics}) \quad (7.2)$$

$$\mathbf{h}_t = f_\phi(\mathbf{a}_{t-1}, \mathbf{c}_t, \mathbf{h}_{t-1}, \mathbf{z}_{t-1}) \quad (\text{precise dynamics}) \quad (7.3)$$

$$\hat{\mathbf{z}}_t^c \sim p_\phi^c(\hat{\mathbf{z}}_t^c \mid \mathbf{a}_{t-1}, \mathbf{c}_t, \mathbf{z}_{t-1}) \quad (\text{coarse prior}) \quad (7.4)$$

$$\hat{\mathbf{z}}_t^h \sim p_\phi^h(\hat{\mathbf{z}}_t^h \mid \mathbf{c}_t, \mathbf{h}_t) \quad (\text{precise prior}) \quad (7.5)$$

$$\mathbf{z}_t \sim q_\phi(\mathbf{z}_t \mid \mathbf{c}_t, \mathbf{h}_t, \mathbf{i}_t) \quad (\text{posterior}) \quad (7.6)$$

Equations in red are exclusive to C-RSSM.^{7.2} We separate the RSSM’s latent state \mathbf{s}_t into three parts (Eq. 7.1): a stochastic state \mathbf{z}_t , a continuously updated, high-dimensional, deterministic state \mathbf{h}_t , and a sparsely changing, low-dimensional context \mathbf{c}_t . At time t the C-RSSM first updates the context \mathbf{c}_t (Eq. 7.2). The coarse dynamics g_ϕ is implemented using a GATELORD cell (see Chap. 5), such that actual \mathbf{c}_t changes only occur *sparsely in time*.^{7.3} Next, C-RSSM updates \mathbf{h}_t through a GRU (Chung et al., 2014) cell f_ϕ (Eq. 7.3). The C-RSSM makes two predictions about the next stochastic state $\hat{\mathbf{z}}_t^h$: (1.) a *precise* prior based on both \mathbf{h}_t and \mathbf{c}_t (Eq. 7.5), and (2.) a *coarse* prior $\hat{\mathbf{z}}_t^c$ based on the context, stochastic state, and action, ignoring \mathbf{h}_t (Eq. 7.4). Given the input image \mathbf{i}_t , C-RSSM updates its posterior \mathbf{z}_t (Eq. 7.6). Following DreamerV2 (Hafner et al., 2020a), we sample \mathbf{z}_t from a vector of categorical distributions. Thus, \mathbf{z}_t is a vector of one-hot encoded values.

Importantly, Eq. 7.2 and Eq. 7.4 do not depend on \mathbf{h}_{t-1} . This creates a **coarse processing pathway** independent of \mathbf{h} . This allows predictions using only \mathbf{c}_t as a deterministic memory, which is crucial because it (1.) encourages encoding prediction-relevant information in \mathbf{c}_t and (2.) allows predictions without \mathbf{h}_t which we will use later.^{7.4}

In addition to encoding latent dynamics, C-RSSM is trained to reconstruct observable variables y_t of the outside world from its latent states \mathbf{s}_t . Two output heads o_ϕ generate precise and coarse predictions:

$$\hat{y}_t \sim o_\phi(\hat{y}_t \mid \mathbf{s}_t) \quad (\text{precise prediction}) \quad (7.7)$$

$$\hat{y}_t^c \sim o_\phi^c(\hat{y}_t^c \mid \mathbf{c}_t, \mathbf{z}_t) \quad (\text{coarse prediction}) \quad (7.8)$$

for some aspect y_t of the world that we want to predict. More specifically, we predict the input image \mathbf{i}_t , the reward r_t , and reward discount γ_t ^{7.5}, i.e., $y_t \in \{\mathbf{i}_t, r_t, \gamma_t\}$.

^{7.2}Removing \mathbf{c} in all black equations recovers the equations for the RSSM (Eqns. 7.1,7.3,7.5,7.6).

^{7.3}In previous chapters, we referred to all RNN latent states, including the ones of GATELORD, as \mathbf{h}_t . Since the C-RSSM maintains a GRU latent state as well as the latent state of GATELORD, we refer to the latter now as \mathbf{c}_t to distinguish the two.

^{7.4}We provide a more detailed explanation of our design choices in Suppl. E.3.1

^{7.5}The discount γ_t is set to 0 if an episode terminates and to a fixed value γ otherwise.

Sparse context updates: The latent context code \mathbf{c}_t is designed to change sparsely in time, ideally at distinct transition points, specific to the environment. Consequently, the coarse dynamics g_ϕ (Eq. 7.2) are modeled by a GATELORD cell (see Chap. 5), which learns sparsely changing latent states \mathbf{c}_t through an update gate, whose activity is L_0 -regularized via loss term $\mathcal{L}^{\text{sparse}}$.

Loss function: Given a sequence of length T of input images $\mathbf{i}_{1:T}$, actions $\mathbf{a}_{1:t}$, rewards $r_{1:T}$, with discounts $\gamma_{1:T}$, the parameters ϕ of C-RSSM are jointly optimized to minimize the loss $\mathcal{L}(\phi)$:

$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi} [\beta^{\text{pred}} \mathcal{L}^{\text{pred}}(\phi) + \beta^{\text{KL}} \mathcal{L}^{\text{KL}}(\phi) + \beta^{\text{sparse}} \mathcal{L}^{\text{sparse}}(\phi)], \quad (7.9)$$

including the prediction loss $\mathcal{L}^{\text{pred}}$, the KL loss \mathcal{L}^{KL} , and sparsity loss $\mathcal{L}^{\text{sparse}}$ with respective hyperparameters β^{pred} , β^{KL} , and β^{sparse} scaling their impact. The prediction loss $\mathcal{L}^{\text{pred}}$ drives the system to accurately predict perceptions y via its output heads o_ϕ , including context-conditioned coarse predictions (Eq. 7.8). The KL loss \mathcal{L}^{KL} minimizes the KL divergences between prior predictions p_ϕ^h and p_ϕ^c and the approximate posterior q_ϕ . The sparsity loss $\mathcal{L}^{\text{sparse}}$ encourages consistency of context \mathbf{c}_t . The exact loss functions are provided in Suppl. E.3.3. We set β^{pred} and β^{KL} to DreamerV2 defaults (Hafner et al., 2020a) and only modify the sparsity loss scale β^{sparse} depending on the scenario (cf. Suppl. E.2).

In summary, we have modified the RSSM to maintain a sparsely changing context \mathbf{c}_t . As we will see later, C-RSSM learns to update \mathbf{c}_t at discrete points in time to memorize unobservable information in order to better reconstruct the present and predict the future. Additionally, we have equipped the system with two potential prediction pathways: It can make detailed predictions about the future using the full latent state \mathbf{s}_t . In addition to that, it can also make coarse predictions using mainly the context \mathbf{c}_t and stochastic states \mathbf{z}_t . The coarse predictions will be less accurate whenever unobservable information is encoded in \mathbf{h}_t and not in \mathbf{c}_t , e.g. when there exist continuously changing latent factors in the environment. We will now use both new features, i.e. sparsely changing contexts and the coarse prediction pathway, to learn a hierarchy of world models.

7.2.2 Hierarchical World Model

To learn a hierarchical world model, we leverage C-RSSM’s discrete context \mathbf{c}_t updates by means of our **Temporal Hierarchies from Invariant Context Kernels (THICK)** algorithm. A C-RSSM world model w_ϕ segments sequences into periods of stable context activity ($\mathbf{c}_t = \mathbf{c}_{t+1} = \dots = \mathbf{c}_{\tau-1}$), interspersed with sparse context updates (cf. Fig. 7.3a). THICK uses these discrete context dynamics as an **adaptive timescale** for training a high-level network W_θ . The core assumption is that the states that prompt context

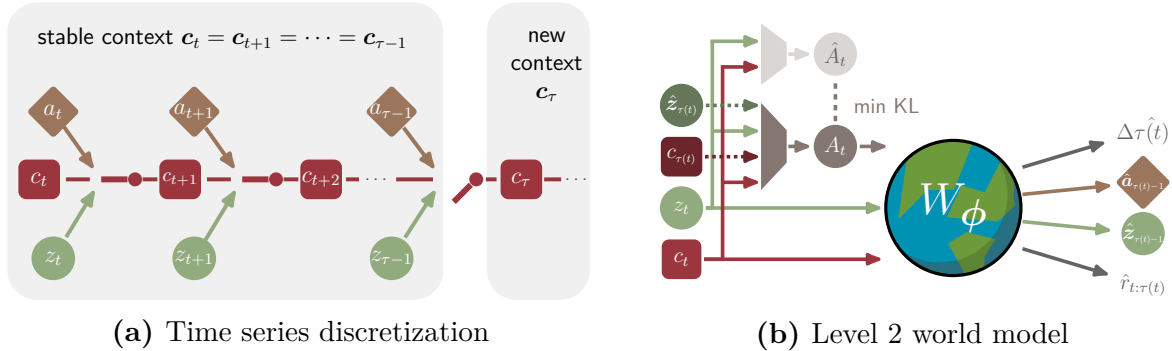


Figure 7.3: High-level segmentation: (a) The low-level C-RSSM discretizes sequences into segments with constant contexts. We use this segmentation to determine inputs and targets for the high level. (b) The high-level world model predicts the states and actions that lead to a context change at time $\tau(t)$ from latent states \mathbf{z}_t and \mathbf{c}_t . High-level actions (\mathbf{A}_t or $\hat{\mathbf{A}}_t$) distinguish high-level outcomes.

updates coincide with crucial changes in latent generative factors. These key states are predicted by the high-level network W_{θ} , while the states between context updates are ignored.

To train the high-level world model W_{θ} , we require input-target pairs for a given sequence of T images $\mathbf{i}_{1:T}$, actions $\mathbf{a}_{1:T}$, and episode termination flags $d_{1:T}$. The sequence is passed through the low-level model w_{ϕ} to obtain a sequence of contexts $\mathbf{c}_{1:T}$. The targets are defined as all time steps τ with context changes, i.e., where $\mathbf{c}_{\tau} \neq \mathbf{c}_{\tau-1}$ or the episode ends. As in previous chapters, we define the function $\tau(\cdot)$ as

$$\tau(t) = \min(\{\tau \mid \tau > t \wedge (\mathbf{c}_{\tau} \neq \mathbf{c}_{\tau-1} \vee d_{\tau} = 1)\}). \quad (7.10)$$

Thus, $\tau(\cdot)$ maps every point t to the next point in time $\tau(t)$ with context change, effectively implementing a variable temporal abstraction that generates target predictions $\tau(t)$ for every t .^{7.6}

High-level targets: We predict all variables at $\tau(t)$ that may cause a context change or are needed for planning across a context change: $\hat{\mathbf{z}}_{\tau(t)-1}$, $\hat{\mathbf{a}}_{\tau(t)-1}$, $\Delta \hat{\tau}(t)$, $\hat{r}_{t:\tau(t)}^{\gamma}$ (cf. Fig. 7.3b). In particular, we predict the stochastic states $\hat{\mathbf{z}}_{\tau(t)-1}$ and actions $\hat{\mathbf{a}}_{\tau(t)-1}$ immediately before a context change at time $\tau(t)$, because both can cause an update of $\mathbf{c}_{\tau(t)}$ (see Eq. 7.2). Intuitively, this means that observations, e.g. seeing something fall, as well as actions, e.g. catching something, could contribute to a change of \mathbf{c}_t . We furthermore

^{7.6}Note this is essentially Eq. 3.1 from THICK MDPs or Eq. 6.12 of the skip network but tailored to the C-RSSM.

predict the elapsed time $\Delta\tau(t)$ and the accumulated discounted reward $r_{t:\tau(t)}^\gamma$, which may account for variable duration and rewards when evaluating high-level outcomes:

$$\Delta\tau(t) = \tau(t) - t \quad (\text{elapsed time}) \quad (7.11)$$

$$r_{t:\tau(t)}^\gamma = \sum_{\delta=0}^{\Delta\tau(t)-1} \gamma^\delta r_{t+\delta} \quad (\text{accumulated rewards}) \quad (7.12)$$

High-level inputs: To predict high-level targets, we use the low-level stochastic state \mathbf{z}_t and context \mathbf{c}_t as inputs. However, we need to disambiguate different potential outcomes, which generally depend on the world and the policy pursued by the agent. Consequently, similar to actions at the low level, we create self-organizing high-level ‘‘actions’’ \mathbf{A}_t , similar to skills or options (Sutton et al., 1999b). \mathbf{A}_t encode a categorical distribution over probable next context changes. To learn \mathbf{A}_t , the high-level world model implements a *posterior* action encoder Q_θ and a *prior* action encoder P_θ (cf. Fig. 7.3b). In summary, the high-level world model W_θ with learnable parameters θ is computed by:

$$\mathbf{A}_t \sim Q_\theta(\mathbf{A}_t \mid \mathbf{c}_t, \mathbf{z}_t, \mathbf{c}_{\tau(t)}, \mathbf{z}_{\tau(t)}) \quad (\text{posterior high-level action}) \quad (7.13)$$

$$\hat{\mathbf{A}}_t \sim P_\theta(\hat{\mathbf{A}}_t \mid \mathbf{c}_t, \mathbf{z}_t) \quad (\text{prior high-level action}) \quad (7.14)$$

$$\hat{\mathbf{z}}_{\tau(t)-1} \sim F_\theta^{\hat{\mathbf{z}}}(\hat{\mathbf{z}}_{\tau(t)-1} \mid \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \quad (\text{high-level state prediction}) \quad (7.15)$$

$$\hat{\mathbf{a}}_{\tau(t)-1} \sim F_\theta^{\hat{\mathbf{a}}}(\hat{\mathbf{a}}_{\tau(t)-1} \mid \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \quad (\text{high-level action prediction}) \quad (7.16)$$

$$\Delta\hat{\tau}(t) \sim F_\theta^{\hat{\tau}}(\Delta\hat{\tau}(t) \mid \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \quad (\text{high-level time prediction}) \quad (7.17)$$

$$\hat{r}_{t:\tau(t)}^\gamma \sim F_\theta^{\hat{r}}(\hat{r}_{t:\tau(t)}^\gamma \mid \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \quad (\text{high-level reward prediction}) \quad (7.18)$$

The posterior Q_θ receives not only \mathbf{c}_t and \mathbf{z}_t as its input but also privileged information about the actually encountered next context, i.e. $\mathbf{c}_{\tau(t)}$ and $\mathbf{z}_{\tau(t)}$ (Eq. 7.13), which leads to the emergence of individualized, result-conditioned action encodings in \mathbf{A}_t . The prior P_θ learns a distribution over $\hat{\mathbf{A}}_t$ approximating the posterior without privileged information about the future (Eq. 7.14). During training, THICK samples the high-level action \mathbf{A}_t from Q_θ . During evaluation, we obviously cannot know the future states, so we sample from the prior P_θ instead. We model $\hat{\mathbf{A}}_t$ and \mathbf{A}_t as one-hot encoded categorical variables.

Loss function: The high-level world model W_θ with learnable parameters θ is trained to minimize the loss

$$\mathfrak{L}(\theta) = \mathbb{E}[\zeta^{\text{pred}} \mathfrak{L}^{\text{pred}}(\theta) + \zeta^{\text{KL}} \mathfrak{L}^{\text{KL}}(\theta)], \quad (7.19)$$

with hyperparameters ζ^{pred} and ζ^{KL} scaling the prediction $\mathfrak{L}^{\text{pred}}$ and action \mathfrak{L}^{KL} loss terms, respectively. The prediction loss is the summed negative log-likelihood of the high-level

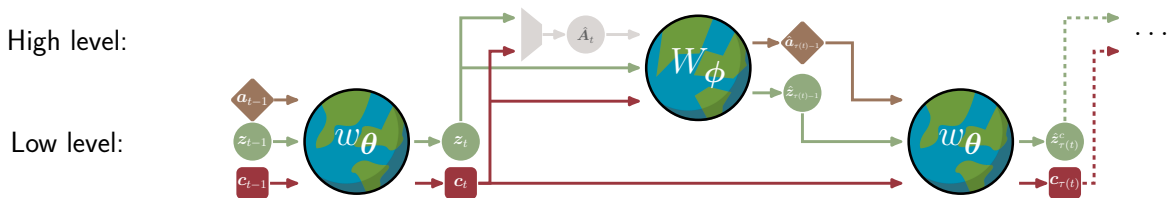


Figure 7.4: Temporal abstract predictions. From a low-level context c_t and stochastic state z_t , the high level predicts a future stochastic state $\hat{z}_{\tau(t)-1}$ as well as the action $\hat{a}_{\tau(t)-1}$. With these predictions, the context $c_{\tau(t)}$ is updated at the low level together with the coarse prior $\hat{z}_{\tau(t)}^c$. This process can be repeated (dashed line) to create a temporal abstract roll-out.

predictions. The action loss drives the system to minimize the KL divergence between the posterior high-level action distribution Q_θ and the prior distribution P_θ . The exact loss functions can be found in Suppl. E.3.4.

In sum, our THICK world model augments traditional flat world models by a high level, which learns predictions of variable length, anticipating situations where low-level context changes occur. We have carefully designed the low-level C-RSSM and the high-level world model to seamlessly switch between coarse low-level predictions and temporal abstract high-level predictions. This process is outlined in Fig. 7.4 and the supplementary Algorithm 2 (see Suppl. E.1). Given a context c_t , stochastic state z_t , and sampled high-level action \hat{A}_t , the high-level model W_θ predicts a scenario $(\hat{a}_{\tau(t)-1}, \hat{z}_{\tau(t)-1})$ immediately before the next anticipated context change. By design, the context should remain unchanged until then. By feeding this prediction into the coarse processing pathway of C-RSSM, we can predict the subsequent, new context $c_{\tau(t)}$ (Eq. 7.2) and a coarse prior estimate of the corresponding stochastic state $\hat{z}_{\tau(t)}^c$ (Eq. 7.4). Longer *temporal abstract roll-outs* can be created by feeding $c_{\tau(t)}$ and $\hat{z}_{\tau(t)}^c$ again into W_θ (dashed line in Fig. 7.4). In this way, the actual context change predictions are generated naturally by C-RSSM.

7.3 Downstream Applications of THICK World Models

World models have been applied in many downstream tasks, including MBRL (Ha & Schmidhuber, 2018; Hafner et al., 2019a, 2020a, 2023), exploration (Sekar et al., 2020; Sancaktar et al., 2022), or model-predictive control (MPC) (Hafner et al., 2019b; Vlastelica et al., 2022). With minimal changes, the hierarchical roll-outs from THICK can be seamlessly integrated where flat roll-outs were previously utilized. We exemplify this integration in two key areas: MBRL and MPC.

7.3.1 THICK Dreamer: MBRL with Hierarchical Roll-outs

Dreamer (Hafner et al., 2019a) learns behavior by training an actor and a critic from the “imagined” roll-outs of its RSSM world model. More specifically, Dreamer imagines a sequence of states $\mathbf{s}_{t:t+H}$ from a start state \mathbf{s}_t given an actor-generated action sequence $\mathbf{a}_{t:t+H}$. Dreamer computes the general λ -return $V^\lambda(\mathbf{s}_t)$ (Sutton & Barto, 2018) that is recursively defined as follows

$$V^\lambda(\mathbf{s}_t) = \hat{r}_t + \hat{\gamma}_t \begin{cases} (1 - \lambda)v_\xi(\mathbf{s}_t) + \lambda V^\lambda(\mathbf{s}_{t+1}) & \text{for } t < H, \\ v_\xi(\mathbf{s}_{t+1}) & \text{for } t = H \end{cases} \quad (7.20)$$

with \hat{r}_t and $\hat{\gamma}_t$ the rewards and discounts predicted based on \mathbf{s}_t (see Eq. 7.7). The critic v_ξ is trained via

$$\mathcal{L}(\xi) = \mathbb{E}_{p_\phi} \left[\sum_{t=1}^H \frac{1}{2} \left(v_\xi(\mathbf{s}_t) - \text{sg}(V^\lambda(\mathbf{s}_t)) \right)^2 \right], \quad (7.21)$$

to regress the λ -return using a squared loss, with the stop gradient operation $\text{sg}(\cdot)$.

In sparse reward tasks, one challenge is the propagation of rewards to train the critic (Andrychowicz et al., 2017; Patil et al., 2021). Here, Dreamer faces a difficult trade-off: Long roll-outs (large H) accelerate reward propagation but degrade the quality of the predicted roll-outs. We propose **THICK Dreamer**, which combines value estimates from low- and high-level predictions to boost reward propagation. THICK Dreamer maintains an additional critic v_χ to evaluate temporal abstract predictions. Like Dreamer, we first imagine a low-level roll-out of H states $\mathbf{s}_{t:t+H}$. Additionally, for every time t in the roll-out, we predict a temporal abstract outcome $\mathbf{c}_{\tau(t)}$ and $\mathbf{z}_{\tau(t)}$ and estimate a long-horizon value V^{long} as

$$V^{\text{long}}(\mathbf{s}_t) = \hat{r}_{t:\tau(t)}^\gamma + \gamma^{\Delta t} \left(\hat{r}_{\tau(t)}^c + \hat{\gamma}_{\tau(t)}^c v_\chi(\hat{\mathbf{c}}_{\tau(t)}, \hat{\mathbf{z}}_{\tau(t)}) \right), \quad (7.22)$$

with all variables predicted via the THICK world model and immediate rewards via Eq. 7.8 of C-RSSM given THICK’s world model predictions (cf. also supplementary Algorithm 2). We estimate the value of a state \mathbf{s}_t as a mixture of short- and long-horizon estimates with

$$V(\mathbf{s}_t) = \psi V^\lambda(\mathbf{s}_t) + (1 - \psi) V^{\text{long}}(\mathbf{s}_t), \quad (7.23)$$

where the hyperparameter ψ controls the trade-off between the two estimates. We set $\psi = 0.9$ in all experiments and train both critics v_ξ and v_χ to regress the value estimate using a squared loss:

$$\mathcal{L}(\nu) = \mathbb{E}_{p_\phi} \left[\sum_{t=1}^H \frac{1}{2} \left(v_\nu(\mathbf{s}_t) - \text{sg}(V(\mathbf{s}_t)) \right)^2 \right], \quad (7.24)$$

for the two critics $v_\nu \in \{v_\xi, v_\chi\}$ with parameters $\nu \in \{\xi, \chi\}$, and $\text{sg}(\cdot)$ the stop gradient operator. In summary, to speed up credit assignment when learning a value function, THICK Dreamer combines low-level roll-outs with temporal abstract predictions to additionally estimate the value of likely long-horizon outcomes.

7.3.2 THICK PlaNet: Hierarchical MPC

The original RSSM was proposed in PlaNet (Hafner et al., 2019b) as a world model for MPC. PlaNet searches for the optimal action sequence $\mathbf{a}_{t:t+K}^*$ to maximize the predicted returns $\hat{r}_{t:t+K}$ for a planning horizon K . Therefore, PlaNet employs zero-order trajectory optimization through the cross-entropy method (CEM) (Rubinstein & Davidson, 1999). Once $\mathbf{a}_{t:t+K}^*$ is identified, the initial action \mathbf{a}_t^* is executed, and the procedure is repeated.

CEM optimizes randomly sampled trajectories. Sampling a good action sequence is exponentially harder for increasing task horizons. We hypothesize that such tasks could be solved with much fewer high-level actions. For this, we propose **THICK PlaNet**. THICK PlaNet plans on the high level to solve the task and uses the low level to follow this plan. We define a reward function $R(\cdot)$ to estimate the return of a high-level action sequence $\mathbf{A}_{1:K}$ with length K recursively as

$$R(\mathbf{A}_{k:K}, t') = \hat{r}_{t':\tau(t')}^\gamma + \gamma^{\Delta t} \begin{cases} \hat{r}_{\tau(t')}^c + \hat{\gamma}_{\tau(t')}^c R(\mathbf{A}_{k+1:K}, \tau(t') + 1) & \text{for } k < K, \\ \hat{r}_{\tau(t')}^c & \text{for } k = K \end{cases} \quad (7.25)$$

with all variables predicted via a temporal abstract roll-out (see Sec. 7.2.2) starting with $k = 1$ and $t' = t$. We search for the optimal sequence $\hat{\mathbf{A}}_{1:K}^*$ maximizing $R(\cdot)$ with Monte Carlo Tree Search. Based on the first action $\hat{\mathbf{A}}_1^*$ we sample a subgoal $\hat{\mathbf{z}}_t^{\text{goal}} \sim F_\theta(\hat{\mathbf{z}}_t^{\text{goal}} | \hat{\mathbf{A}}_1^*, \mathbf{c}_t, \mathbf{z}_t)$. This subgoal is valid as long as it has not been reached and nothing has changed drastically in the environment. Thus, we only replan on the high level when the context has changed. We apply CEM at the low level to reach $\mathbf{z}_t^{\text{goal}}$ while also maximizing task return, with

$$\mathbf{a}_{t:t+K}^* = \arg \max_{\mathbf{a}_{t:t+K}} \sum_{t'=t}^{t+K} \hat{r}_{t'} + \kappa \text{sim}(\mathbf{z}_{t'}, \mathbf{z}_t^{\text{goal}}) \quad \text{with} \quad \hat{r}_{t'} \sim o_\phi(\hat{r}_{t'} | \mathbf{s}_{t'}), \quad (7.26)$$

for a planning horizon K .^{7.7} The function $\text{sim}(\cdot)$ is a measure of similarity between $\mathbf{z}_t^{\text{goal}}$ and \mathbf{z}_t . The hyperparameter κ controls the trade-off between external and internal rewards. Previously, the similarity between the Gaussian distributed \mathbf{z}_t of the RSSM was

^{7.7}This corresponds to hierarchical planning in THICK MDPs according to Eq. 3.5, but replacing minimization of subgoal divergence (KL term in Eq. 3.5) with maximization of similarity to a subgoal.

estimated using cosine similarity (Mendonca et al., 2021). However, for categorically distributed \mathbf{z}_t , the cosine similarity can be low even when they come from the same distribution. Instead, we use the cosine similarity of the logits, i.e.

$$\text{sim}(\mathbf{z}_t, \mathbf{z}_t^{\text{goal}}) = \frac{\boldsymbol{\omega}_t \cdot \boldsymbol{\omega}_t^{\text{goal}}}{\|\boldsymbol{\omega}_t\| \|\boldsymbol{\omega}_t^{\text{goal}}\|}, \quad (7.27)$$

where \cdot is the dot product and $\boldsymbol{\omega}_t$ and $\boldsymbol{\omega}_t^{\text{goal}}$ are the logits of the distributions that produced \mathbf{z}_t and $\mathbf{z}_t^{\text{goal}}$, respectively. Thus, we compute the similarity between the sampling distributions rather than the similarity of the samples. Compared to other similarity measures, e.g. KL divergence, our measure has the desirable property that $\text{sim}(\mathbf{z}_t, \mathbf{z}_t^{\text{goal}}) \in [0, 1]$, which simplifies setting the hyperparameter κ , which we set to $\kappa = 0.025$ to mainly guide the behavior in the absence of external reward.

7.4 Experiments

We empirically evaluate THICK to answer the following questions:

- **Can THICK learn temporal abstractions?** We show that the learned high-level world model indeed discerns meaningful, interpretable temporal abstractions across various scenarios (Sec. 7.4.2).
- **Can THICK’s hierarchical predictions improve MBRL?** We show that THICK Dreamer achieves higher returns than Dreamer in long-horizon tasks with sparse rewards (Sec. 7.4.3).
- **Can THICK’s world model be used to plan hierarchically?** We show that MPC with THICK world models is better than flat world models at solving long-horizon tasks (Sec. 7.4.4).

7.4.1 Simulation Environments

We evaluate our THICK world models in various scenarios with pixel-based inputs.

MiniHack (Samvelyan et al., 2021) is a sandbox framework for designing RL environments based on Nethack (Küttler et al., 2020). We test our system on benchmark problems, as well as newly created tasks. All problems, detailed in Suppl. E.4.1, have hierarchical structures in which sub-goals must be achieved (e.g. fetch a wand) to fulfill a task (e.g. kill a monster) to exit a dungeon and receive a sparse reward. The observation is a pixel-based, ego-centric view of ± 2 grid cells around the agent. MiniHack uses discrete actions. All problems are described in Suppl. E.4.1.

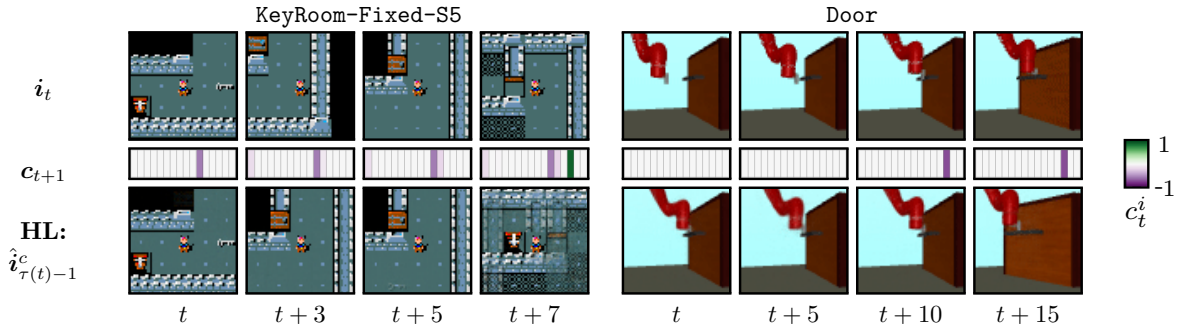


Figure 7.5: Exemplary context changes. We show the input images i_t , 16-dimensional contexts c_{t+1} and reconstructions $\hat{i}_{\tau(t)-1}^c$ of the high-level predictions. For **KeyRoom** the context changes when finding a key, picking it up, opening a door (here from a diagonally adjacent grid), or exiting the room. In **Door** the context changes when the robot grabs the handle or the door is fully opened. The high level predicts the states before the next changes.

VisualPinPad (Hafner et al., 2022) is a suite of long-horizon visual RL problems. Here, an agent (black square) needs to step on a fixed sequence of pads to receive a sparse reward. We use three levels of difficulty based on the number of pads and target sequence length (three, four, five).

MultiWorld (Pong et al., 2018) is a suite of robotic manipulation tasks for visual RL. In these tasks, a Sawyer robot has to either move an object to a goal position (puck in **Pusher** or ball in **PickUp**) or open a door (**Door**). We use fixed goals and take the normalized distance between the to-be-controlled entity and the goal position as dense rewards (in **Pusher-Dense**, **PickUp**, **Door**) and thresholded distances as sparse rewards (in **Pusher-Sparse**). Details are provided in Suppl. E.4.2.

7.4.2 Interpretable Contexts and Hierarchical Predictions

First, we analyze the predictions of THICK world models across diverse tasks. Example sequences are displayed in Fig. 7.5 and Suppl. E.5.1. In **MiniHack**, context updates typically coincide with item discovery, item collection, map changes, area exploration, or dungeon exits. In **Multiworld**, context changes occur due to object interactions or at workspace boundaries. In **VisualPinPad**, activating the pads can prompt context changes. The high-level model predicts the states preceding context changes, often abstracting details, leading to blurry reconstructions. For instance, in **KeyRoom**, the system forecasts the agent’s level exit without knowledge of the exact room layout (Fig. 7.5, $t+7$). Nonetheless,

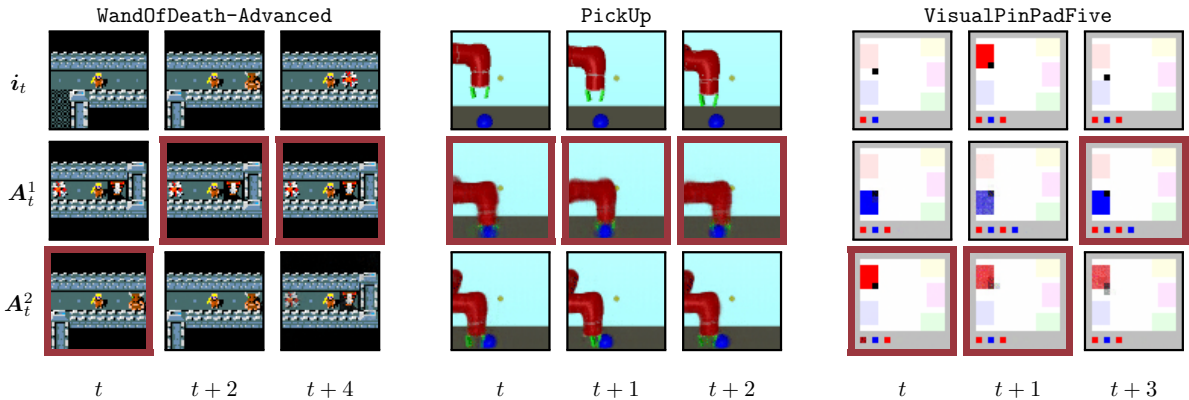


Figure 7.6: Exemplary high-level actions \mathbf{A}_t . We show input images \mathbf{i}_t and predictions $\hat{\mathbf{i}}_{\tau(t)-1}^c$ for two high-level actions \mathbf{A}_t^1 and \mathbf{A}_t^2 . Red frames depict sampled actions $\hat{\mathbf{A}}_t$. Exemplar actions \mathbf{A}_t are exiting the room or attacking a monster (left), grasping or pushing a ball (center), and activating pads (right).

the lower level consistently predicts the next frames accurately, as shown in Fig. 7.1b.

Abstract action representations \mathbf{A}_t emerge on the high level, as illustrated in Fig. 7.6. These actions categorically encode different agent-world interactions, e.g., grasping or pushing a ball in `PickUp` (center in Fig. 7.6) or stepping on different pads in `VisualPinPadFive` (right in Fig. 7.6). The prior Q_θ learns to sample actions based on the likelihood of their outcomes (red frames in Fig. 7.6). If there are more actions \mathbf{A}_t than necessary, different actions encode the same outcome. We provide more examples and analysis of predictions and contexts in Suppl. E.5.1.^{7,8}

7.4.3 Model-Based Reinforcement Learning

We investigate whether hierarchical roll-outs can improve MBRL in the MiniHack suite by comparing THICK Dreamer with DreamerV2 (Hafner et al., 2020a) and Director (Hafner et al., 2022), a hierarchical RL method based on Dreamer. Fig. 7.7a–7.7d show that THICK Dreamer matches or outperforms flat Dreamer in all tasks in terms of sample efficiency or overall success rate. The advantage of THICK Dreamer is more pronounced in tasks that require the completion of multiple subgoals (e.g. completing five subgoals in `EscapeRoom` vs. finding a key to open a door in `KeyRoom`). Director outperforms the other methods in `KeyRoom`, but fails to learn other MiniHack tasks. We investigate the

^{7,8}Animations are also provided on our supplementary website <https://sites.google.com/view/thick-world-models>.

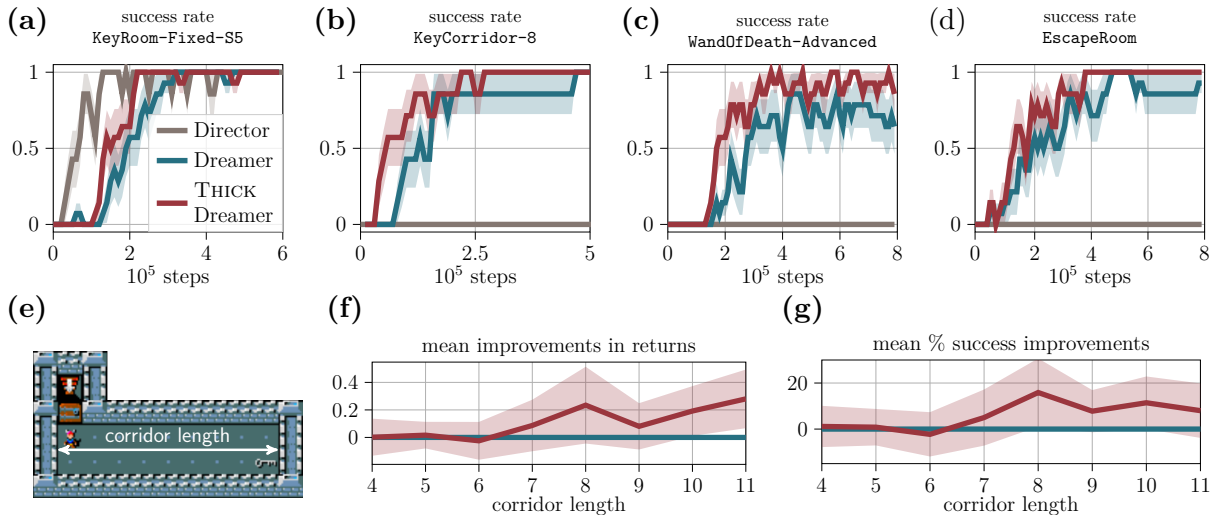


Figure 7.7: Long-horizon RL in MiniHack. Top graphics (a-d) plot the mean success rate during evaluation for various MiniHack tasks using 7 seeds. For *KeyCorridor* (e) we systematically vary the length of the corridor and plot the mean difference in evaluation returns (f) and percentage of task success (g) between THICK Dreamer and Dreamer over different lengths. The shaded areas depict \pm one standard error.

failure cases of Director in Suppl. E.5.3 and show more MiniHack results in Suppl. E.5.2.

We hypothesize that the length of the task horizon is the main factor boosting THICK Dreamer’s performance. To investigate this, we systematically vary the task horizon in the *KeyCorridor* problem (see Fig. 7.7e) by modifying the length of the corridor. Fig. 7.7f–7.7g plot the mean difference in the rewards obtained and the success rate over 500k training steps between THICK Dreamer and Dreamer for different corridor lengths. The performance gain of THICK Dreamer tends to increase with the length of the corridor. At some length, both approaches fail to discover rewards, detailed in Suppl. E.5.2.

We further analyze the effect of the task horizon in *VisualPinPad*. *VisualPinPad* presents two challenges: exploration and long-horizon behavior. To analyze the latter in isolation, we bypass the challenge of discovering the sparse rewards by initially filling the replay buffer of all models with 1M steps of exploration using Plan2Explore (Sekar et al., 2020) (details in Suppl. E.5.4). Fig. 7.8 shows the performance of THICK Dreamer, DreamerV2, and Director. THICK Dreamer matches Dreamer in *VisualPinPadThree* and is slightly more sample efficient in the more challenging tasks.^{7.9} Thus, fusing hierarchical

^{7.9}Previously, Hafner et al. (2022) reported that Director outperforms Dreamer in *VisualPinPad*. We believe this improvement stems from more sophisticated exploration, which is not necessary in our setting.

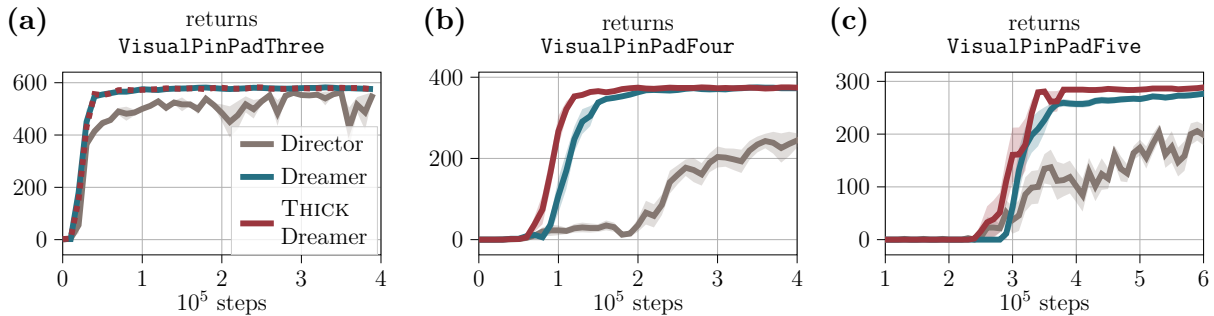


Figure 7.8: Sample efficient RL in VisualPinPad. We plot the mean evaluation returns for 7 seeds. Shaded areas depict the standard error.

predictions to train a single policy in THICK Dreamer seems better suited for long-horizon learning than the hierarchical policies of Director or not employing hierarchies.

7.4.4 Zero-Shot Model-Predictive Control

Lastly, we analyze whether our hierarchical predictions are suitable for planning by comparing THICK PlaNet and PlaNet (Hafner et al., 2019b) in Multiworld. We consider the challenging setup of MPC for models trained on an offline dataset of 1M samples collected by Plan2Explore (Sekar et al., 2020). Fig. 7.9 shows the zero-shot performance over training. For Pusher-Dense, i.e. a short-horizon task^{7.10} with dense rewards, there is no notable difference between both methods. When rewards are sparse (Pusher-Sparse) or the task horizon is long (Door and PickUp), THICK PlaNet achieves higher returns than PlaNet. Additionally, the sub-goals set by the high level can be decoded, shown in Suppl. E.5.6, which improves the explainability of the system’s behavior.

7.5 Related Work

Hierarchical RL (HRL): HRL is an orthogonal research direction to hierarchical world models. In HRL, a high-level policy either selects a low-level policy or provides goals or rewards for a low level (Pateria et al., 2021; Eppe et al., 2022). In contrast, our THICK Dreamer uses high-level predictions to train a flat RL agent. Typically in HRL, the high level operates on fixed time scales (Hafner et al., 2022; Nachum et al., 2018; Vezhnevets et al., 2017; Grtler et al., 2021). Alternatively, the high level is activated

^{7.10}Since the puck starts between the gripper and goal, the task can be solved by moving directly to the goal.

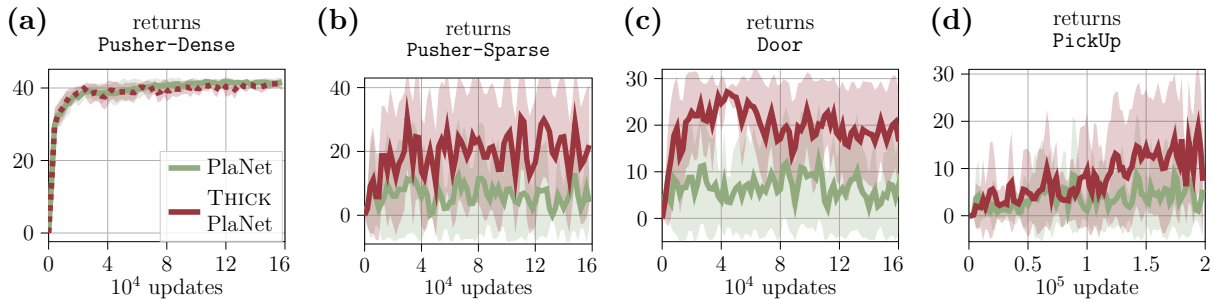


Figure 7.9: Zero-shot MPC in Multiworld. Each graphic plots the mean returns for zero-shot planning in Multiworld over world model updates using 10 seeds. Shaded areas depict the standard deviation.

task-dependently based on subgoal completion (Bacon et al., 2017; Levy et al., 2019). In THICK world models, the high level is learned time- and task-independently purely from predictions and latent state regularization.

Temporal abstractions with fixed time scales: Saxena et al. (2021) introduced a hierarchical video prediction model (i.e. without action) that used different clock speeds at each level to learn long-term dependencies using pixel-based input. Although this was suitable for learning slow-moving content at higher levels of the temporal hierarchy, unlike C-RSSM and THICK, it requires explicitly defining the time scale factors.

Temporal abstractions from predictability: Adaptive Skip Intervals (ASI) (Neitz et al., 2018) is a method for learning temporal abstract autoregressive predictions. In ASI, a network is trained to predict those inputs within a predefined horizon that best allow predicting extended sequences into the future. As a result, the model learns to skip a number of inputs towards predictable transitions. Similarly, temporal-agnostic predictions (TAP) (Jayaraman et al., 2019) identify frames of a video within a time horizon that are highly predictable. TAP is then trained to only predict those predictable “bottleneck” frames. Zakharov et al. (2022b) provide a learning-free mechanism to detect context change by evaluating how predictable future states are. Briefly, their approach detects changes in latent representation of each layer in the model hierarchy and introduces temporal abstraction by blocking bottom-up information propagation between different contexts. Another approach is to use unexpected prediction errors from a forward model for self-supervised time series segmentation (Gumbsch et al., 2019). Here, the idea is that in certain states the dynamics of the agent-environment interactions change, e.g. changing the terrain during locomotion, which leads to a temporary spikes in the prediction errors.

Temporal abstractions from learning boundary detectors: In addition to using indirect measures to segment a sequence, a straightforward approach is to train a boundary

detector that signals the boundary of subsequences (Kim et al., 2019; Zakharov et al., 2022a). Kim et al. (2019) train a boundary detector that is regularized by specifying the maximum number of subsequences allowed and their maximum length. This requires prior knowledge about the training data and imposes hard constraints on the time scales of the learned temporal abstractions. Our sparsity loss instead implements a soft constraint – while a low number of context changes is desired, C-RSSM can violate this constraint if prediction accuracy demands it. Conversely, Zakharov et al. (2022a) introduced a boundary detection mechanism using a non-parametric posterior over the latent states. Here, the model learns to transition between states only if a change in the represented features had been observed, otherwise temporally persistent states were clustered together.

Different world model backbones: An alternative approach to developing hierarchies for long-horizon learning in world models is to enhance the memory module. For example, Transformers (Vaswani et al., 2017) can improve the learning of long-horizon dependencies in world models (Chen et al., 2022; Robine et al., 2023; Micheli et al., 2023). Along similar lines, Deng et al. (2023) showed that replacing the GRU with an S4 model (Gu et al., 2021) improves the memory and long-horizon predictions of the RSSM. However, neither applying Transformers nor S4 leads to temporal abstract predictions that can serve as subgoals for hierarchical planning, necessary for THICK PLaNet.

7.6 Discussion

We have introduced C-RSSM and THICK– fully self-supervised learning methods to construct hierarchical world models. By embedding a GATELORD cell within the RSSM and imposing a sparsity objective, C-RSSM develops context codes that update only in critical situations, where prediction-relevant aspects of the environment change, e.g. when the agent interacts with objects. As a result, C-RSSM tends to generate an approximation of a hierarchical hidden Markov world model, where the context conditions approximate lower-level Markov models. On a higher level, THICK learns to anticipate context-altering states. Categorical high-level action codes enable the anticipation of different outcomes, accounting for multiple lower-level context transitions. As a result, THICK world models can predict both abstract context transitions and exact low-level dynamics. Additionally, we have shown that through minor changes to existing MBRL and MPC methods, hierarchical predictions can improve long-horizon learning.

THICK relies on setting the hyperparameter β^{sparse} , which determines the high-level segmentation. Ideally, this hyperparameter should be tuned for every task. However, we found that the same value works well across similar tasks, e.g. setting the value only per suite. Furthermore, except for improving long-horizon learning, our downstream applications have similar restrictions as the method they build upon. For example, if

Dreamer never discovers a solution to a task, THICK cannot decompose it.

A promising direction for extending THICK world models is the combination of MCTS with RL (Schrittwieser et al., 2020). The system could search for high-level goals that goal-condition low-level policies (Nasiriany et al., 2019; Akakzia et al., 2021) –a technique that may even be related to computations unfolding in our brains (Mattar & Lengyel, 2022).

Another potential lies in the integration of more active epistemic-driven exploration (Sekar et al., 2020; Sancaktar et al., 2022) on both levels of the hierarchy. The agent could, for example, either seek out context transitions that cause uncertainty on the high level or actively avoid transitions to further explore a certain context. This could lead to a more robust consolidation of context codes and transitions between them and foster the exploration of long-horizon temporal dependencies.

Future extensions could also explore richer predictions purely from the context \mathbf{c}_t . This would allow the high-level to directly predict context transitions without predicting observable state information used for intermediate queries to the low-level.

Lastly, we employed THICK to establish a two-level hierarchy of world models. However, the principles of THICK could be applied on multiple levels to build an N -level world model hierarchy. A potential generalization of THICK, detailed in Chap. 3, could implement each level as a C-RSSM and thus recursively develop a hierarchy of nested time scales. An alternative route would be to add multiple coarse prediction pathways and contexts to the low-level world model that are regularized with separate sparsity parameters β^{sparse} . For each coarse prediction pathway, a separate high-level model could be trained. As a result, the network would maintain multiple levels of abstraction in parallel and could learn to select which level is suitable for the task at hand. However, in this approach the time scales would not necessarily be nested.

In sum, we see great potential for THICK world models as a tool to build more sophisticated agents that explore and plan on multiple time scales and develop increasingly abstract levels of prediction.

8

Discussion

I conclude this thesis by discussing the general approach and results of this thesis. For this, I first summarize my contributions before critically analyzing them. In addition, I discuss the relations and implications of my approach for both artificial intelligence research and cognitive science. Last but not least, I provide an outlook on how future research could build on and benefit from the presented work.

8.1 Summary of Contributions

The primary objective of my research is to improve the problem-solving capabilities of artificial intelligence (AI). The review of the literature in Chap. 1 revealed that two mechanisms seem to be present in biological agents but are still largely missing in many artificial agents: (1.) Artificial systems often struggle with **generalization across distributions**, presumably because they lack the ability to encode the compositional and causal structures of the environment. (2.) **Hierarchical model-based planning** is underrepresented in many planning systems, although it could enable a system to solve complex problems by internally simulating the necessary sub-steps of the solution. In humans, causal inference and hierarchical planning have been linked to **event cognition**. Converging theories from different disciplines propose that humans seem to memorize the past, reason about the present, and predict the future in terms of discrete, hierarchically organized models of events. Therefore, I set out to **equip artificial agents** with the

ability to encode their sensorimotor experience by means of events to improve their problem-solving capabilities through better generalization and hierarchical planning.

After reviewing the theoretical background of event cognition in biological agents and sequential decision making in artificial agents in Chap. 2, I combined these insights into **THICK MDP**, a **theoretical framework of hierarchical event-based decision making**. The main proposal of this framework is THICK (Temporal Hierarchies from Invariant Context Kernels) an **algorithm to learn hierarchical abstractions** of transition functions. I proposed hierarchical processing with a specific nested structure. That is, persistent latent states or *contexts* encode the events unfolding at each level of abstraction. In THICK, a higher level would only become active when the context of the next lower level changes. In Chap. 3, I detailed how such bottom-up activation of levels during learning could give rise to hierarchical abstractions with nested time scales; and, inversely, how top-down activation could enable hierarchical goal-directed planning.

With THICK MDPs, I made two claims: (1.) The framework is a simplified model of the processes involved in human event perception and anticipatory goal-directed planning, and (2.) implementing such a structure for artificial agents could improve their planning capabilities. In the following chapters, I then provided evidence for these claims employing THICK MDPs to **model human goal-anticipatory behavior** and to **improve state-of-the-art planning and reinforcement learning (RL)**.

To investigate cognitive plausibility, **CAPRI** was outlined in Chap. 4. CAPRI is a predictive model, following the structure of THICK MDP, which encodes its interactions with the environment through two levels of a prediction hierarchy. CAPRI showed that when gaze behavior is selected to minimize uncertainty about both levels of prediction, the system **developed goal-anticipatory gaze behavior**, i.e. it shifted its gaze to the goal of a reaching movement in anticipation. Crucially, this behavior emerged with increasing experience in reaching and only for familiar hand agents, not for unfamiliar claw agents. This qualitatively matches the experimental data of eye-tracking studies for infants under 12 months of age (Kanakogi & Itakura, 2011; Adam & Elsner, 2020). Thus, inferring actions to minimize predicted uncertainty, which has been proposed elsewhere (Friston et al., 2015; Parr et al., 2022), combined with the structure of THICK MDPs seems to explain numerous findings on the development and functionality of goal-anticipatory gaze behavior in infants.

Next, the challenge of learning to segment sensorimotor activity from scratch was tackled. Chap. 5 introduced **GATELORD**, a recurrent neural network (RNN) designed to maintain piecewise constant latent states, meaning its memory is only updated rarely. GATELORD employs a specialized update gate and an auxiliary loss term to **regularize its latent states towards sparse changes in time**. In Chap. 5 we experimentally demonstrated in a variety of scenarios, ranging from grid-world scenarios to realistic

physics simulations, that replacing other RNNs by GATELORD brings **several advantages for prediction, planning, and RL**: GATELORD boosts sample efficiency of sparse-reward RL, enhances long-horizon prediction, and improves generalization across training schemes and spurious correlations in its training data. Furthermore, GATELORD naturally **segments activity** into parts with stable constant latent state activity, similar to events, and detects sparse points in between, similar to event boundaries, in which latent states are updated.

In Chap. 6, we employed GATELORD to learn hierarchical predictions to overcome previous limitations for **modeling goal-anticipatory gaze behavior through self-supervised hierarchical predictions**. In our hierarchical deep learning architecture the lower level maintains piecewise constant latent states and a high-level skip network is trained to predict only situations in which latent states change. Experiments in a robotic manipulation scenario showed that latent states developed that encode simple interaction events, e.g. reaching. In addition, the skip network learned to make meaningful predictions about upcoming event boundaries. In Chap. 6, we partially reproduced the modeling findings of CAPRI, demonstrating that the modeling approach holds for a more complicated setup, i.e. in a more realistic simulator, with self-supervised segmentations, and when internally simulating motor commands required for the observed movement.

Finally, Chap. 7 introduced **THICK world models**, i.e. hierarchical, generative forward models for high-dimensional inputs. As before, the low-level model encoded its interactions via sparsely changing latent states and a high-level model predicted potential situations in which the lower-level latent states change. Therefore, the system developed **categorical high-level action encodings** that allowed the system to disambiguate different context changes, or event boundaries, that it might encounter. In Chap. 7, we showed that THICK world models can **outperform flat models in both RL and planning**: In particular, we demonstrated that anticipating the next abstract temporal outcome can increase the sample efficiency of learning a value function in model-based RL in various tasks with sparse rewards. Furthermore, we showed that through high-level planning, various robotic manipulation tasks could be solved more efficiently.

In summary, the present research showed how **latent states that encode events** can develop and **improve prediction and planning** of a wide set of deep learning agents with different input modalities and action spaces, as well as within different environments. Furthermore, I outlined and experimentally demonstrated how such latent states can give rise to **hierarchical predictions** suitable for hierarchical planning. In addition to that, **cognitive plausibility** of these mechanisms was demonstrated by modeling experimental data of human goal-anticipatory gaze behavior. Thus, this thesis also provides a model that explains various findings on the development of goal anticipations in infants.

Nevertheless, much progress is needed to achieve the behavioral flexibility exhibited by

intelligent animals, such as the crow in Fig. 1.1. In what follows, I first critically discuss current shortcomings and potential extensions of the presented methods for segmenting activity (Sec. 8.2.1), toward more explicitly encoding objects (Sec. 8.2.2), and extensions for other forms of hierarchical learning (Sec. 8.2.3). Next, I discuss the general approach of taking inspiration from cognitive science for building AI (Sec. 8.3). Subsequently, I relate my approach to other theories and notions in cognitive science (Sec. 8.4). Lastly, I provide an outlook on how future research could build on this work (Sec. 8.5).

8.2 Limitations and Extensions

8.2.1 Segmentation of Temporal Abstractions

For learning hierarchical model-based abstraction, we rely on a meaningful segmentation of sensorimotor activity. As a backbone for the segmentation of time series, the presented methods employed GATELORD. GATELORD segments activity by means of piecewise-constant latent states that are regularized towards temporally sparse updates. When used in predictive models, this segmentation criterion is closely related to surprise- or prediction error-based segmentation (Zacks et al., 2007; Reynolds et al., 2007; Gumbsch et al., 2019; Franklin et al., 2020; Basgol et al., 2024), because the loss of GATELORD aims to minimize prediction error while simultaneously minimizing latent state changes.

Since the latent state regularization is defined *relative* to the prediction error (or other loss terms), the segmentation hinges on the predictability of the data and the prediction power of the overall model. The hierarchical models in Chap. 6 and Chap. 7 only employ feed-forward networks besides GATELORD for their (coarse) predictions. Thus, the systems segments activity mainly on the basis of partial observability. That is, the boundaries correspond to situations in which latent factors in the environment change or task-relevant information needs to be memorized, e.g. a key is picked up. Thus, tasks could be easily constructed where this approach fails to discover a suitable segmentation of time series, e.g. making environments fully observable and trivial to predict.

That being said, I hypothesize that sufficiently complex, realistic applications tend to be partially observable, and their latent generative factors tend to change somewhat systematically. Complex systems, e.g. biological agents, have restricted sensors, which creates the need to maintain latent beliefs about entities and events that unfold in their environment. Furthermore, in practice, GATELORD updated its latent states even in fully observable scenarios, triggered, for example, by sudden strong perceptual changes whose prediction needs to be timed precisely.^{8.1} Scaling the impact of latent state regularization

^{8.1}An example for this is the VisualPinPad suite in Chap. 7 which is fully observable. Stepping on a pad causes the pad to light up which is often accompanied by a context update.

on the overall model loss^{8.2} can also compensate for prediction difficulty (or simplicity) of a task.

Future work could extend the segmentation of time series by further criteria to develop more robust temporal abstractions. For example, changes in objects and their interactions (detailed in Sec. 8.2.2) or changes in a causal graph (Pitis & Garg, 2020; Seitzer et al., 2021; Pitis et al., 2022) seem promising candidates to supplement the current segmentation based on latent state stability.

8.2.2 Towards Object-Centric Representations

Events revolve around entities and their interactions. Most events describe the spatiotemporal dynamics of at least one entity, potentially affecting other entities. Gärdenfors (2014) proposes that a prototypical event describes an agent entity generating a force that can affect a patient entity. According to Gärdenfors (2014), the event structure is somewhat analogous to the subject-object structure of sentences, where the roles of the subject or objects are filled by the involved entities.

Hence, the role of entities and their interactions seems to be crucial for the structure of events. However, in the presented research I did not fully make use of this structure for the segmentation or the representation of temporal abstractions.^{8.3} Future work could explore whether **encoding events using object-centric representations** can guide the discovery and learning of event-based temporal abstractions.

Graph neural networks (GNNs) are a common approach to encode objects and their interactions (Battaglia et al., 2018). GNNs represent the state of an entity as a node and the interaction between objects as an edge. The per-edge and per-node functions are typically reused within the graph, thus supporting combinatorial generalization across objects (Battaglia et al., 2018). Extending GNN-based dynamics models to include events is a promising research direction to better include the role of entities.

How could a GNN-based world model be extended to include events? One approach could be to reuse GATELORD’s regularized update gates to **sparsely route the information flow across entities**. By embedding sparse gates at the edges of a GNN, one could control which aspects of an entity’s state currently affect another entity, illustrated in Fig. 8.1. This essentially implements a bias towards sparse object interactions and could guide the system towards discovering local causal relationships (Goyal & Bengio,

^{8.2}The impact of latent state regularization can be scaled by tuning the hyperparameter β (Chap. 5–6) or β^{sparse} (Chap. 7).

^{8.3}For example, the x -, y -, and z -position was provided for a hand, object and a goal entity for learning temporal abstractions of reaching in Chap. 6. However, the system did not explicitly encode that a three-dimensional position triple belongs to a single entity.

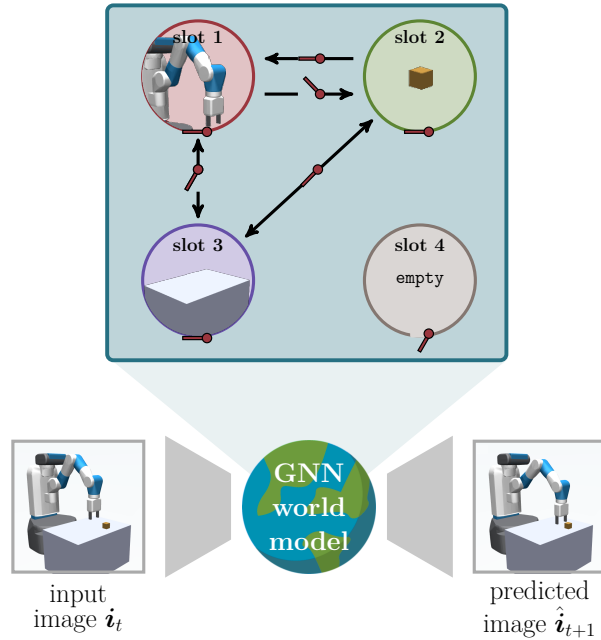


Figure 8.1: Towards object-centric world models: In a Graph Neural Network (GNN) world model the information flow between entities could be controlled through gates regularized toward sparse activity, as before in GATELORD (cf. Chap. 5). This could allow adaptively determining the number of objects (gates at nodes) and detecting object interactions (gates at edges).

2022; Seitzer et al., 2021). For example, when a robot reaches for a block, the position of the block influences the next movement of a gripper (Fig. 8.1, arrow from green to red node). However, in reverse, the gripper would not affect the position of an object while out of reach (open gate at the arrow from red to green node). Event boundaries could be detected by registering changes in the interactions of entities. For example, upon contact, the gripper would affect the future positions of the object.

In addition to guiding event segmentation, sparsely-operating gates could help **extract objects from pixel-based inputs**. Recently, slot-based vision systems have been developed that assign parts of an image to slots, thus discovering objects from pixel-based inputs (Locatello et al., 2020; Kipf et al., 2021; Traub et al., 2023; Seitzer et al., 2023; Zadaianchuk et al., 2023). However, typically only a fixed number of objects are detected. Gating the recruitment of new slots via sparse gates could enable the detection of a variable number of objects. The system could learn to assign new slots, causing a gate opening and sparsity penalty, only if otherwise prediction losses ($\mathcal{L}^{\text{task}}$) would increase.

Changes in the number of active slots could also indicate event boundaries, for example, if an object falls out of a box once the box is picked up.

In summary, future work could explore the combination of event-based abstraction and object-centric representations. This could improve the discovery of temporal abstractions and generalization in model-based RL. Consider, for example, a block-stacking robot. By encoding interaction events in an object-centric structure, an RL agent could learn to seamlessly generalize across the number of objects, e.g. blocks. Changes in the interaction graph could mark event boundaries. Additionally, the system could potentially learn to better generalize across out-of-distribution shifts by learning which aspects of an object state, e.g. position, are affected by an event, e.g. lifting, and which are not, e.g. mass and color. Thus, physical reasoning could be learned from simple object interaction events, similar to how it can be observed in children during their first years of life (Lin et al., 2022).

8.2.3 Extending Model-Based Hierarchies

The hierarchical model-based abstractions discovered by the presented methods are best characterized as a *partonomy* of events, i.e. a hierarchical organization reflecting the relationship of parts and subparts (Zacks & Tversky, 2001). For example, moving a gripper towards an object is part of a <reaching for an object>-event. According to Zacks & Tversky (2001), events can also be organized into a *taxonomy*, i.e. a hierarchical organization that defines an “instance-of” relationship between abstraction levels. For example, <filling water into a teapot> or <pouring tea into a cup> could both be instances of the more abstract event <filling a container>.

Future work could explore how to **learn a taxonomy of events**. Qualitatively, it can be observed that GATELORD tends to encode similar dynamics within the same latent state dimensions.^{8.4} Thus, one could potentially learn more abstract event categories from suitably clustering the learned event codes. Potentially, further regularization of the latent states is needed to foster encoding similar dynamics proximally in the latent space.

THICK world models (Chap. 7), developed a two-level (partonomic) hierarchy of world models. THICK, the algorithm for learning model-based hierarchies, is defined level-wise. It could in principle generalize to arbitrary levels of abstraction by adding new levels whenever the latent state changes of the currently highest level.

^{8.4}Note, that latent state similarity was not quantified. However, examples illustrate the systematics of GATELORD’s latent updates. For example in Fetch Pick&Place, one model (one seed) updates the same dimensions at certain event boundaries, e.g. when grasping an object (Fig. C.15). Similarly, at the ends of two different events with similar dynamics (linearly moving to a goal or object), the same latent state dimensions are updated (cf. reaching and pointing in Fig. 6.4).

However, it is questionable whether adding higher levels in this way is useful for model-based planning. Each added level comes with the high computational cost of training a new network. Additionally, for complex agents facing multiple domains, it seems unrealistic to globally define the granularity of segmentation.

From a cognitive science perspective, it is also debatable whether biological action representations are strictly hierarchically organized. Botvinick (2007), for example, argues that graded, loosely hierarchical representations may develop from dense connections without an explicit hierarchy. Neuroimaging studies suggest that there may exist gradients of abstraction in the brain (Badre & D’Esposito, 2009; Nee & Brown, 2012; Brunec et al., 2018), i.e. certain axes along which the representations of memories (Brunec et al., 2018) or actions (Badre & D’Esposito, 2009) appear to increase in their level of abstraction. This could also hint at a more continuous scale of abstraction.

Could THICK be extended to **more fluent levels of abstraction**? In practice, the segmentation relied on update gates $g_\phi(\mathbf{x})$ for some inputs \mathbf{x} and learnable parameters ϕ , which were regularized to sparse activity. The regularization was set via a hyperparameter β .^{8.5} To learn continuous abstractions, one could treat β as an adjustable parameter and simultaneously learn segmentations for various values of β . For example, a hypernetwork (Ha et al., 2016) could learn to set the weights ϕ of the gating network conditioned on a continuously sampled β . Subsequently, a high-level model trained on the resulting segmentation could make predictions dependent on β . Thus, for high-level predictions in the resulting hierarchy, β would determine the temporal horizon and could be adjusted on the fly depending on the current task and the available computational resources.

8.3 The Bitter Lesson and Cognition-Inspired AI

In the current age of large language models (LLMs), taking inspiration from cognitive science to improve AI has fallen out of popularity. In his seminal essay, ‘The Bitter Lesson’, Sutton (2019) argued that the biggest lesson of decades of AI research is that **methods that leverage computation will ultimately outperform methods that leverage domain knowledge**. The rise of LLMs largely supports his point: At their core, LLMs are generative language models (Devlin et al., 2018; Radford et al., 2019) that are massively scaled up with respect to model size, dataset size, and duration of training (Brown et al., 2020). Interestingly, by scaling up these models, abilities beyond simple text generation seem to emerge (Wei et al., 2022; Bubeck et al., 2023), e.g. the ability to solve canonical tasks of cognitive psychology reasonably well (Binz & Schulz, 2023; Buschhoff et al., 2023). Is cognition-inspired AI doomed to fail in the long run when

^{8.5}In Chap. 7 this hyperparameter is called β^{sparse} .

“computation is all you need”?

Taking inspiration from biological intelligence has helped AI development before, when the inspired design choices are not literal copies but respect hardware and processing differences in vivo and in silico. For example, as the name suggests, neural networks are inspired by the network of neurons found in our brains (Goodfellow et al., 2016; Schmidhuber, 2022). The main inspiration here is that through the composition of small computational units that learn locally from their interactions with connected units, very complex functions can be learned. However, how artificial neurons in typical deep learning systems learn, i.e. through gradient backpropagation, is much different from neurobiological learning (Bengio et al., 2015b). Similar points can be made about machine learning paradigms, e.g. RL, or deep learning building blocks, e.g. self-attention, which took some inspiration from natural processes.^{8.6} Therefore, we should ask not only which structures aid biological cognition, but also how such structures can be learned so that they scale with computation and data. Or, to quote Sutton (2019), “we want AI agents that can discover like we can, not which contain what we have discovered”.

I believe that the work of this thesis is largely in the vein of the proposal of Sutton (2019). I investigated how event-based temporal abstractions can be discovered without explicit supervision and suitably represented in deep neural networks. Much of the thesis focused on showing that this approach scales well.^{8.7} Hence, this research worked towards giving AI the ability to discover model-based temporal abstractions of sensorimotor data, as humans can discover events from their sensorimotor experiences.

8.4 Integrating Theories on Cognition

The models developed in this thesis attempt to integrate and partially unify approaches from different fields of cognitive science and AI research. Most notably, I took inspiration from research on event cognition (Hommel et al., 2001; Zacks et al., 2007; Butz et al., 2021) to improve sequential decision making, such as model-predictive control and RL.

^{8.6}RL has its historic roots in psychology, dating back to the era of behaviorism, and animal experiments on conditioning and learning by trial and error (Sutton & Barto, 2018). Self-attention (Cheng et al., 2016), one of the main driving forces of Transformers (Vaswani et al., 2017) and LLMs, was developed based on precursors of the attention mechanism (Bahdanau et al., 2015; Graves et al., 2014), which were partially inspired by psychological theories of working memory and attentional focus (Graves et al., 2014). However, neither RL nor attention attempt to exactly replicate their natural counterparts.

^{8.7}For example, it was demonstrated that GATELORD’s compression of time series into latent states scales from toy environments to realistic physics simulations (Chap. 5), from scripted to RL-generated sequences (Chap. 6), and from low-dimensional position-based inputs (Chap. 6) to high-dimensional pixel-based inputs (Chap. 7). Similarly, the modeling of goal-anticipatory gaze scaled from predefined scripted events (Chap. 4) to self-segmented events in more realistic simulations (Chap. 6).

Beyond that, ideas from predictive processing (Friston et al., 2006; Hohwy, 2013; Clark, 2015), active inference (Friston et al., 2015; Sajid et al., 2021a; Parr et al., 2022), and causality research (Schölkopf et al., 2021; Schölkopf, 2019) were integrated. Combining different theories can create interesting synergies, as demonstrated by CAPRI (Chap. 4): By applying active inference to a hierarchical model that mirrors the structure of events and event boundaries, various findings of developmental research (Kanakogi & Itakura, 2011; Adam & Elsner, 2020) could be modeled without any additional assumptions.

In this section, I discuss further unification options and the relation to other theories in cognitive science. First, I discuss whether the main approach of this thesis, to some extent, can unify the terminology of actions, events, and context (Sec. 8.4.1). Next, I discuss how the presented models relate to more general theories of resourceful cognition (Sec. 8.4.2).

8.4.1 Action, Event or Context?

In this work I used the terms context, event, and action somewhat interchangeably.^{8.8} To show that this is not due to sloppy terminology, imagine a breakfast scene, where a human subject is preparing a cup of tea and a group of scientists study the neural and cognitive representations involved when performing or perceiving this activity. How would they call the segments that make up the activity, e.g. <grabbing the teapot> or <pouring tea into a cup>? Arguably, the everyday notion for such a segment is an *action*. Cognitive scientists or psychologists who investigate goal-directed behavior would probably also refer to this as an *action*, e.g. Cooper & Shallice (2000). Some researchers in AI may hesitate to call this an action, because in sequential decision making, *action* typically refers to the elementary actions of the underlying decision process (see Sec. 2.2). For cognitive psychologists studying the segmentation of perceptual activity, for example, from an observational perspective, the more common notion for a segment of activity is an *event*, e.g. Zacks (2010). To make matters even more complicated, neuroscientists studying brain activity might refer to a stable segment as the *context*, e.g. Shahnazian et al. (2022).

So what is the relation between actions, events, and context? Do these terms refer to different representations that encode redundant content? Or does each notion refer to a different aspect of the same underlying representation?

The main framework of this thesis, THICK MDP (Chap. 3), argues for the latter. In THICK MDPs ongoing activity and latent information is compressed into **temporally**

^{8.8}For example, in Chap. 7, we argue that the *context* \mathbf{c}_t of the lower level of a THICK world model encodes *events*. At the higher level, embeddings \mathbf{A}_t of low-level codes \mathbf{c}_t are referred to as high-level *actions*.

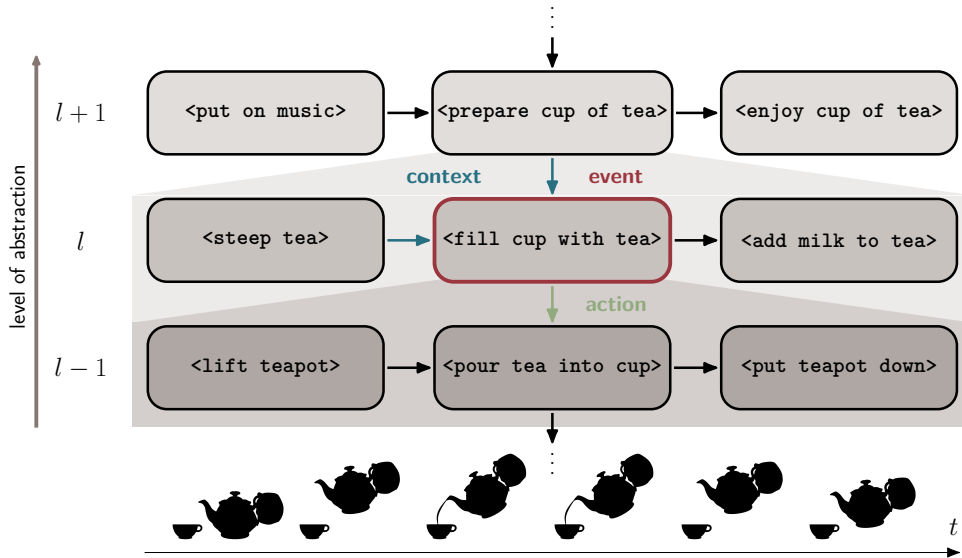


Figure 8.2: Actions, events, and context in THICK MDPs are all encoded as latent states \mathbf{c}_t^l . However, at a certain level of abstraction l , the notions might refer to different codes. For example, an *event* (red) describes the ongoing dynamics encoded in \mathbf{c}_t^l at the abstraction level l , e.g. that an agent fills a cup with tea. The *context* (blue) is past and top-down information conditioning processing at level l , e.g. higher-level information \mathbf{c}_t^{l+1} that the agent wants to prepare a cup of tea. The *action* (green) is the next desired state \mathbf{c}_t^{l-1} set for the lower level $l-1$, e.g. to pour tea into a cup.

persistent codes \mathbf{c}_t . Context, events, and (higher-level) actions are all encoded as such codes \mathbf{c}_t . However, at a certain level l different terms may refer to different aspects of these stable codes, illustrated in Fig. 8.2. For example, at a certain level l , *context* could refer to past and top-down information \mathbf{c}_t^{l+1} from higher levels that conditions the processing. In contrast, *events* may refer to ongoing predicted dynamics at the level l encoded within \mathbf{c}_t^l . For example, within the context `<prepare a cup of tea>`, an event may be `<fill a cup with tea>`. During planning, the level l provides top-down goals for the next lower level $l-1$, in the form of a desired and potentially achievable next code \mathbf{c}_{t+1}^{l-1} . This can be seen as the *action* at this level of processing. For example, actions during a `<fill a cup with tea>`-event might be `<grasp the teapot>`, `<pour tea into the cup>` and `<put teapot down>`. Note that in THICK MDPs the notions are level-dependent. At level $l+1$, for example, the event \mathbf{c}_t^{l+1} may be `<prepare a cup of tea>` in the context of `<prepare breakfast>` and an elementary action might be `<fill a cup with tea>`.

Thus, THICK MDPs blurs the line between context, events, and action representations. Similar propositions have been made before. [Hommel et al. \(2001\)](#) emphasized that

event codes relate actions to perceptions. This strong connection between events and actions has received increasing attention over the last decades (Elsner & Hommel, 2001; Butz, 2016; Radvansky & Zacks, 2014; Cooper, 2021; Kuperberg, 2021; Elsner & Adam, 2021). Furthermore, contextual shifts have been associated with event boundaries when studying episodic memory (DuBrow & Davachi, 2013; Zheng et al., 2022). Finally, in cognitive models, the term *context* is commonly used to refer to the encodings of events or event-like dynamics (Butz et al., 2019; Humaidan et al., 2020; Heald et al., 2021, 2023).

8.4.2 Resourceful Cognition

While many design choices for the models presented were primarily motivated by theories of event cognition, the proposed models can also be understood in light of **resourceful cognition**. To account for the limitations of the human brain, computational models of cognition need to consider processing constraints. From a neuroscience perspective, *efficient coding* proposes that sensory processing attempts to maximize performance under the constraints of its processing capacities (Barlow et al., 1961; Sims, 2018). In cognitive modeling, theories of *bounded rationality* or *resource rationality* (Simon, 1997; Gershman et al., 2015; Lieder & Griffiths, 2020; Bhui et al., 2021) claim that humans make rational decisions subject to cognitive resource constraints. Cognitive resources typically refer to the amount of time and computation the brain spends on mental operations (Lieder & Griffiths, 2020), such as memorizing, mental simulation, or focusing attention. Similarly, the concept of *mental effort* has been proposed (Kahneman, 1973; Shenhav et al., 2017). Mental effort has been formalized as a cost to update previous beliefs about the world (Zénon et al., 2019), for example, when switching between tasks (Butz, 2022).

How does resourceful cognition relate to the methods developed in this thesis? GATELORD (Chap. 5) is trained to minimize a task-based loss while simultaneously minimizing the number of latent state updates. If updating the latent state is seen as an effortful or resource-costly operation, similar to adding an item to working memory (Lieder & Griffiths, 2020) or changing a prior belief (Zénon et al., 2019), then GATELORD aims at maximizing task-based performance subject to resource constraints on latent updates. Event boundary prediction of CAPRI (Chap. 4) or high-level predictions of the Skip Network (Chap. 6) or THICK world models (Chap. 7) can also be viewed as means of saving computational resources. Let us assume that a high-level prediction is as computationally costly as a low-level prediction. Then the system can save computational resources for long-horizon prediction by querying the high-level model *once* as opposed to running open-loop simulations of the low-level for multiple iterations. Seeing that principles inspired by event cognition can also be explained through the lens of resourceful cognition suggests that the associated theories could be unified even further, as has been partially

done by Butz (2022).

8.5 Outlook

Future work could build upon the methods developed in this thesis to further advance the problem-solving skills of artificial agents. The systems presented in this thesis generalize better across various factors, e.g. spurious temporal correlations or training schemes. In addition to that, they can plan their behavior hierarchically. Further scaling and extending these mechanisms (e.g. as described in Sec. 8.2), could give artificial agents the ability to flexibly reason about their experience on different time scales and levels of abstraction. Planning on longer time scales could allow the agent to learn more complex behavior by focusing on long-horizon outcomes and automating the necessary low-level substeps (Schmidhuber, 1992; LeCun, 2022). Adding further levels of abstraction could allow problem solvers to generalize across irrelevant details. This could make their behavior more robust to changes in their environment, their body, or task specifics, which is a crucial step toward open-ended or continual learning (Doncieux et al., 2018; Khetarpal et al., 2022) and zero-shot skill transfer (Kirk et al., 2023).

Moreover, this thesis presented tools that enable artificial agents to encode sensorimotor dynamics into stable, discrete, latent state codes. Different theories of cognitive science propose that such event encodings may give rise to more symbolic concepts and structures in language (Knott, 2012; Gärdenfors, 2014; Butz, 2016). Future work could explore how the latent states discovered by the presented systems could be linked to language. This could allow embodied agents to ground high-level symbolic representations in their sensorimotor experience, a long-standing challenge in cognitive science and AI research (Harnad, 1990; Barsalou, 2008; Butz & Kutter, 2017; Taniguchi et al., 2019; Lake et al., 2019; Greff et al., 2020; LeCun, 2022; Lin et al., 2023). The grounding of language in cognitively plausible representations would be a huge advancement for human-AI alignment. Not only could a user more easily provide instructions to an AI system, but also the system could better explain its decisions by directly expressing them in words, making the overall process much more transparent. When the artificial system learns language functionally in a way similar to that of humans, the risk of miscommunication could also be somewhat mitigated. Alignment of AI with humans is a pressing challenge as AI becomes more and more inter-webbed in our daily lives. I hope that my research can provide a step towards this direction.

Lastly, the outlined work could also pave the way for computational cognitive modeling in more elaborate settings. Events, episodes, or other forms of temporal abstraction of experience are at the heart of many cognitive models (Radvansky & Zacks, 2014). However, including them in computational models requires knowledge about event segmentation in

the to-be-modeled data. The methods developed in this thesis provide a practical deep learning implementation for event segmentation without direct supervision and can be applied to high-dimensional data, such as videos. Thus, this work offers a bridge between models of higher-level cognition and high-dimensional unprocessed stimuli, by grounding events and model-based abstractions directly in the low-level data.

In summary, the outlined research could further advance AI development and cognitive science research. On the one hand, the presented work could lead towards further advancing AI systems by providing practical methods that artificial agents can use to hierarchically represent their experience and plan based on these representations. On the other hand, my research could contribute further to a better understanding of human cognition by modeling human behavior and providing tools that may be useful in other computational cognitive models. Thus, I hope that the research of this thesis provides a small step towards a better understanding of what constitutes intelligence, both artificial and natural.



Background and Approach: Supplementary Material

A.1 Hierarchical Planning and the Cognitive Revolution

[Tolman \(1948\)](#) was a central critic of behaviorist theories. He demonstrated that rats can find rewards in a maze faster when they have visited the maze before, even if their previous visits had not been rewarded ([Tolman & Honzik, 1930](#)). These findings suggest that rats form representations that allow them to plan their behavior prospectively toward a certain outcome ([O'Doherty et al., 2017](#)). Along similar lines, [Lashley \(1951\)](#) challenged the view of motor control as reflex chains in his famous seminal work 'on the problem of serial order in behavior'. [Lashley](#) argued that all behavior, from spoken language to reaching and grasping movements, is hierarchically organized. [Lashley](#) already provided a large amount of evidence for his hypotheses, which have been expanded since then (see [Rosenbaum et al. \(2007\)](#) for a review), and his theory in large part stands the test of time ([Fitch & Martins, 2014](#)). A few years later, [Miller et al. \(1960\)](#) proposed the TOTE model, a highly influential computational model for cognitive action control. In TOTE, actions are selected based on an iterative, hierarchical feedback loop. Here, a desired anticipated state is compared to the current sensory state to select the next action. These ideas sparked the development of numerous other hierarchical models of behavior (see [Botvinick, 2008](#) for a review).

A.2 Details on Sequential Decision Making

A.2.1 Bellman Equation

Equation 2.2 contains the expectation $\mathbb{E}_\pi[\cdot]$ over a policy π . This denotes the expectation of a random variable, e.g. a state \mathbf{s}_k or action \mathbf{a}_k , assuming the agent follows the policy π (Sutton & Barto, 2018). We can directly express this expectation for G using a recursive definition with

$$G(\mathbf{s}_t, \pi) = \sum_{\mathbf{a}_t \in \mathcal{A}} \pi(\mathbf{a}_t | \mathbf{s}_t) \sum_{\mathbf{s}_{t+1} \in \mathcal{S}} T(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t) R(r_t | \mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t) + \gamma G(\mathbf{s}_{t+1}, \pi). \quad (\text{A.1})$$

This equation is known as a *Bellman equation*, based on Bellman (1957), and it relates the value of a state to all of its successor states (Sutton & Barto, 2018).

A.2.2 Discount Factor

An important effect of the discount factor γ is that when $\gamma < 1$ and all rewards r_t are bounded, the sum over infinite elements in Eq. 2.2 has a finite value (Sutton & Barto, 2018). For example, if the agent would receive a constant reward r at every time step, Eq. 2.2 forms a geometric series, that can be computed in closed form as

$$\sum_{k=t}^K \gamma^k r = \frac{r}{1 - \gamma}. \quad (\text{A.2})$$

From this we can derive a general upper bound for G . If r^{\max} is the maximum reward possible then

$$G(\mathbf{s}_t, \pi) \leq \frac{r^{\max}}{1 - \gamma}, \forall \gamma < 1. \quad (\text{A.3})$$

In practice, the discount factor is typically chosen as $\gamma \in [0.95, 0.99]$. This keeps G bounded, while still taking future rewards into account for rather long horizons.

A.2.3 Differences in Formulations of the Free Energy Principle

There exists a lot of literature on the Free Energy Principle (FEP) with varying notation and simplifications. In this section, notable differences are discussed between the review of the FEP in Sec. 2.2.4 and other descriptions in the literature.

- **states:** Sometimes a distinction is made between the state of the environment and hidden causes of the sensory signal (Friston, 2010; Friston et al., 2011). Often (e.g. in Friston et al., 2015, 2016; Sajid et al., 2021a) this is not the case.

- **actions:** In some formulations of the FEP the true actions \mathbf{a}_t of the POMDP are distinct from internal action representations of the agent (e.g. Friston et al., 2015). In other variants, this distinction is ignored (e.g. Friston et al., 2011; Sajid et al., 2021a; Schwöbel et al., 2018). For simplicity, we follow the latter.
- **generative model:** We define the generative model $f_\phi(\mathbf{o}_t, \mathbf{s}_t \mid \mathbf{o}_{1:t-1}, \mathbf{a}_{1:t})$ as a predictive distribution that generates states \mathbf{s}_t and observations \mathbf{o}_t based on previous sensorimotor experiences, done similarly in Friston et al. (2011, 2010); Sajid et al. (2021a). In other formulations of the FEP, the generative model f_ϕ is a joint distribution over states, observations, and actions, e.g. Friston et al. (2015, 2016); Schwöbel et al. (2018).
- **recognition density:** In most versions of the FEP, the recognition density q_θ is conditioned on past actions $\mathbf{a}_{1:t}$ or on a policy π (e.g. Friston et al., 2015, 2016; Sajid et al., 2021a; Schwöbel et al., 2018). In other variants, there is no explicit conditioning on actions (e.g. Friston et al., 2010, 2011). Rarely is the recognition density conditioned on past observations. However, this is in principle also possible: The recognition density could be defined as $q_\theta(\mathbf{s}_{t+1} \mid \mathbf{a}_{1:t-1}, \mathbf{o}_{1:t-1})$ or alternatively, by using past state belief $q_\theta(\mathbf{s}_{t+1} \mid \mathbf{a}_t, \mathbf{s}_{t-1})$. In fact, the latter of the formulations is how variational free energy is implemented as a model loss in recent model-based reinforcement learning approaches (Hafner et al., 2019a, 2020a).
- **prior preferences:** In some active inference formulations, the prior over desired states is expressed as part of the generative model f_ϕ (e.g. in Friston et al., 2015). In our formulation, we split preferences and predictions into two separate components, g and f_θ , respectively, to avoid convoluted terminology. However, without loss of generality, the prior preferences could also be a prediction of the generative model.

A.3 THICK MDP details

A.3.1 Semi-Markovian Context Changes

Let $\mathcal{M}^l = (\mathcal{S}^l, \mathcal{A}^l, T^l, R^l)$ be the Markov Decision Process at a certain level l transitioning at time t^l . The states $\mathbf{s}_{t^l}^l \in \mathcal{S}^l$ are composed of event-encoding contexts $\mathbf{c}_{t^l}^l$ and residual state information $\mathbf{z}_{t^l}^l$, i.e. $\mathbf{s}_{t^l}^l = (\mathbf{c}_{t^l}^l, \mathbf{z}_{t^l}^l)$. I want to allow for context changes that occur sparsely in time t^l . However, these changes in context should be systematic and fully predictable from state and action information. How can we formalize this?

Let Π be the set of all possible policies π in \mathcal{M}^l . Furthermore, let us denote $\tau(t^l)$ as the next time step after t^l for which the context changes, i.e. $\mathbf{c}_{\tau(t^l)}^l \neq \mathbf{c}_{t^l}^l$ (see Eq. 3.1). For

every MDP \mathcal{M}^l of our framework, we require the following equality to hold:

$$T^l(\mathbf{c}_{\tau(t^l)}^l \mid \mathbf{s}_{t^l}^l, \pi) = T^l(\mathbf{c}_{\tau(t^l)}^l \mid \mathbf{s}_{1:t^l}^l, \pi), \forall \pi \in \Pi, \mathbf{s}_{t^l}^l \in \mathcal{S}. \quad (\text{A.4})$$

This means that the transitions T^l in contexts $\mathbf{c}_{t^l}^l$, conditioned on an arbitrary policy $\pi \in \Pi$ and state $\mathbf{s}_{t^l}^l \in \mathcal{S}$, need to fulfill the Markov property (cf. Eq. 2.1). In other words, these context transitions are semi-Markovian on the time scale t^l of the underlying Markov model \mathcal{M}^l (see Sec. 2.2.2 for details on Semi-MDPs).

What does this entail? This means that given a state and a policy, we are able to fully predict the probabilities of the next context transitions. This restricts what kind of information states $\mathbf{s}_{t^l}^l$ and contexts $\mathbf{c}_{t^l}^l$ can encode. For example, systematic changes based on interacting with the state space can be encoded in $\mathbf{c}_{t^l}^l$, such as $\mathbf{c}_{t^l}^l$ representing temporally extended sequences, e.g. `<reach>`, or parts of the state space, e.g. `<being in the kitchen>`. However, unpredictable context transition probabilities are not allowed.

A.3.2 Available Actions

In THICK MDPs, I equate actions \mathbf{a}^l of a level l with event contexts \mathbf{c}^{l-1} of a lower level $l - 1$. However, not every event is possible in every situation. How can we define the available actions $\mathcal{A}(\mathbf{s}^l)$ for a certain state \mathbf{s}^l ?

First, we define the set $\mathcal{S}^{l-1}(\mathbf{s}^l)$ as the states that can occur on the lower level $l - 1$ at the same time as a high-level state \mathbf{s}^l . Now we can define $\mathcal{A}(\mathbf{s}^l)$ with

$$\mathcal{A}(\mathbf{s}^l) \doteq \{\mathbf{c}^{l-1} \mid \exists \mathbf{s}^{l-1} \in \mathcal{S}^{l-1}(\mathbf{s}^l), \mathbf{a}^{l-1} \in \mathcal{A}^{l-1}(\mathbf{s}^{l-1}) \text{ s.t. } T^{l-1}(\mathbf{c}^{l-1} \mid \mathbf{s}^{l-1}, \mathbf{a}^{l-1}) > 0\}. \quad (\text{A.5})$$

Thus, an event \mathbf{c}^{l-1} is only an available action if there exists a transition to a state with this event-encoding context on the level $l - 1$.

B

CAPRI: Supplementary Material^{B.1}

B.1 Relation to THICK MDPs

In Chap. 3 I introduced THICK MDPs as a formal framework for hierarchical decision making inspired by various findings on event cognition. The main preposition of THICK MDPs is that hierarchical levels interact bidirectionally via temporal persistent event contexts \mathbf{c}_t^l . The state of level $l + 1$ changes on the time scale of context changes at level l . During planning, a high level $l + 1$ guides the lower level l by proposing the desired next contexts \mathbf{c}^l .

CAPRI can be understood as a simplified two-leveled THICK MDP. Intuitively, CAPRI’s low level ($l = 1$) predicts the next observations via its event dynamics models. At the same time, its high level ($l = 2$) predicts upcoming event boundaries, i.e. situations that change low-level events codes or contexts. Applying active inference to both levels attempts to minimize the uncertainty about both types of predictions. In the remainder of this section, I demonstrate how the action selection strategy of CAPRI can be derived directly from THICK MDPs with some simplifications.

In THICK MDPs, for every level l the full state $\mathbf{s}_t^l = (\mathbf{c}_t^l, \mathbf{y}_t^l)$ is composed of a temporally persistent part, the context code \mathbf{c}_t^l , and a continuously changing residual part \mathbf{y}_t^l . Let us

^{B.1}This chapter is based on the supplementary material of:
Gumbsch, C., Adam, M., Elsner, B., & Butz, M. V. (2021). Emergent Goal-Anticipatory Gaze in Infants via Event-Predictive Learning and Inference. *Cognitive Science*, 45:e13016.

assume that in CAPRI the residual state information \mathbf{y}_t^l is some function of the observation \mathbf{o}_t . Thus, for simplicity, we set $\mathbf{y}_t^l = \mathbf{o}_t$ for every level l . The context \mathbf{c}_t^l needs to capture stable events. Thus, we set the context code of the lowest level $l = 1$ to the currently active event $\mathbf{c}_t^1 = e_t$. The second level $l = 2$, becomes active when the lower level context \mathbf{c}_t^1 changes, i.e. at event boundaries. Let us assume that the environment of CAPRI is simple enough so that for level 2 no context changes of \mathbf{c}_t^2 occur. Thus, we can omit the context here and set $\mathbf{c}_t^2 = \mathbf{0}$.

How are the actions then inferred? We start at the highest level $l = 2$ with

$$J^2(\pi^i, K) = \mathbb{E}_{\pi^i} \sum_{t=t^2}^K R^2(r_t^2 \mid \mathbf{s}_t^2, \mathbf{a}_{t+1}^2, \mathbf{s}_{t+1}^2). \quad (\text{B.1})$$

for a policy π^i and planning horizon K . We can replace actions at level $l = 2$ with lower level contexts $\mathbf{a}_t^2 = \mathbf{c}_t^1 = e_t$. Additionally, we can rewrite states as $\mathbf{s}_t^2 = (\mathbf{c}_t^2, \mathbf{y}_t^2) = (\mathbf{0}, \mathbf{o}_t) = \mathbf{o}_t$. We can do this because of the simplifications we introduced earlier, i.e. setting residual states to observations ($\mathbf{y}_t^l = \mathbf{o}_t$) and assumed that there are no event contexts on level 2 ($\mathbf{c}_t^2 = \mathbf{0}$). As a result, we get

$$J^2(\pi^i, K)' = \mathbb{E}_{\pi^i} \sum_{t=t^2}^K R^2(r_t^2 \mid \mathbf{o}_t, e_{t+1}, \mathbf{o}_{t+1}). \quad (\text{B.2})$$

We plan by means of active inference (Parr et al., 2022). Thus, we replace our reward function R^2 with expected free energy (EFE) focusing on minimizing uncertainty as in Eq. 4.6, which gives us

$$J^2(\pi^i, K)'' = \mathbb{E}_{\pi^i} \frac{1}{K} \sum_{t=t^2}^K \mathcal{H}[f^2(\mathbf{o}_{t+1} \mid e_{t+1}, \mathbf{o}_t)], \quad (\text{B.3})$$

with f^2 the model of level $l = 2$. What is the model f^2 ? As outlined above, level $l = 2$ should predict context changes, i.e. event boundaries. Thus, we get

$$J^2(\pi^i, K)''' = \mathbb{E}_{\pi^i} \frac{1}{K} \sum_{t=t^2}^K \mathcal{H}[P_{e_{t+1}}^{\text{start}}(\mathbf{o}_{t+1} \mid \pi^i) P_{e_t}^{\text{end}}(\mathbf{o}_{t+1} \mid \mathbf{o}_t, \pi^i)], \quad (\text{B.4})$$

by replacing the model $f^2(\mathbf{o}_{t+1} \mid e_{t+1}, \mathbf{o}_t)$ with the event boundary models that model the transition from the current event e_t to the next event e_{t+1} . Additionally, we condition the models on the policy π^i , since all event schmemata are policy-conditioned.

The policy on level $l = 1$ is inferred based on Eq. 3.5 to maximize external rewards, according to the reward function R^1 , while minimally diverging from a high-level plan.

For simplicity, we enforce minimal divergence from the high level plan by simultaneously optimizing J^2 . As a result, we can write

$$J^1(\pi^i, K) = \mathbb{E}_{\pi^i} \sum_{t=t^1}^K R^1(r_t^1 | \mathbf{s}_t^1, \mathbf{a}_t^1, \mathbf{s}_{t+1}^1) + J^2(\pi^i, K). \quad (\text{B.5})$$

We can rewrite $\mathbf{s}_t^1 = (\mathbf{e}_t, \mathbf{o}_t)$. Additionally, as before we replace the reward function R^1 with EFE from Eq. 4.6 to get

$$J^1(\pi^i, K)' = \mathbb{E}_{\pi^i} \frac{1}{K} \sum_{t=t^1}^K \mathcal{H}[f^1(\mathbf{o}_{t+1}, e_{t+1} | e_t, \mathbf{o}_t, \mathbf{a}_t)] + J^2(\pi^i, K) \quad (\text{B.6})$$

What is the generative model of level $l = 1$? As outlined earlier, level 1 predicts event dynamics. This is why we simply replace f^1 with the event dynamics models and get

$$J^1(\pi^i, K)'' = \mathbb{E}_{\pi^i} \frac{1}{K} \sum_{t=t^1}^K \mathcal{H}[P_{e_t}^{\text{event}}(\mathbf{o}_{t+1} | \mathbf{o}_t, \pi^i)] + J^2(\pi^i, K). \quad (\text{B.7})$$

Note that we replaced \mathbf{a}_t with π^i since our event dynamics model directly takes a policy as an argument. Additionally, we omit the likelihood of event e_{t+1} because this is not predicted by the event dynamics models. If we insert J^2 from Eq. B.4, we end up with

$$J^1(\pi^i, K)''' = \mathbb{E}_{\pi^i} \frac{1}{K} \sum_{t=t^1}^K \mathcal{H}[P_{e_t}^{\text{event}}(\mathbf{o}_{t+1} | \mathbf{o}_t, \pi^i)] \quad (\text{B.8})$$

$$+ \frac{1}{K} \sum_{\tau=t^2}^K \mathcal{H}[P_{e_{\tau+1}}^{\text{start}}(\mathbf{o}_{\tau+1} | \pi^i) P_{e_{\tau}}^{\text{end}}(\mathbf{o}_{\tau+1} | \mathbf{o}_{\tau}, \pi^i)]. \quad (\text{B.9})$$

This is our overall objective for policy inference. When we set $K = 1$ and compute the expectation based on the probabilities of past events $P(e_t | \mathbf{o}_{1:t}, \pi_{1:t-1})$, we arrive at Eq. 4.7.

B.2 Implementation Details

B.2.1 Pseudocode

Here we provide pseudocode of CAPRI. A Python implementation can be found at <https://github.com/CognitiveModeling/CAPRI>.

Algorithm 1 Sensorimotor loop of CAPRI

```
1: procedure CAPRI
2:   while simulation is running do
3:     start new episode  $E$ 
4:      $t \leftarrow 1$ 
5:     determine whether training or testing
6:     initialize  $\pi_1$ 
7:     while  $E$  is not finished do
8:       receive  $\mathbf{o}_t$ 
9:       if training then
10:        receive currently active event  $e_t$ 
11:        set  $P(e_t^i | \mathbf{o}_{1:t}, \pi_{1:t}) = 1$  for  $e^i = e_t$ 
12:        set  $P(e_t^j | \mathbf{o}_{1:t}, \pi_{1:t}) = 0 \forall e^j \neq e_t$ 
13:        update  $P_{e^i}^{\text{start}}$ ,  $P_{e^i}^{\text{event}}$ ,  $P_{e^i}^{\text{end}}$  (see Sec. 4.3.1)
14:         $\pi_t \leftarrow \pi_{t-1}$ 
15:       else
16:        compute  $P(e_t^i | \mathbf{o}_{1:t}, \pi_{1:t}) \forall e^i$  (Eq. 4.1)
17:        compute  $\widehat{FE}(\pi) \forall \pi$  (Eq. 4.7)
18:         $\pi_t \leftarrow \arg \min_{\pi} \widehat{FE}(\pi)$ 
19:       execute  $\pi_t$ 
20:        $t \leftarrow t + 1$ 
```

B.2.2 Gaussian Neural Networks

Each event schema aims to model a distribution $P(\mathbf{o}_t \mid \mathbf{x})$ over observations at time t and \mathbf{x} some conditional variables. Because we assume that this distribution can locally be modeled as a multivariate Gaussian distribution dependent on \mathbf{x} . Thus, in our implementation, each likelihood distribution $P(\mathbf{o}_t \mid \mathbf{x})$ is parameterized by a separate neural network that outputs a mean $\boldsymbol{\mu}$ and diagonal covariance matrix $\boldsymbol{\Sigma}$ from inputs \mathbf{x} (Bishop, 2006). We use linear activations to predict the mean $\boldsymbol{\mu}$ and employ exponential linear units (ELU) shifted by 1 for $\boldsymbol{\Sigma}$, to ensure that $\boldsymbol{\Sigma} > 0$.

The networks can be trained using the negative log likelihood (NLL) of observations \mathbf{o}_t under a parameterized Gaussian as its loss function. We modify the NLL loss slightly to avoid problems that could arise from gradient-based training :

$$\mathcal{L}(\boldsymbol{\phi}) = \tanh \left(-\log \mathcal{N}(\mathbf{o}_t \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) \cdot b \right), \quad (\text{B.10})$$

with $\boldsymbol{\phi}$ the parameters of the neural network and b a scaling hyperparameter. We squash the loss by a tanh-function avoids that a very low likelihood (≈ 0) lead to exploding weights in the beginning of training. The empirically chosen constant factor $b = 0.01$ scales the likelihood so that little relevant information is lost in the squashing process. We use single-layered networks predicting mean and covariance matrix separately, seeing that linear dependencies are sufficient in our simulations. To speed up the training of the networks while avoiding local overfitting, we use a sampling buffer with a large capacity (1000 samples) for each network and draw two additional training samples per update step. With this balanced training scheme, we could train the networks with a rather high learning rate of $\eta = 0.1$ using stochastic gradient descent. The weights of the neural networks were randomly initialized following a zero-centered Gaussian distribution with a standard deviation of 0.1.

B.2.3 Training details

During training, the system learns its generative Gaussian neural networks in a supervised manner. Each training episode E can consist of one or more events, i.e. $E = (e^i, e^j, \dots, e^z)$. When an event e^i starts at time t^s , the starting condition $P_{e^i}^{\text{start}}$ is updated using π_{t^s-1} as input and \mathbf{o}_{t^s} as target. At each time step t during the event e^i , the event dynamics model $P_{e^i}^{\text{event}}$ is updated using \mathbf{o}_{t-1} and π_{t-1} as inputs and \mathbf{o}_t as target. When an event e^i ends at time step t^e , the end condition $P_{e^i}^{\text{end}}$ is updated $t^e - t^s$ times. $P_{e^i}^{\text{end}}$ is updated using \mathbf{o}_{t^e} as target and $\pi(t^e - 1)$ and \mathbf{o}_{t^e-l} as inputs with $l \in [1, \dots, t^e - t^s]$. These multiple model updates do not only increase the training data for the end condition, but also allow CAPRI to learn to generally predict how an event ends from any observation that had previously occurred during this event.

B.2.4 Derivation of Event Probability Estimation

Here we provide a full derivation of Eq. 4.2:

$$\begin{aligned}
P(e_t^i \mid \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j) &= \frac{P(e_t^i, \mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)}{P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)} && \text{Bayes rule} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j)}{P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)} && \text{Bayes rule} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j) \cdot P(e_{t-1}^j)}{P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j)} \\
&\text{factoring the joint likelihood } P(\mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \text{ in numerator} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j) \cdot P(e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j, e_t^h)} \\
&\text{marginalization of } P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_{t-1}^j) \text{ over } e^h \text{ in denominator} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j) \cdot P(e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^h, e_{t-1}^j) \cdot P(e_t^h \mid e_{t-1}^j) \cdot P(e_{t-1}^j)} \\
&\text{factoring the joint probability } P(\mathbf{o}_t, \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \text{ in denominator} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^h, e_{t-1}^j) \cdot P(e_t^h \mid e_{t-1}^j)} \\
&\text{canceling } P(e_{t-1}^j) \text{ in numerator and denominator} \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \cdot P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^h, e_{t-1}^j) \cdot P(e_t^h \mid e_{t-1}^j)} \\
&\text{Markov Assumption (Eq. 2.1): } P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_t^h, e_{t-1}^j) = P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_{t-1}^j) \\
&= \frac{P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^i, e_{t-1}^j) \cdot P(e_t^i \mid e_{t-1}^j)}{\sum_{e^h} P(\mathbf{o}_t \mid \mathbf{o}_{t-1}, \pi_{t-1}, e_t^h, e_{t-1}^j) \cdot P(e_t^h \mid e_{t-1}^j)} \\
&\text{canceling } P(\mathbf{o}_{t-1}, \pi_{t-1} \mid e_{t-1}^j) \text{ in numerator and denominator}
\end{aligned}$$

B.3 Simulation Details

Here we provide further details on the simulation environment. In our simulation, the system received an 18-dimensional real-valued observation \mathbf{o} with:

$$\mathbf{o} = [p^{a,x}, p^{a,y}, p^{a,z}, v^{a,x}, v^{a,y}, v^{a,z}, \lambda^a, p^{p,x}, p^{p,y}, p^{p,z}, v^{p,x}, v^{p,y}, v^{p,z}, \lambda^p, p^{p-a,x}, p^{p-a,y}, p^{p-a,z}, \delta^{p-a}], \quad (\text{B.11})$$

with \mathbf{p} positions, \mathbf{v} velocities, λ appearance, and δ distance. Entities are denoted by superscript letters, \mathbf{a} for agent, \mathbf{p} for patient, and $\mathbf{p} - \mathbf{a}$ denoting agent relative to patient. The positions were bound with $p^{x/y} \in [-1, 1]$ and $p^z \in [0, 1]$. The appearance was always $\lambda \in [0, 1]$ and uniformly sampled for all entities except for hands and claws.

During training, the starting positions of the agent and the patient were randomly sampled for each new event sequence, ensuring that agent and patient were sufficiently distant ($\delta^{p-a} > 0.1$). Additionally, the patient started always on the ground, with $p^{p,z} = 0$. For all events involving agent motions, a goal position was either uniformly sampled, for e^{rand} and e^{tran} , or set to the patient position for e^{reach} . Per step the agent decreased the overall distance by a fraction $v \in [0.01, 0.05]$ by linearly moving to the goal. For e^{reach} this means $\mathbf{v}^a = v \cdot \mathbf{p}^{p-a}$. For e^{rand} , we simulated a simplified version of friction where v decreased over time by a fixed factor of 0.97.

To ensure that the test event sequences had the same duration and were comparable between simulations, E^{test} was determined differently. The patient was always placed in the center of the screen, i.e. $\mathbf{p}^p = (0, 0, 0)$. The agent started in the air ($p^{a,z} = 0.6$) and close to one corner of the screen ($p^{a,x/y} \in \{-0.6, 0.6\}$). For reaching, we set $v = 0.1$. The goal of transporting the object was set to $(\pm 0.77, \pm 0.77, 0)$. The direction and velocity of e^{rand} were determined randomly.

The different events ended based on different criteria: All `<standing still>`-events ended after 100 time steps. Randomly directed motion ended when the agent's velocity was approximately zero ($|\mathbf{v}^a| < 0.0005$). Reaching and lifting ended when the distance between the agent and the goal was less than a threshold ($\delta^{p-a} < 0.05$).

The appearance of hand or claw was randomly sampled for each simulation. The appearance of a hand could be $\lambda^{\text{hand}} \in [0, 0.5[$, whereas the appearance of a claw could be $\lambda^{\text{claw}} \in]0.5, 1.0]$. The idea for this is that we do not want to make any assumptions on how differently hands and claws were perceived. In this way we randomize the difference in visual appearance between simulations, and the difference in appearance lies $\in (0, 1]$. Note that setting the shape of a hand agent to $s^a = 0$ and the shape of a claw agent to $s^a = 1$ yielded very similar results in pre-tests for both the inferred event probabilities

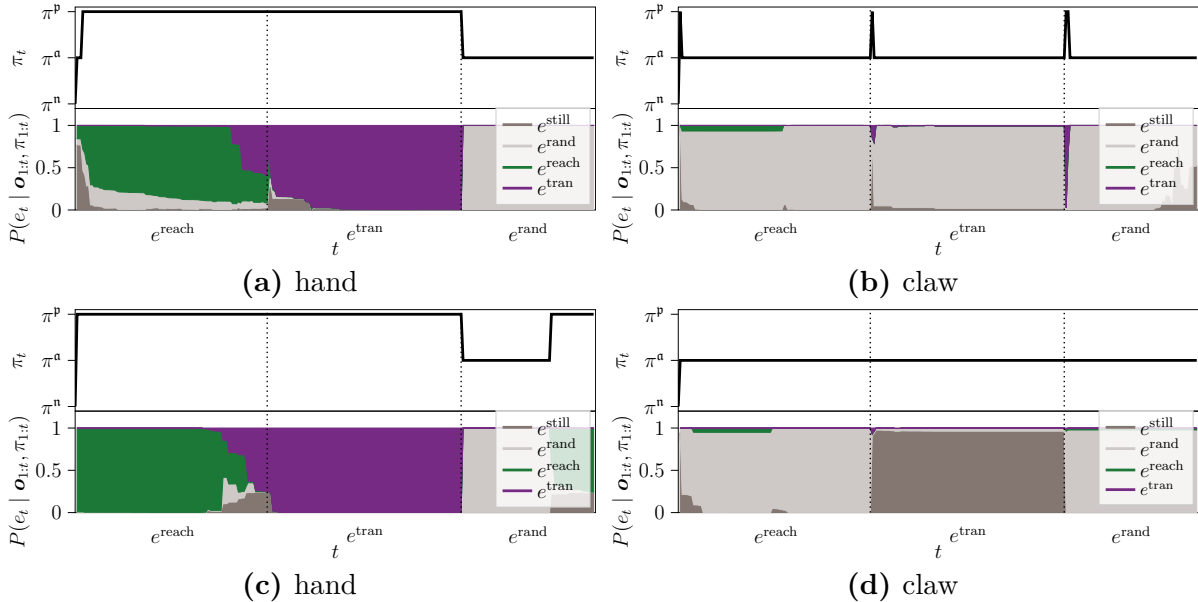


Figure B.1: More exemplary event and policy inference of CAPRI after full training over the course of one event sequence E^{test} for hand (a)&(c) or claw agents (b)&(d). The upper rows show the active policy π_t . The bottom rows show the inferred event probability estimates. Event boundaries are marked by dotted lines.

and the gaze behavior. However, for fixed appearances, there was less variation between simulations.

The policy π was represented by a three-dimensional one-hot encoding. During training for each new event sequence a policy was randomly sampled and the system executed this policy over the whole event sequence. During testing, the system always started with the no fixation policy π^n before inferring its policy.

B.4 Additional Experiments and Analysis

B.4.1 Examples

Figure B.1 provides more exemplary sequences for event and policy inference of CAPRI. While the exact inference patterns vary across random simulations, CAPRI tends to infer the correct events more often when observing hands. Different gaze patterns (top) may develop in different simulations.

B.4.2 Comparison with Dynamics Simulation

Previously, it has been proposed that trajectory-based information is used by infants to estimate then end of an observed movement (Ganglmayer et al., 2019). Similarly, goal-anticipation in infants was modeled before through open-loop forward simulations (Copete et al., 2016). In its classical form, active inference also does not require hierarchical predictions, but instead assumes that expected free energy is minimized over a fixed prediction horizon of low-level environment steps (Friston et al., 2015, 2016). Thus, an alternative explanation for the development of goal-anticipatory gaze behavior in infants, is that, instead of directly predicting the event boundary, infants use low-level forwards simulations over a longer time horizon to predict the future. Could such a non-hierarchical prediction scheme also yield goal-anticipatory gaze behavior when gaze is inferred to minimize predicted uncertainty?

We test this hypothesis by slightly modifying CAPRI so that actions are inferred only based on the predictions of the event dynamics models. For this, we can reformulate the general active inference equation of Eq. 4.6 to

$$\begin{aligned} \text{EFE}''(\pi^i, \mathbf{o}_{1:t}, \pi_{1:t-1}, K) = \\ \sum_{k=1}^K \mathbb{E}_{\mathbf{o}_{t:k} \sim f_\phi} \sum_{e^j} \mathcal{H}[P_{e^j}^{\text{event}}(\mathbf{o}_{t+k+1} | \mathbf{o}_{t+k}, \pi^i)] P(e_{t+k}^j | \mathbf{o}_{1:t+k}, \pi_{1:t}, \pi_{t+1:t+k}^i) \end{aligned} \quad (\text{B.12})$$

where K is a prediction horizon expanding into the future. This corresponds to the uncertainty of event dynamics prediction of Eq. 4.7 over a horizon of K steps.

Computing this expectation in Eq. B.12 in closed form quickly becomes intractable. Thus, we estimate future observation and event probabilities iteratively using Monte Carlo estimates starting with $k = 1$. For each step $t + k$ in the future, an event schema e_{t+k}^j is first sampled based on the latest event probabilities $P(e_{t+k}^j | \mathbf{o}_{1:t+k}, \pi_{1:t}, \pi_{t+1:t+k}^i)$. Next, we predict the following observation \mathbf{o}_{t+k+1} by sampling it according to the corresponding dynamics model of e_{t+k}^j , i.e. $\mathbf{o}_{t+k+1} \sim P_{e^j}^{\text{event}}(\mathbf{o}_{t+k+1} | \mathbf{o}_{t+k}, \pi^i)$. Based on the predicted \mathbf{o}_{t+k+1} the event probabilities are updated for time $t + k + 1$ (see Sec. 4.3.2). This process continues until $k = K$. As before, we select the policy with the lowest EFE'' (Eq. 4.8).

We compare different prediction horizons K ($K \in [1, 5, 10]$), trained with 10 random seeds, to CAPRI. Note that $K = 1$ corresponds to CAPRI without minimization of event boundary uncertainty (Eq. 4.7).

Fig. B.2 shows the time t when all versions of the system first focused on the patient, i.e. activated π^p , within a E^{test} sequence for hands (Fig. B.2a) and claws (Fig. B.2b) over training. Only CAPRI, reliably looked at the patient when observing a reach by hand agents. With long-horizon predictions ($K \geq 5$) and early during training (phases ≤ 5),

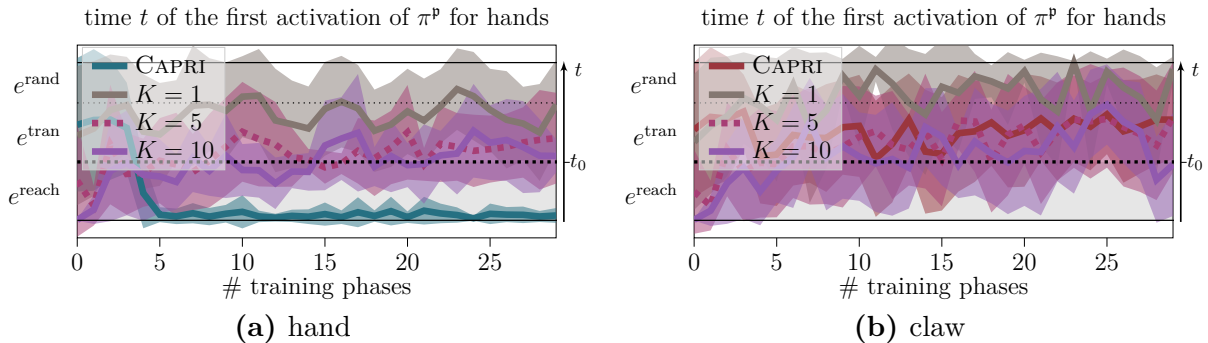


Figure B.2: Effect of prediction horizon: We compare CAPRI to an ablation that predicts the future over K low-level prediction steps. We plot the mean time t during E^{test} when the system first activated π^p , over training phases. The y -axis shows time during one sequence E_{test} with t_0 marking the arrival of the agent at the patient. We compare hand **(a)** and claw agents **(b)**. Shaded areas show the standard deviation.

the system sometimes activated π^p during e^{reach} . However, we mainly attribute this to poorly learned models and the stochasticity of iterative sampling. Although increasing K appeared to result in an earlier gaze at the object, without hierarchical predictions, the system on average did not look at the object before agent-patient contact (t_0).

Thus, if a non-hierarchical prediction scheme is used in CAPRI, goal-anticipatory gaze does not develop. We see this as evidence that hierarchical event-based predictions are necessary to elicit the goal-anticipatory gaze pattern observed in the eye-tracking studies of infants.

B.4.3 Effect of Event Experience

To investigate how the experience of reaching affects the goal-anticipatory gaze of CAPRI we manipulated how often the system encountered the event sequence E^{grasp} during training. We tested five conditions that differ in the percentage of E^{grasp} sequences during training: The system was trained on 0.1%, 1%, 10%, 20% and 50% E^{grasp} sequences of overall 1000 training event sequences. We ran this experiment 10 times per condition with different random seeds. After complete training, we tested gaze inference of the systems for hand or claw-agents on 10 E^{test} sequences each.

Figure B.3 shows the point in time t when CAPRI focused the patient, i.e. activated π^p , for the first time within a E^{test} sequence for different conditions. With more experiences of E^{grasp} the system tended to look at the patient earlier. For 0.1 – 10% experience with E^{grasp} the system selected π^p on average during e^{tran} or at the beginning of e^{rand} regardless

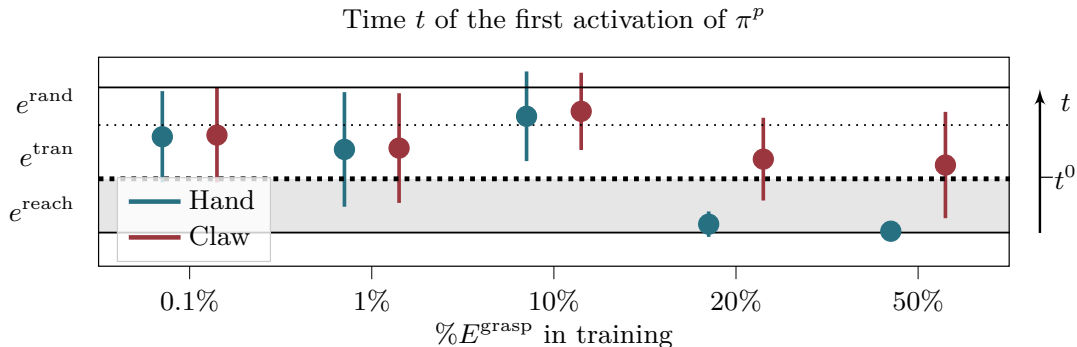


Figure B.3: Effect of reaching experience on gaze for CAPRI when observing hands (blue) or claws (red). We plot the mean time t during E^{test} when CAPRI first activated policy π^p , which corresponds to looking at the patient, over percentage of grasping sequences E^{grasp} during training. The y -axis shows time during one sequence E^{test} with t_0 marking the agent-patient-contact arrival of the agent at the patient. Error bars show the standard deviation. Data points in the gray area mark anticipatory gaze behavior.

of the type of agent. For more than 20% experience with E^{grasp} and when observing a hand agent, CAPRI activated π^p shortly after the onset of the reaching event e^{reach} .

Thus, when CAPRI has little experience with reaching events e^{reach} , the system does not show goal-anticipatory gaze behavior. With increasing experience, the system tends to look at the patient earlier, typically once it was able to identify the reaching event.

These results are in line with experimental findings on the goal anticipation in infants. 4-month-old infants, who have little or no experience grasping themselves, do not show a goal-anticipatory gaze when watching reaching movements (Kanakogi & Itakura, 2011). Instead, they reactively track the moving hand with their gaze. Similarly, our system reactively tracked the hand with little reaching experience. Furthermore, in infants, the goal anticipations for hand agents appear to be positively correlated with grasping experience (Kanakogi & Itakura, 2011). In CAPRI, goal-anticipatory gaze develops with sufficient reaching experience.

B.4.4 Differences in Gaze Shift Timing

Our main findings (Sec. 4.4) demonstrated that CAPRI shows gaze behavior comparable to that of infants. However, there are numerical differences between the gaze timing of CAPRI and the eye tracking results in infants. Namely, infants tended to shift their gaze to the object during the last 10 – 20% of a hand-reaching event (Adam & Elsner, 2020).

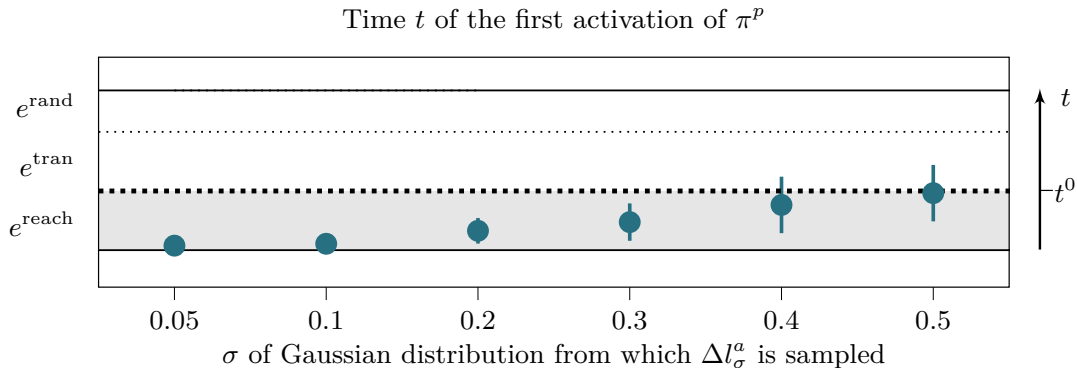


Figure B.4: Effect of hand perception on the gaze behavior in CAPRI when the appearance of the hand is systematically modified. We modified l^a by adding randomly sampled noise Δl_σ^a . We plot the mean time t during an event sequence E^{test} when the system first activated policy π^p , i.e. looking at the patient, over different standard deviations σ . The y -axis shows time during one testing sequence E^{test} with t^0 marking agent-patient-contact. Error bars show the standard deviation.

Our system focused on the object right after the onset of reaching.

We hypothesize that these numerical differences can be attributed to the simplifications of our scenario. An explanation could be that our system experiences E^{test} sequences exactly like the E^{grasp} sequences that had occurred during training, while infants observe reaching movements from other hands than their own and on a screen. Thus, it may be harder for the infant to recognize the hand, and as a result, it might take more time to recognize the reaching event.

To test this assumption, we systematically altered the perceived appearance of the hand agents during testing. We took fully trained systems from our main experiments (Sec. 4.4), and conducted an additional testing phase, each composed of 10 testing sequences E^{test} with hand agents. For each event sequence E^{test} , we altered the perceived appearance of the hand agent (λ^a) by an offset $\Delta \lambda_\sigma^a$. The offset $\Delta \lambda_\sigma^a$ was randomly sampled per sequence from a zero-centered Gaussian distribution with standard deviation σ . We tested five different conditions with $\sigma \in [0.05, 0.1, 0.2, 0.3, 0.4, 0.5]$.

Figure B.4 shows the point in time t when CAPRI first looked at the patient, i.e. activated π^p , during the E^{test} sequence with a hand agent. The x -axis shows different standard deviations σ for the shape modification $\Delta \lambda_\sigma^a$. For $\sigma \leq 0.1$ the system activated π^p immediately after the onset of e^{reach} . With increasing values of σ the system activated π^p later during the reaching event. For $\sigma = 0.5$ the system activated on average π^p

shortly before e^{reach} was concluded. Besides delaying the goal-anticipatory gaze, larger shape modifications resulted in larger variations across simulations.

Thus, our results largely confirm the hypothesis that the earlier gaze in CAPRI can partially be attributed to the similarity of training and testing for our system. This can be understood when we focus on the event inference of CAPRI: For an agent with a less ‘hand-like’ appearance, the system requires on average more observations to correctly infer that reaching is being observed. Later recognition of a reaching event results in a later goal-anticipatory gaze.



GATELORD: Supplementary Material^{C.1}

C.1 Relation to other RNNs

In Sec. 5.2 we set out to create an RNN f_ϕ that maintains piecewise constant latent states over time. This led us to the conclusion that a simple approach to implement this is by employing an internal gating function Λ that controls the latent state update (e.g. as in Eq. 5.4). The gating function Λ can be binarized using the Heaviside step function, and a sparse gating can be incentivized using the loss function described in Eq. 5.5.

GRUs (Chung et al., 2014) and LSTMs (Hochreiter & Schmidhuber, 1997) both use internal gates with the sigmoid activation function σ to control the update of their latent state \mathbf{h}_t . GRUs update \mathbf{h}_t with

$$\mathbf{h}_t = (1 - \sigma(\mathbf{s}_t))\mathbf{h}_{t-1} + \sigma(\mathbf{s}_t)\tilde{\mathbf{h}}_t, \quad (\text{C.1})$$

where \mathbf{s}_t is a linear projection of input \mathbf{x}_t and previous latent state \mathbf{h}_{t-1} and $\tilde{\mathbf{h}}_t$ is a proposed new latent state, also determined based on the input \mathbf{x}_t and previous latent state \mathbf{h}_{t-1} .

^{C.1}This chapter is based on the supplementary material of:
Gumbsch, C., Butz, M. V. & Martius, G. (2021). Sparsely Changing Latent States for Prediction and Planning in Partially Observable Domains. *Advances in Neural Information Processing Systems* (NeurIPS 2021), 17518–17531.

LSTMs use two gates, i.e. a forget and an input gate, with the sigmoid activation function σ to determine whether to update their latent (cell) state \mathbf{h}_t with

$$\mathbf{h}_t = \sigma(\mathbf{s}_{1,t})\mathbf{h}_{t-1} + \sigma(\mathbf{s}_{2,t})\tilde{\mathbf{h}}_t, \quad (\text{C.2})$$

where $\mathbf{s}_{1,t}$ and $\mathbf{s}_{2,t}$ are linear projections and $\tilde{\mathbf{h}}_t$ a non-linear function of the input and previous hidden state (RNN cell output).

Nonetheless it is not straightforward to apply our approach, described in Sec. 5.2, to GRUs and LSTMs. Our loss (see Eq. 5.5) punishes non-zero gate activation. The sigmoid activation function σ only achieves an output of zero if its input converges to negative infinity, thus never truly achieving zero output. Thus, their gating function would need to be modified or replaced, e.g. by our ReTanh gate Λ .

However, even when their gate activation function is replaced, the performance of LSTMs and GRUs is negatively affected by piecewise-constant latent states. For both networks, input information essentially needs to pass through the latent state to affect the network output. For GRUs, the network output corresponds to the latent state \mathbf{h}_t . Thus, a GRU with constant latent states will produce constant outputs. In LSTMs, the network output is computed by multiplying the latent (cell) state with an input-dependent output gate. Thus, in LSTMs, a constant latent state will result in a constant output that is scaled depending on the network input.

GATELORD attempts to overcome the downsides outlined by using LSTMs and GRUs with our proposed latent state regularization. Like GRUs, GATELORD uses a single update gate to avoid unnecessary parameters. Furthermore, GATELORD separates the latent state from the network output, as done in LSTMs, which have both a cell state and a hidden state. In addition to that, GATELORD uses more powerful functions for computing the network output such that the input and latent states both have additive and multiplicative effects on the network output. Note that GATELORD still has approximately the same number of parameters as a GRU.

C.2 Experimental Details

The code for running our experiments can be found at:

<https://github.com/martius-lab/GateLORD>

C.2.1 Predictive Models: General Training Principles and Hyperparameter Search

In the following, we will outline the general training principles that we used in all experiments, when the RNNs were trained as *predictive models*. Training details for reinforce-

Table C.1: Learning rate choices for Billiard Ball (BB), Robot Remote Control (RRC), Shepherd, and Fetch Pick&Place (FPP)

Experiment	GATELORD	LSTM	GRU	Elman RNN
BB teacher forcing (Sec. 5.4.3)	0.001	0.001	0.001	0.00005
BB scheduled sampling (Sec. 5.4.3)	0.0005	0.0005	0.0005	0.0005
RRC (Sec. 5.4.4)	0.005	0.005	0.005	0.005
RRC improved baselines (Suppl. C.4.3)	0.005	0.001	0.001	0.001
Shepherd (Sec. 5.4.5)	0.001	0.001	0.001	0.001
FPP filtered data (Suppl. C.4.4)	0.005	0.005	0.005	0.005
FPP full data (Suppl. C.4.5)	0.001	0.001	0.001	0.001

ment learning experiments are found in Suppl. C.2.6. Suppl. C.2.2 - C.2.5 provide further details specific to each simulation independent of the hyperparameter search (e.g. dataset size, batch size, etc.).

In our experiments, we train each network to predict the change in the observations instead of the next observation (i.e. residual connections) to avoid the trivial solution of achieving high prediction accuracy simply by outputting the input observation. However, since the change in observation can be quite small (typically $\Delta \mathbf{o}_t < 0.1$) we use a constant c to scale the network output when used as autoregressive input i.e. $\hat{\mathbf{o}}_{t+1} = \mathbf{o}_t + c \cdot \hat{\mathbf{y}}_t$. We set $c = 0.1$ in all of our experiments, which corresponds to scaling $\Delta \mathbf{o}_t$ by a factor of 10. For the task-based loss, i.e. $\mathcal{L}^{\text{task}}$ in Eq. 5.2, we use the mean squared error between predicted observations $\hat{\mathbf{o}}_t$ and real observations \mathbf{o}_t .

We train the networks using Adam (Kingma & Ba, 2015) with the hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 0.0001$. The learning rate η was determined through a grid search with $\eta \in \{0.005, 0.001, 0.0005, 0.0001, 0.00005\}$ for each scenario. For this grid search, we examined two random seeds for each parameter configuration and chose the setting that results in the lowest mean squared prediction error on a validation set after full training. The learning rates chosen for all experiments are listed in Table C.1.

In addition to determining the learning rate, we also use grid search to determine the number of RNN layers for all scenarios with simulated physics, i.e. Billiard Ball and Fetch Pick&Place. For LSTMs, GRUs, and Elman RNNs we compare the 1-layered RNNs to a stacked version composed of up to three RNN cells (f_ϕ in Fig. 5.1d). For GATELORD we instead considered 1- to 3-layered r_ϕ and g_ϕ -networks (see Fig. 5.1a), since we found that this typically results in a stronger increase in performance with fewer parameters compared to stacking GATELORD cells. In Billiard Ball (Sec. 5.4.3) and Fetch Pick&Place (full data, Suppl. C.4.5), all networks achieve a slightly better mean prediction accuracy with the 3-layered versions, which is why we use the 3-layered versions to compare the

prediction accuracy. However, for GRUs and LSTMs the 3-layered versions have three times the number of latent state dimensions, which negatively affects the interpretability of the latent states. Therefore, to make a fair comparison in terms of explainability, we additionally performed experiments with 1-layered LSTMs and GRUs to visualize the latent states (e.g. in Fig. 5.8b). For Fetch Pick&Place with pre-selected reach-grasp-lift sequences (Suppl. C.4.4) there was no noticeable improvement when increasing the number of layers, therefore, we used one-layered versions of the networks.

RNNs can suffer from the exploding gradient problem when predicting long sequences (Hochreiter, 1998). An effective technique to deal with this is *gradient norm clipping* (Pascanu et al., 2013). Here, the norm of a backpropagated gradient is clipped when it exceeds a threshold. We applied gradient norm clipping in all our experiments with a clipping threshold of 0.1.

In Sec. 5.4.3 we showed that training models using teacher forcing can be problematic. Thus, in all our other experiments, we train the networks using scheduled sampling (Bengio et al., 2015a), a curriculum learning strategy that smoothly changes the training regime from teacher forcing to autoregressive predictions. When applying scheduled sampling, a probability p_i is used to stochastically determine whether the real input is fed into the network (teacher forcing) or whether to use the previous network output. This sampling probability p_i decreases with training time i . Based on Bengio et al. (2015a), we use an exponentially decreasing probability p_i with

$$p_i = \max(k^i, p_{\min}) \tag{C.3}$$

where i is the epoch number, $k < 1$ a constant, and p_{\min} the minimum sampling probability. We set $k = 0.998$ in all experiments. The minimum sampling probability p_{\min} is chosen individually for each scenario.

All experiments using predictive models were performed with 20 different random seeds for each setting.

C.2.2 Billiard Ball

In Billiard Ball, a ball is shot on a pool table with low friction. We generated sequences of 50 time steps by shooting the ball from a random starting position in a random direction with a randomly selected velocity. The sequences were generated using the Open Dynamics Engine (ODE)^{C.2}—an open source physics simulator for simulating rigid body dynamics. Sequences contain only observations $\mathbf{o}_t \in [-1, 1]^2$, which are composed of the positions of the ball, and no actions ($\mathcal{A} = \emptyset$).

^{C.2}ODE, available at <http://www.ode.org/>, is licensed under the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation.

The networks were trained on a training set of 12.8k sequences and tested on a testing set of 3.2k sequences. Hyperparameters were determined based on a validation set of 3.2k sequences. All datasets were balanced to include different velocities and to guarantee that in at least 15% of the sequences the ball drops into a pocket. We trained the networks using minibatches of size 128 for 5k epochs. We applied scheduled sampling (Bengio et al., 2015a) by exponentially annealing the sampling probability p_i to 0.

We used an 8-dimensional latent state \mathbf{h}_t for all RNNs. The latent state \mathbf{h}_0 was initialized based on the first two inputs using a 3-layered MLP f_ϕ^{init} (neurons per layer: $64 \rightarrow 32 \rightarrow 16$). All RNNs used a 3-layered MLP f_ϕ^{pre} (neurons per layer: $64 \rightarrow 32 \rightarrow 16$) to preprocess inputs and a single linear mapping as a readout layer f_ϕ^{post} .

C.2.3 Robot Remote Control

In the Robot Remote Control scenario, an agent continuously moves through a room based on its two-dimensional actions $\mathbf{a}_t \in [-1, 1]^2$. After the agent reaches a computer, it also controls the position of a robot in another room through its actions. The goal during planning is to move the robot to a goal area. The observation $\mathbf{o}_t \in [-1, 1]^4$ is composed of the position of the agent and the position of the robot. The robot and agent start from randomly sampled positions, while the computer and goal area are always at the same fixed positions. The robot is controlled as soon as the distance between the agent and the computer is below a certain interaction threshold (0.1).

We generated datasets composed of 50 time step rollouts using two synthetic policies. The dataset $\mathcal{D}^{\text{time}}$, which contains spurious temporal dependencies, was generated by sampling uniformly distributed random actions that were scaled by a factor that increases linearly with time from 0.0001 to 1.0. The (generalization) dataset $\mathcal{D}^{\text{rand}}$ was generated by sampling uniformly distributed random actions without further modifications. Both datasets were balanced in terms of robot control events such that in half of the sequences the robot was controlled by the agent. The datasets were split into equally sized training, validation, and testing sets (6.4k sequences each). The validation sets were used to determine hyperparameters. The networks were trained for 5k epochs using minibatches of size 128. We trained the networks using scheduled sampling (Bengio et al., 2015a) by exponentially annealing the sampling probability p_i to a minimum value of $p_{\min} = 0.02$.

In this scenario, the latent states \mathbf{h}_t of all RNNs were 8-dimensional and initialized based on the first input using a 2-layered MLP f_ϕ^{init} (neurons per layer: $16 \rightarrow 8$). All RNNs used a 3-layered preprocessing f_ϕ^{pre} (neurons per layer: $32 \rightarrow 16 \rightarrow 8$) and a linear mapping f_ϕ^{post} from the output of the RNN cell to the overall output.

During planning, the goal was to move the robot to the goal area (distance < 0.15) within 50 time steps. For model-based planning, we used iCEM (Pinneri et al., 2021a).

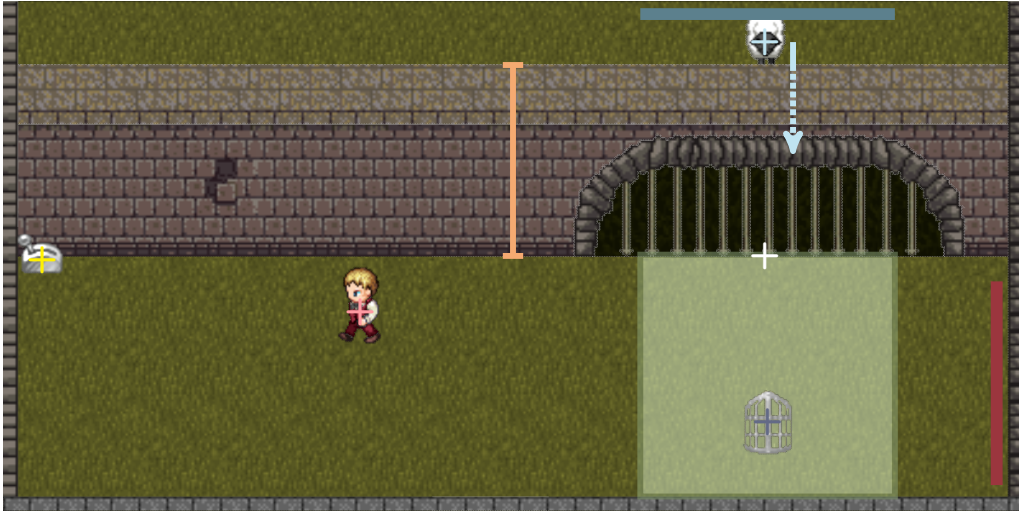


Figure C.1: Shepherd scenario: Relevant positions are marked by a plus (+), with the lever in yellow, the agent in pink, the sheep in cyan, its reappearance position in white, and the cage in purple. The orange bar visualized the wall height. The blue line and red line illustrate the sheep’s and agent’s starting position, respectively. The green area illustrates the cost function used for model-based planning. See text for more details.

We left the default hyperparameters as outlined in Pinneri et al. (2021a), but employed a planning horizon $J = 50$ and simulated 256 trajectories per optimization step. Additionally, we used colored noise with a colored noise scaling exponent $\rho = 3$. Cost was defined as the distance between the robot and the goal area. We found that iCEM, which was previously used with the ground truth simulator as a model (Pinneri et al., 2021a), was relatively sensitive to model errors, resulting in the agent often slightly missing the computer or walking over it without activating the robot. To avoid floor effects based on the planning method, we simplified the task during planning by increasing the radius to interact with the computer by 50%.

C.2.4 Shepherd

In the Shepherd scenario, illustrated in Fig. C.1, an agent’s goal is to catch a sheep using a portable cage. The agent’s actions $\mathbf{a}_t \in [-1, 1]^3$ control the agent’s two-dimensional movement and whether the cage is grasped and carried ($a_t^3 > 0$) if it is in proximity. When the cage is carried, it moves with the agent. In every sequence, a sheep starts on the upper side of the scene (blue line in Fig. C.1). The sheep moves downwards with a randomly selected velocity, i.e. only changing its y -position (cyan arrow in Fig. C.1). Therefore, the

horizontal x -position of the sheep remains the same. Once the sheep reaches a wall, its position is occluded from the observation. The height of the wall (orange bar in Fig. C.1) varies between simulations. The agent triggers the reappearance of the sheep by activating a lever in a fixed position (yellow + in Fig. C.1). The lever is activated once the distance between agent and lever is below a certain interaction threshold. As a result, a gate in the wall opens, causing the sheep to appear in the same horizontal position as before, but in a lower vertical position (white + in Fig. C.1). After its reappearance, the sheep moves downward with the same velocity as before. It stops moving once it reaches the cage (distance below a certain threshold) or once it reaches the lower border of the scene. Observation $\mathbf{o}_t \in [-1, 1]^7$ contains the agent’s position (pink + in Fig. C.1), the sheep’s position (cyan +), the cage’s position (purple +), and the height of the wall (orange bar). When the sheep is occluded, its position is masked by replacing it with a fixed value (-1) outside the normal range of coordinates.

We generated a dataset of 100 time step sequences by using randomly sampled actions. In 75% of the sequences up- and left-movements were sampled more frequently to get the agent to activate the lever. The dataset was divided into training data (12.8k sequences), testing data (12.8k sequences), and validation data (6.4k sequences). To balance the datasets across possible events, we ensured that in each dataset during 75% of the sequences the lever was activated and in 25% of the sequences the sheep was caught in the cage. We trained the networks using minibatches of size 128 for 10k epochs. We used scheduled sampling (Bengio et al., 2015a) as a training regime and exponentially decreased the sampling probability p_i to a minimum value of $p_{\min} = 0.05$.

All RNNs used 8-dimensional latent states \mathbf{h}_t . The latent state \mathbf{h}_0 was initialized based on the first two inputs using a 3-layered MLP f_{ϕ}^{init} (neurons per layer: $64 \rightarrow 32 \rightarrow 16$). All RNNs used a 3-layered preprocessing f_{ϕ}^{pre} (neurons per layer: $64 \rightarrow 32 \rightarrow 16$) and a linear mapping f_{ϕ}^{post} as a readout layer.

During planning, the agent started on the right side of the environment (red line in Fig. C.1) holding the cage. The agent had 60 time steps to place the cage, move to the lever to open the gate, and let the sheep enter the previously placed cage. We chose a tight task horizon of 60 time steps for this task to eliminate time-consuming solutions that avoid predicting the future position of the occluded sheep, e.g. by catching the sheep after its reappearance by going back and replacing the cage. For model-based planning, we used iCEM (Pinneri et al., 2021a) with the same parameters as in Suppl. C.2.3 but predicting for a longer planning horizon of $J = 100$ time steps as during training. Cost was defined as the distance between the sheep and the cage, which was clipped to a large constant value when the sheep was above the gate (i.e. outside of the green area in Fig. C.1). As in Robot Remote Control (Suppl. C.2.3), we increased the radius of interaction between the lever and the cage during planning by 50%.

C.2.5 Fetch Pick&Place

Fetch Pick&Place is a benchmark reinforcement learning environment of OpenAI Gym (Brockman et al., 2016). In Fetch Pick&Place a 7 degrees-of-freedom robotic arm with a two-fingered gripper is position controlled through its four-dimensional action. The state of the scenario $\mathbf{s}_t \in \mathbb{R}^{25}$ is composed of the positions of the endeffector and the object, the relative position between endeffector and object, the distance of the fingers to the center of the gripper, the rotation of the object, and the positional and rotational velocities of the endeffector, the object, and the fingers. To make the scenario partially observable, we omitted the positional and rotational velocities as well as the rotation of the object in the observation $\mathbf{o}_t \in \mathbb{R}^{11}$. The four-dimensional actions $\mathbf{a}_t \in [-1, 1]^4$ control the three-dimensional position of the endeffector and the closing or opening of the fingers. Internally, the position control of the endeffector is realized by a PID-controller that runs at a higher frequency.

We generated our training data using APEX (Pinneri et al., 2021b), a policy-guided model predictive control method, which was trained to move the object to a random position of the goal. APEX was deployed using the ground-truth simulator as the internal model and hyperparameters as detailed in Pinneri et al. (2021b).

APEX finds various surprisingly creative ways to move the object to the goal position, including pushing, sliding, or flicking the object. For the experiments on policy generalization (Suppl. C.4.4), we only considered sequences in which the object was grasped and lifted. Thus, we excluded all sequences in which the object moved while not being inside the gripper. For training and testing we considered 3.84k sequences with a length of 25 time steps, in which the hand grasps the object exactly at $t = 5$. A grasp was only considered if the relative x - and y - distance to the gripper was less than 0.0005 and the relative z -distance was less than 0.15. We randomly split this dataset into a training (3.2k) and testing set (640). For the generalization set, we composed of 3.2k randomly selected sequences in which the grasp occurs later ($t \in [6, 10]$).

In an additional experiment outlined in Suppl. C.4.5 we train the networks on all kinds of sequences. For that, we randomly split the collected dataset without further filtering into training (12.8k sequences), validation (6.4k sequences) and testing (6.4k sequences) sets. Here, we considered sequences that are 50 time steps long.

In both experiments, we trained the networks using minibatches of size 128 for 5k epochs using scheduled sampling (Bengio et al., 2015a), where we exponentially decreased the sampling probability p_i to a minimum value of $p_{\min} = 0.05$. The latent state \mathbf{h}_t of all RNNs was 16-dimensional. The first latent state \mathbf{h}_0 was initialized based on the initial input $(\mathbf{o}_1, \mathbf{a}_1)$ using a 2-layered MLP f_ϕ^{init} (neurons per layer: $32 \rightarrow 16$). All RNNs used a 3-layered preprocessing f_ϕ^{pre} (neurons per layer: $64 \rightarrow 32 \rightarrow 16$) and a linear mapping two-

layered MLP f_ϕ^{post} to the network output. In the experiment using simpler filtered data (Suppl. C.4.4) we used one-layered RNN cells. For the diverse data set (Suppl. C.4.5) we use stacked RNN cells (3 layers) and GATELORD with 3-layered g_ϕ - and r_ϕ -functions.

C.2.6 Reinforcement Learning: General Training Principles and Hyperparameter Search

For our RL experiments in MiniGrid (Chevalier-Boisvert et al., 2018b), we used an actor-critic architecture as previously done by Chevalier-Boisvert et al. (2018a).^{C.3} The architecture can be seen as a modified version of our general architecture (Fig. 5.1d), shown in Fig. C.2. The image-like input is preprocessed by a three-layered convolutional neural network f_ϕ^{pre} with 2×2 convolution kernels and with max-pooling after the first layer. The 64-dimensional image embedding is processed by an LSTM with 64-dimensional latent state. The LSTM output is processed by two separate MLPs, similar to using two f_ϕ^{post} in Fig. 5.1d, which take the role of the actor and the critic. The actor MLP f_ϕ^{actor} outputs the policy π_t , which determines the next one-hot encoded action \mathbf{a}_t . The critic MLP f_ϕ^{critic} outputs a value estimate v_t . Both MLPs use two layers with 64 neurons on the intermediate layer. In our experiments with GATELORD, we only replace the LSTM cell and leave f_ϕ^{pre} , f_ϕ^{actor} , and f_ϕ^{critic} unmodified.

As in Chevalier-Boisvert et al. (2018a), we train the system using Proximal Policy Optimization (PPO) (Schulman et al., 2017) with a batch size of 256. We took the PPO hyperparameters from (Chevalier-Boisvert et al., 2018a), setting $\gamma = 0.99$ and the generalized advantage estimation to 0.99.

We train the system using Adam (Kingma & Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 0.0001$. To determine the learning rate η we ran a grid search on the vanilla system (LSTM) with $\eta \in \{0.005, 0.001, 0.0005, 0.0001\}$ for two random seeds and compared the mean rewards after training. In five of the six environments $\eta = 0.001$ achieved the best results. Thus, for consistency we ran the MiniGrid experiments with a learning rate of $\eta = 0.001$. For the one environment (KeyCorridorS3R2) in which a lower learning rate ($\eta = 0.0005$) produced better results, we additionally evaluated the system with the optimized

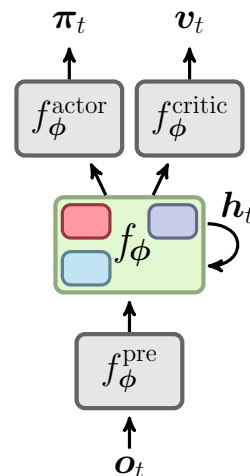


Figure C.2: RL architecture used in MiniGrid.

^{C.3}We used an implementation by one of the authors available at <https://github.com/lcswillems/rl-starter-files>. The code is licensed under MIT license.

learning rate and report the results in Suppl. C.4.6. As before, we apply gradient norm clipping (Pascanu et al., 2013) with a clipping threshold of 0.1. The loss was backpropagated for 32 time steps.

When using GATELORD, we simply replaced the LSTM cell and left all hyperparameters unmodified. PPO loss (Schulman et al., 2017) was used as $\mathcal{L}^{\text{task}}$ in Eq. 5.2 and we set $\beta = 0.01$ in all experiments. All reinforcement learning experiments were run with 10 random seeds per configuration.

C.2.7 MiniGrid

MiniGrid (Chevalier-Boisvert et al., 2018b) is a library of partially observable benchmark reinforcement learning problems.^{C.4} All MiniGrid environments consist of $N \times M$ tiles. Each tile can be empty or contain one entity such as keys, doors, or walls. The agent receives an image-like, egocentric view of the 7×7 tiles in front of the agent. For each tile, the agent receives a 3-dimensional signal, describing what type of object is in this tile, the color of the object, and its state (e.g. open, closed, or locked doors). The agent cannot see through walls or closed doors. In every time step, the agent can perform one of the following actions: move forward, turn left, turn right, pick-up an object, drop-off an object, or interact with an object (e.g. open doors). In all environments, a sparse reward of 1 is received once the task is fulfilled. In some environments, the time to fulfill a task is used to discount the rewards. Figure C.3 shows all the problems we consider.

In `DoorKey-8x8` (Fig. C.3a) the agent needs to move to the green square behind a locked yellow door. The agent needs to learn to pick up a yellow key to open the door. The environment is 8×8 tiles big, but the size of the two rooms varies per simulation. `DoorKey-16x16` (Fig. C.3g) is the same problem, but in a larger 16×16 environment. We use the larger version to test zero-shot generalization, by training the system in the smaller environment and testing it on the larger one (see Sec. 5.4.7).

In `RedBlueDoors-8x8` (Fig. C.3b) the agent is randomly placed in a room (8×8 tiles) with a red and a blue door. The agent has to first open the red door and afterwards open the blue door. Opening the blue door first results in an early episode termination.

In `KeyCorridorS3R2` (Fig. C.3c) the agent needs to pick up a ball. The ball is locked behind a door and the key is hidden in some other room. Thus, the agent needs to learn to explore the rooms, by opening differently colored doors, to find the key. The agent can only pick up the ball if the agent does not hold the key, so after unlocking the door leading to the ball, the agent needs to drop the key.

^{C.4}MiniGrid is available at <https://github.com/maximecb/gym-minigrid>. MiniGrid is licensed under Apache License 2.0.

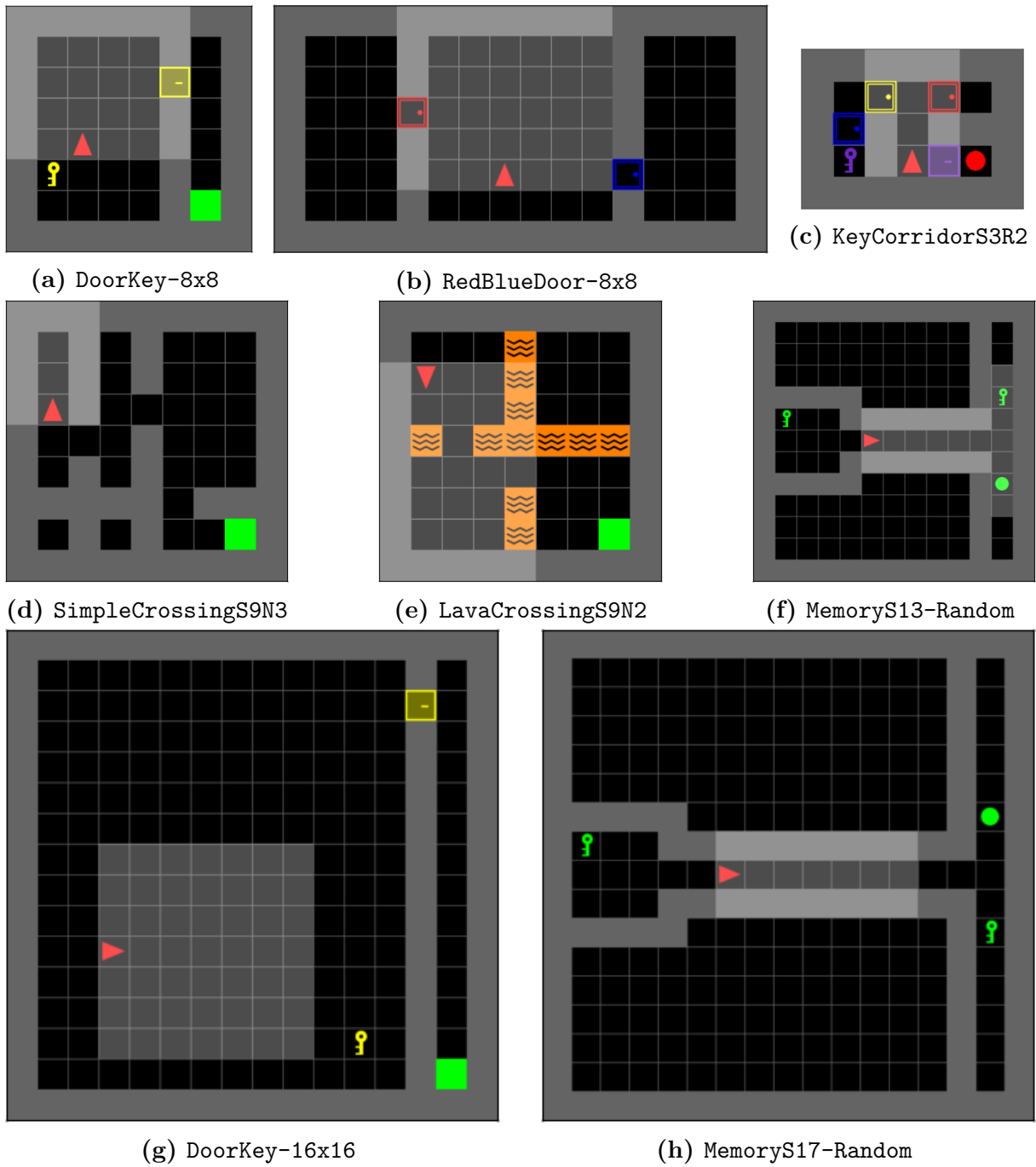


Figure C.3: MiniGrid environments used in this work.

In `SimpleCrossingS9N3` (Fig. C.3d) the agent needs to navigate through a maze to a green square in the bottom left corner. The maze is constructed randomly by three walls that run horizontally or vertically through the room. Each wall has a single gap. `LavaCrossingS9N2` (Fig. C.3e) poses the same problem, however, the walls of the maze are replaced by two lava rivers. Lava rivers do not occlude the view, but entering lava terminates the episode without rewards. Due to early terminations and sparse rewards, this environment is much more challenging to learn than the maze with walls.

`MemoryS13Random` (Fig. C.3f) is a memory task. Here the agent needs to memorize a green object (key or ball) in one room, move through a corridor, and then either go left or right to the matching object. The environment is 13×13 tiles big. The length of the corridor is randomly generated per run. In `MemoryS17Random` (Fig. C.3h) the same problem needs to be solved, but the environment is larger (17×17 tiles). We use this version to test zero-shot generalization by training the system in the smaller environment and testing it in the larger one (see Sec. 5.4.7).

C.3 Ablation Studies

In this section, we investigate the importance of each of the components of our proposed architecture.

C.3.1 Ablation of the Type of Gate Function

We use the Billiard Ball scenario, trained using scheduled sampling (Bengio et al., 2015a) as in Sec. 5.4.3, to analyze the effect of different gate activation functions. In an ablated setting, we replace our `ReTanh` activation Λ in Eq. 5.8 with a sigmoid activation function σ . Furthermore, we test using the Heaviside step function Θ as a gate activation function in Eq. 5.8. When using the Heaviside step function, we estimate the gradients using the straight-through estimator (Bengio et al., 2013), which treats the step function as a linear function during the backward pass (illustrated in Fig. 5.1c). We tested the Heaviside gates both with our L_0 loss ($\beta = 0.001$) and without latent state regularization ($\beta = 0$). Because a gate output of 0 is practically not achieved for the sigmoid function, we tested the sigmoidal gates without latent state regularization ($\beta = 0$).

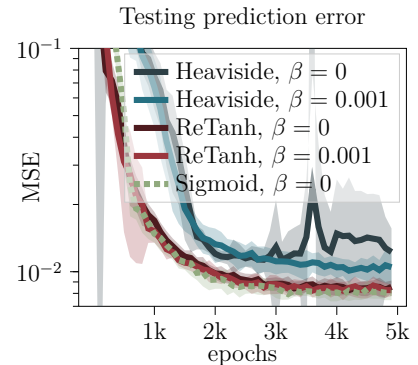


Figure C.4: Gate activation ablation: Billiard Ball test error for GATELORD with different gate functions.

Figure C.4 shows the autoregressive prediction errors of the ablated versions of GATELORD. The ablations with Heaviside gates perform worse than GATELORD with nonbinary gates. When using the Heaviside gate without any regularization, the mean prediction error even increases with training time. GATELORD with a sigmoid gate and our ReTanh gate reach the same level of prediction accuracy.

We believe that the worse performance of the Heaviside gate is due to the network profiting from multiplicative computations when computing the next latent state. For the Heaviside gate, interpolations of old and new latent states are not possible. Here, the latent state is either completely replaced or left unmodified. We conclude that our novel ReTanh gate is as suitable for gating as the classically used sigmoid gate. Furthermore, it has the practical advantage of achieving an output of exactly 0, which allows the gate activation to be regularized, as we do with our L_0 loss.

C.3.2 Effect of Gate Stochasticity

To ablate the effect of the gate noise we compare GATELORD with different strengths of the gate noise. For deterministic gates, we set $\epsilon = 0$ in Eq. 5.10. Additionally, we compare two values for the noise variance σ of the diagonal covariance matrix Σ in Eq. 5.10. We test the effects of gate stochasticity for a fixed value of gate regularization $\beta = 0.01$ in the Billiard Ball task.

Figure C.5a shows the prediction errors when comparing deterministic gates to stochastic gates with different gate noise. There are no noticeable differences in the prediction accuracy between the different settings. Thus, reasonable values of noise on the gate input during training do not noticeably affect the prediction error during testing. Figure C.5b shows the average changes in latent state per sequence, calculated as $\mathbb{E}_{i,t}[\Theta(\Lambda(s_t^i))]$, for all settings. Here, a larger value of gate noise results in fewer gate openings, and thus in fewer changes in the latent state.

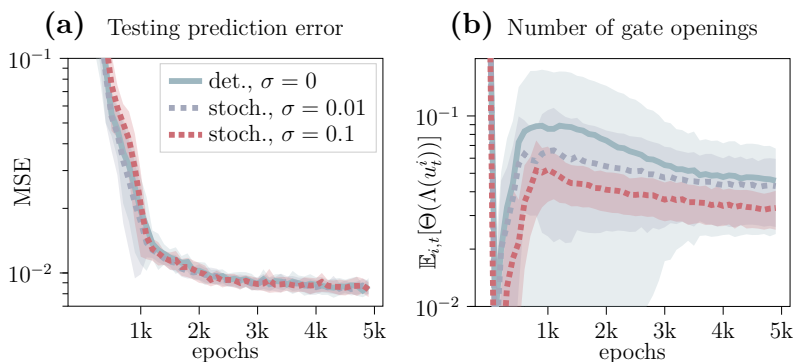


Figure C.5: Gate stochasticity ablation Comparing effect of gate noise on GATELORD ($\beta = 0.01$) in Billiard Ball for prediction errors (a) and mean number of gate openings (b). Shaded areas denote standard deviation.

We conclude that combining stochastic gates together with our L_0 loss has a regularizing effect: GATELORD trained with stochastic gates seems to achieve the same level of prediction accuracy as with deterministic gates, but changes its latent states more sparsely.

C.3.3 Ablation of the Latent State Initialization

Next, we ablate the effect of f_ϕ^{init} , which sets the latent state based on a few initial inputs (see Fig. 5.1d). We compare all RNNs with variants without f_ϕ^{init} in the Billiard Ball scenario. When we omit f_ϕ^{init} , we initialize the latent state with $\mathbf{h}_0 = \mathbf{0}$.

Figure C.6 shows the prediction errors for all RNNs when using the initialization network f_ϕ^{init} (solid lines) and when initializing the latent state with zeros (dotted lines). Prediction accuracy decreases for all types of network when trained without the context network. However, how much their performance drops varies among the different types of RNNs. GRUs seem to be much less affected by using them without f_ϕ^{init} than LSTMs and GATELORD ($\beta = 0.001$).

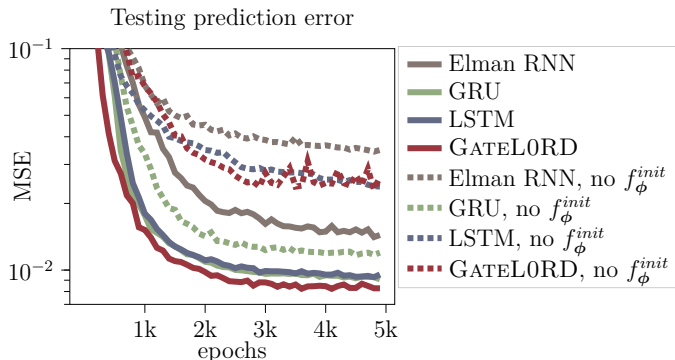


Figure C.6: Latent state initialization ablation: Effect of latent state initialization with or without f_ϕ^{init} on prediction errors in Billiard Ball.

C.3.4 Ablation of the Output Function

After updating its latent state \mathbf{h}_t , GATELORD uses two one-layered MLPs b_ϕ and m_ϕ to compute the network output as $b_\phi(\mathbf{x}_t, \mathbf{h}_t) \odot m_\phi(\mathbf{x}_t, \mathbf{h}_t)$ (see Eq. 5.9). With this output function we want to enable both additive as well as multiplicative effects of the latent state \mathbf{h}_t and input \mathbf{x}_t on the network output. Is this justified, or would a simple MLP as output function suffice?

We analyze the effect of our output function in Robot Remote Control ($\beta = 0.001$, trained on random action rollouts $\mathcal{D}_{\text{rand}}$). Here we compare GATELORD using our standard output function ($b_\phi(\mathbf{x}_t, \mathbf{h}_t) \odot m_\phi(\mathbf{x}_t, \mathbf{h}_t)$) with an ablated version using only a one-layered MLP with tanh activation ($b_\phi(\mathbf{x}_t, \mathbf{h}_t)$).

Figure C.7 shows the resulting prediction errors of GATELORD using its normal output function compared to the case without a multiplicative gate ($b_\phi(\mathbf{x}_t, \mathbf{h}_t)$). Clearly, GATELORD achieves a much better prediction when using a multiplicative output gate instead of a simple MLP. Thus, a multiplicative branch for computing the network output seems to improve the accuracy of prediction. This may also explain the poorer prediction accuracy of Elman RNNs in most tasks, since they lack the multiplicative gates that can be found in all other investigated RNNs.

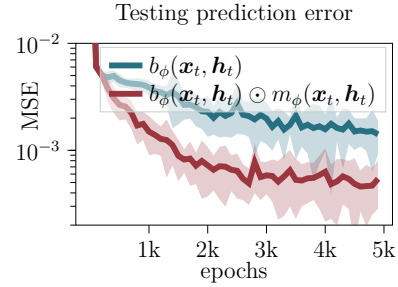


Figure C.7: Output function ablation: GATELORD output function with ($b_\phi \odot m_\phi$) or without (b_ϕ) multiplication in Robot Remote Control

C.3.5 Comparison against L_1/L_2 -versions

Our hypothesis is that sparsely changing latent states enable better generalization across spurious temporal dependencies. GATELORD implements sparse latent updates via the novel ReTanh gate, instead of the commonly used sigmoid gates, and an auxiliary L_0 loss term that is made differentiable using the straight-through estimator. Is this necessary or would a simple sigmoid gate in conjunction with an L_1 or L_2 loss also improve generalization?

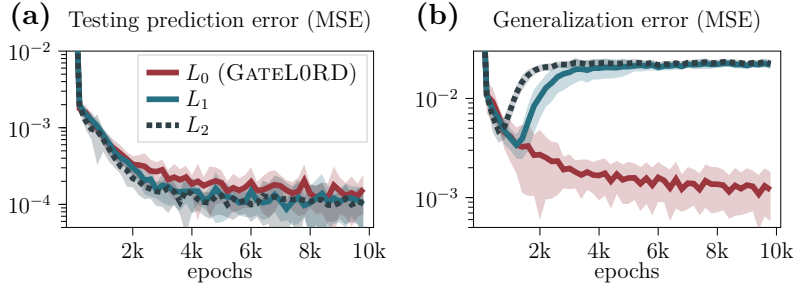


Figure C.8: Comparing L_0 -, L_1 -, and L_2 -regularization for prediction errors on the test set (a) and the generalization set (b) in Robot Remote Control.

To analyze this, we compare GATELORD against ablated versions that use a sigmoid gate and penalize the L_1 or L_2 norm of gate activations in Robot Remote Control (as in Sec. 5.4.4). We train the networks on random action rollouts with linearly increasing action magnitude and test it on either data generated by the same process (testing) or on uniformly sampled random actions (generalization). We chose a suitable regularization hyperparameter $\beta = 0.001$ for all variants.

Figure C.8a shows the prediction errors during testing. The L_1 - and L_2 -ablations

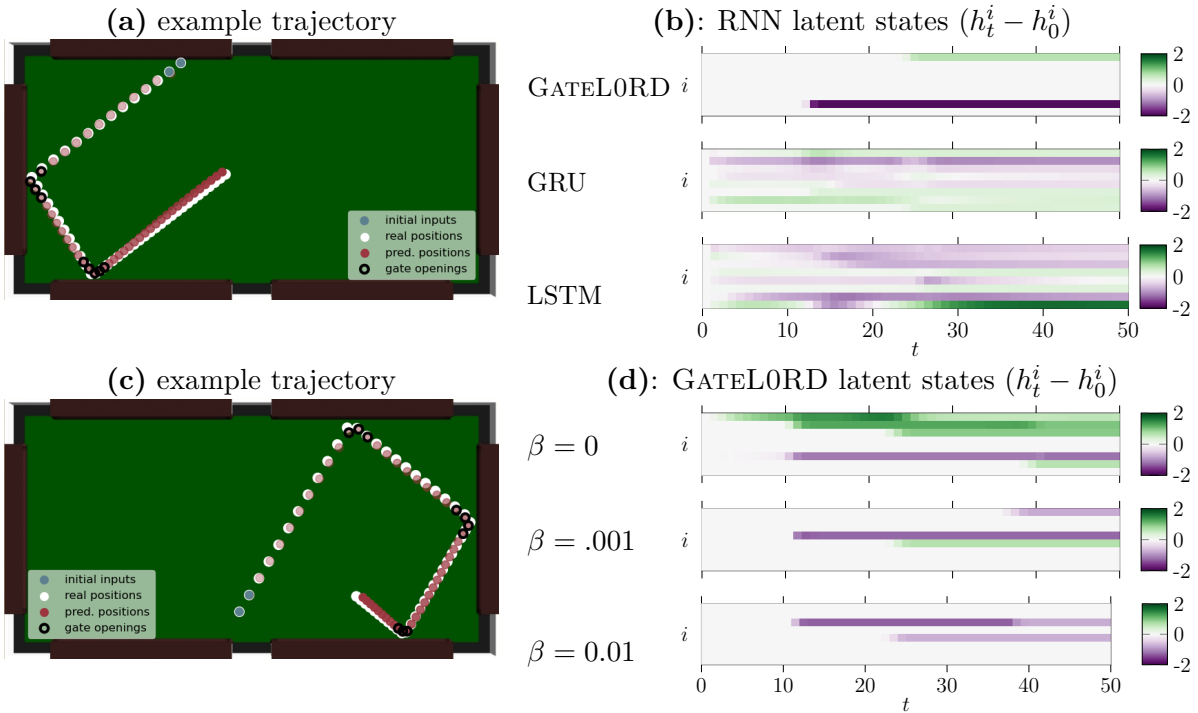


Figure C.9: Example trajectories in Billiard Ball (a) & (c): Exemplary trajectories with real positions in white, the provided inputs in blue, GATELORD ($\beta = 0.01$) position predictions in red (saturation increasing with time). The inputs for which at least one gate opened are outlined in black. **(b):** The latent states \mathbf{h}_t for different RNNs for the sequence shown in (a). **(d):** The latent states \mathbf{h}_t for GATELORD with different values of β for the sequence in (c). Latent states are shown relative to the initial latent state \mathbf{h}_0 .

achieve a very low prediction error on the test set, even exceeding GATELORD’s prediction in terms of accuracy. However, when applied to the generalization set, shown in Fig. C.8b, their prediction error increases drastically.

We conclude that the L_1/L_2 -variants achieve a low testing error, but fail to generalize to data generated by a different policy. This suggests that they also strongly overfit to spurious temporal dependencies, as LSTMs and GRUs, unlike our L_0 -version. However, it should be noted that on the test set the L_2 -variant manages to achieve the highest prediction accuracy of all investigated RNNs.

C.4 Additional Experiments and Analysis

C.4.1 Billiard Ball: Analyzing Latent States and Gate Usage

In this section, we provide further exemplary latent states for RNNs when applied to the Billiard Ball scenario. Figure C.9 shows two exemplary ball trajectory and the corresponding latent states. GATELORD is able to make accurate autoregressive predictions (see red dots in Fig. C.9a and Fig. C.9c) and tends to open its gates around wall collisions (black circles). Figure C.9b shows the latent states of GATELORD ($\beta = 0.01$) compared to the latent states of a GRU and a LSTM for the trajectory shown in Fig. C.9a. GATELORD’s changes in latent states are easily interpretable: GATELORD seems to encode x - and y - velocity in two dimensions of its latent state and updates these latent state dimensions upon collisions. LSTM and GRU latent states are much harder to interpret.

Figure C.9d shows the latent states of differently regularized GATELORD networks for the same sequence, shown in Fig. C.9c. As before, GATELORD with $\beta = 0.01$ uses two dimensions of its latent state to encode ball velocity and updates these two dimensions upon collisions. In this example, GATELORD with $\beta = 0.001$ uses three dimensions to encode the ball’s velocity. With every collision, a different latent state dimension is updated, instead of using the same dimension for changes in y -velocity, as done by GATELORD with $\beta = 0.01$. In this example, GATELORD with $\beta = 0$ uses five dimensions to encode x - and y - velocities. At points of collision, multiple latent dimensions change.

To further illustrate how the regularization hyperparameter β affects latent state changes, we plot the number of latent state dimensions that change on average while predicting a Billiard Ball sequence in Fig. C.10. As expected, a stronger regularization through β results in fewer dimensions of the latent state changing.

As shown in Fig. 5.3d, even without regularization ($\beta = 0$) GATELORD continuously decreases the mean number of gate openings. After 5k epochs, on average a gate opens less than 50% of the time. Similarly, it does not use all dimensions of its latent state, as shown in Fig. C.10. This effect emerges from the interplay of stochastic gradient descent and the ReTanh having gradients of 0 for inputs $s_t^i \leq 0$. Over training time, gates will randomly close and be kept closed if they do not contribute to decreasing the loss. This effect is closely related to the “dying ReLU problem” when using ReLU activation functions (Lu, 2020). While dying ReLUs are considered a problem, in our case this is advantageous

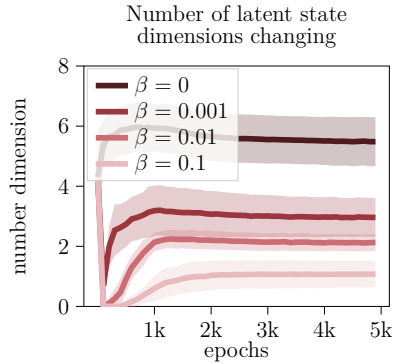


Figure C.10: Latent dimension usage in Billiard Ball for different values of β .

whenever gate regularization is beneficial. We believe that this makes GATELORD more robust to out-of-distribution shifts than GRUs and LSTMs, even without regularization. For example, GATELORD with $\beta = 0$ achieves a smaller mean autoregressive prediction error when trained using teacher forcing (Fig. 5.3a), compared to the other RNNs.

C.4.2 Robot Remote Control: Learned Latent States

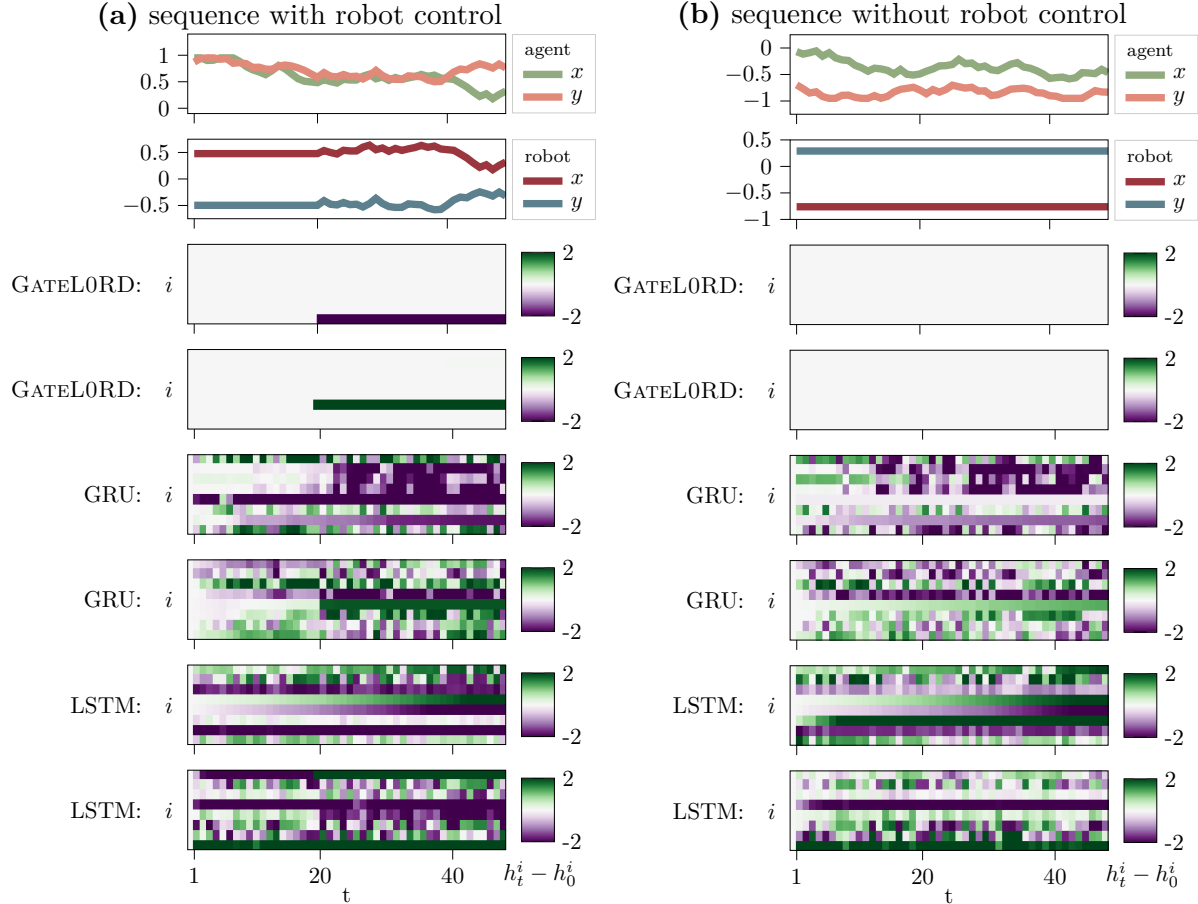


Figure C.11: Latent states in Robot Remote Control for sequences in which the robot was either controlled (a) or not (b). Latent states \mathbf{h}_t are shown relative to their initialization \mathbf{h}_0 . We provide the latent states for two GATELORD, GRUs, and LSTMs (different random seeds). One row shows the same random seed.

In this section, we further analyze latent states in the Robot Remote Control scenario. Figure C.11a shows one exemplary sequence in which the robot was controlled and the

Table C.2: Gating in Robot Remote Control

	gate open	gate closed
control	0.978 ± 0.016 (hits)	0.022 ± 0.016 (misses)
no control	0.089 ± 0.037 (false alarms)	0.911 ± 0.037 (correct rejections)

corresponding latent states for two instantiations of GATELORD, GRU, and LSTM with different random seeds. GATELORD seems to only use one dimension of its latent state to encode robot control. For GRUs and LSTMs the latent states also seem to strongly change around the point where the agent gains control over the robot; however, their latent states are not as interpretable. Figure C.11b shows one exemplary sequence, in which the robot was not controlled. Here, GATELORD does not modify its latent states, whereas LSTMs and GRUs continuously change their latent states.

When the robot is not controlled, as in Fig. C.11b, Robot Remote Control is fully observable. Thus, it seems that GATELORD is capable of learning to distinguish observable from unobservable information and attempts to update its latent state only when unobservable information changes. To evaluate this claim, we feed in all generalization sequences and classify gate usage. The inputs of the sequences were classified based on whether control of the robot was triggered at this time step (control) or not (no control). Additionally, we analyzed for each input whether one of GATELORD’s gates opened (gate open) or not (gate closed). The mean gate openings for the two events are shown in Table C.2 with \pm denoting the standard deviation. GATELORD seems to mostly open its gates when robot control is triggered and tends to keep its gate closed at other time steps. Thus, GATELORD indeed seems to mostly update its latent state when the unobservable state of the environment changes.

C.4.3 Robot Remote Control: Improving RNN Generalization

In Sec. 5.4.4 we showed that LSTMs and GRUs trained on data in which action magnitude was positively correlated with time ($\mathcal{D}^{\text{time}}$), failed to properly generalize to testing data without this correlation ($\mathcal{D}^{\text{rand}}$). GATELORD showed less performance degeneration when tested on the generalization dataset. We hypothesized that GATELORD’s superior generalization performance comes from its robustness towards spurious temporal dependencies in the training data. However, an alternative explanation would be that the overfitting of LSTMs and GRUs was caused by their learning rate.

To investigate whether the other RNNs’ generalization abilities can be improved to the level of GATELORD by choosing a different learning rate, we ran a grid search over three learning rate values ($\eta \in \{0.005, 0.001, 0.0005\}$) for LSTMs, GRUs, Elman RNNs with two random initializations. We selected the learning rate that lead to the lowest mean squared prediction error for the 50 time step predictions on the generalization dataset of $\mathcal{D}^{\text{rand}}$ after 5k epochs. Seeing that a learning rate of 0.001 yielded the best generalization error for all RNNs, we reran the experiment with this learning rate (10 random seeds).

Figure C.12 shows the resulting prediction error when testing the RNNs on the generalization set of $\mathcal{D}^{\text{rand}}$. Although the prediction error of GRUs and LSTMs on the generalization test set improved compared to our previous experiment, GATELORD still achieved a lower generalization error than the other RNNs. Note that GATELORD was not further optimized in this experiment. Thus, we conclude that GATELORD’s superior generalization performance in this setting is not caused by poorly tuned learning rates.

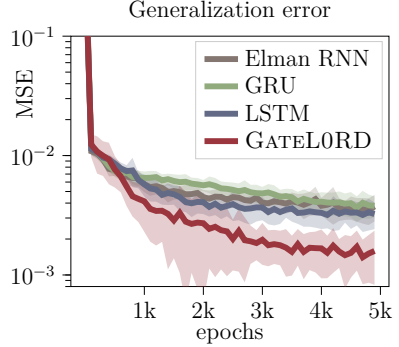


Figure C.12: Improved baselines in Robot Remote Control: Generalization prediction error with optimized learning rates.

C.4.4 Fetch Pick&Place: Generalization across Grasp Timings

In Sec. 5.4.4 we showed that GATELORD is better at generalizing across spurious temporal dependencies in the training data than other RNNs in the simple Robot Remote Control scenario. In a follow-up experiment, we want to investigate if similar effects can be found in a more complex environment and when trained on more natural training data. For that, we use the Fetch Pick&Place environment and train the networks to predict reach-grasp-and-lift sequences. The training sequences were generated by a policy-guided model-predictive control method (Pinneri et al., 2021b). Importantly, we train the network only on sequences in which the gripper first touches the object exactly at time $t = 5$. We test the networks to predict sequences where gripper-object contact occurs as during training (testing) or where the object is grasped later (generalization).

Figure C.13a shows the mean autoregressive prediction errors during testing. All networks achieve a very low prediction error. The prediction accuracy is similar for all RNNs, but LSTMs achieve a slightly lower prediction error than GATELORD ($\beta = 0.0001$). When networks were tested on sequences with different grasp times, they produced much higher prediction errors as shown in Fig. C.13b. GATELORD prediction accuracy does not drop as strongly as the accuracy of the other networks. Thus, as in the previous ex-

periments, GATELORD more robustly generalizes across spurious temporal correlations.

Figure C.15 shows the latent states of the different RNNs when predicting two exemplary sequences. Here, GATELORD ($\beta = 0.0001$) uses one or three dimensions of \mathbf{h}_t that change around the time when GATELORD predicts that the gripper grasps the object. During the predicted transportation of an object, the latent state does not change anymore. This indicates GATELORD encoding the event <transporting an object> in parts of its latent state.

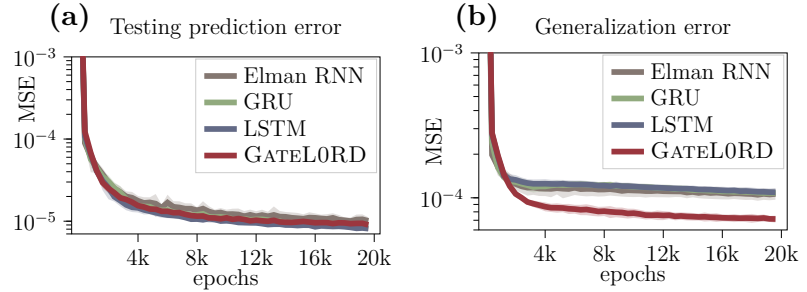


Figure C.13: Generalization in Fetch Pick&Place: prediction error on test set (a) and generalization set (b). Shaded areas denote standard deviation.

C.4.5 Fetch Pick&Place: Training on Diverse Sequences

Previously, we only considered reach-grasp-lift sequences in the Fetch Pick&Place environment. However, there are multiple other ways to move the object to a target position, such as pushing, sliding, or even flicking. Thus, in a next experiment we analyze the performance of RNNs when trained as a model on a diverse set of interactions generated by the policy-guided model-based control method APEX (Pinneri et al., 2021b).

Figure C.14 shows the mean autoregressive prediction errors for different RNNs. Here, all RNNs achieve a very similar prediction accuracy. GATELORD with $\beta = 0.001$ produces a slightly higher mean prediction error than the other RNNs, whereas GATELORD with $\beta = 0$ achieves a slightly lower error. We believe that in this scenario the small differences in prediction accuracy are a result of better approximations of the endeffector velocities. In Fetch Pick&Place the position control of the endeffector is realized by a PID-controller running at a higher frequency. Thus, in this scenario continuous latent state updates are advantageous for predicting the endeffector speed.

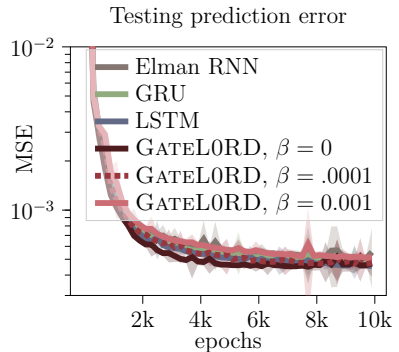


Figure C.14: In-distribution prediction in Fetch Pick&Place trained on diverse data. Shaded areas denote standard deviation.

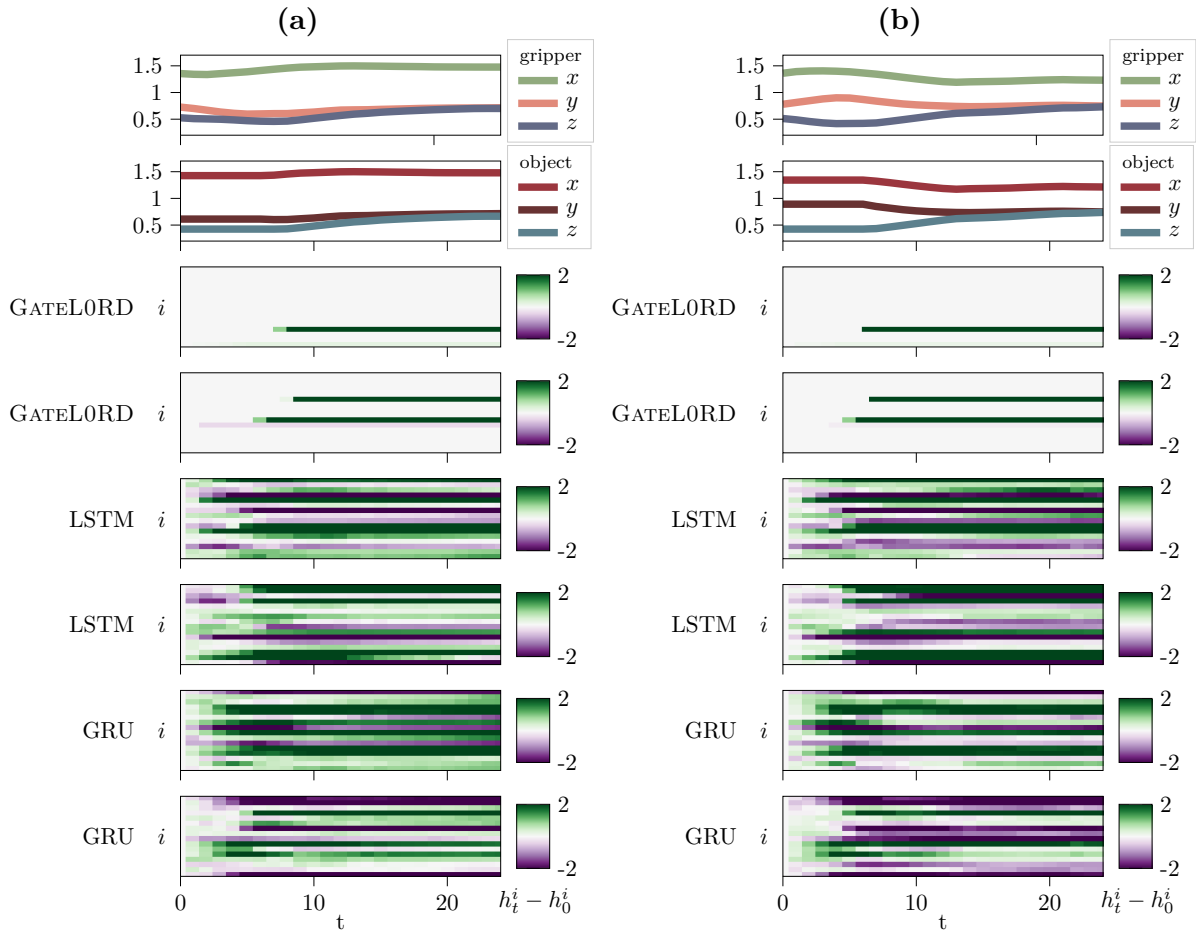


Figure C.15: Latent states in Fetch Pick&Place for two exemplary sequences. Latent states \mathbf{h}_t are shown relative to their initialization \mathbf{h}_0 . We compare the latent states for two random seeds each, where each row shows the same random seed.

C.4.6 MiniGrid: Further Analysis and Experiments

In Sec. 5.4.6 we demonstrated improved sample efficiency in MiniGrid when GATELORD replaces an LSTM in a PPO architecture. Some problems of MiniGrid discount the final reward based on the number of actions taken. Thus, another metric to judge success in MiniGrid is the mean reward collected. Figure C.16 shows the mean rewards for the vanilla architecture and architecture containing GATELORD over training experience. For all problems, the architecture containing GATELORD is more sample efficient in achieving high levels of reward.

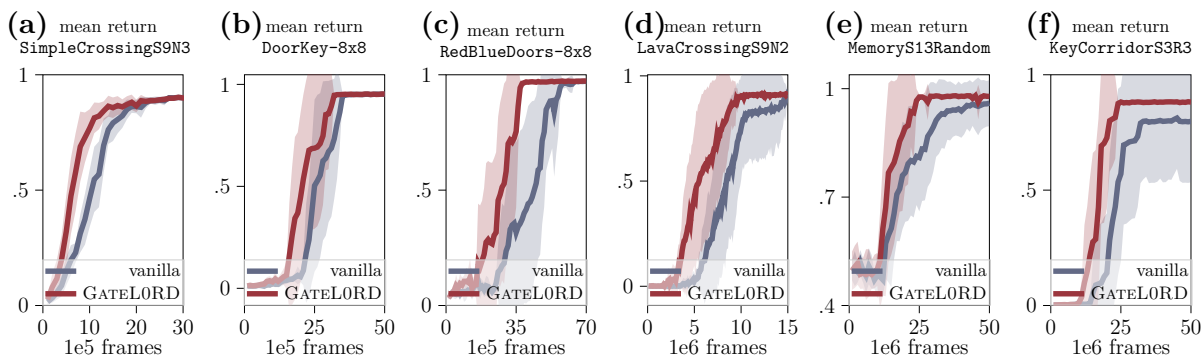


Figure C.16: Rewards in MiniGrid: Mean rewards when GATELORD replaces an LSTM (vanilla) in a PPO architecture. Shaded areas show standard deviation.

For consistency we used the same hyperparameters in all MiniGrid experiments and only swapped the LSTM cell for GATELORD. However, as described in Suppl. C.2.6 a grid search showed that for the KeyCorridorS3R3 problem a lower learning rate ($\eta = 0.0005$) resulted in higher mean rewards for the vanilla architecture. Therefore, to exclude the possibility that GATELORD outperformed the LSTM in this problem based on the choice of learning rate, we performed an additional experiment in the KeyCorridorS3R3 problem with the vanilla architecture using the optimized learning rate. The resulting mean success rate and mean rewards are shown in Fig. C.17a and Fig. C.17b, respectively. Although the vanilla architecture now manages to reach a success rate of 100% and a mean reward greater than 0.8, GATELORD still achieves the same level of performance faster.

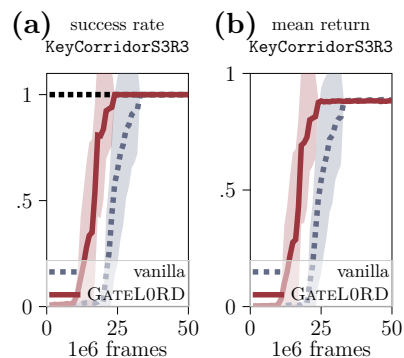


Figure C.17: Improved baselines in MiniGrid: The learning rate of the vanilla system was optimized for KeyCorridorS3R3.

D

Skip Network: Supplementary Material^{D.1}

D.1 Implementation Details

For our system in Chap. 6, we use GATELORD with a 16-dimensional latent state \mathbf{h}_t . The subnetworks g and r are implemented by multilayer perceptrons (MLPs) with three layers (64 \rightarrow 32 \rightarrow 16 feature neurons, tanh hidden activation), while o is implemented as two single-layered neural networks (16 feature neurons per layer), whose outputs are multiplied element-wise. When using GRUs (Chung et al., 2014) we found that a 16-dimensional latent state \mathbf{h}_t performs much worse on $\mathcal{D}_{\text{APEX}}$, which is why we used a GRU with 32 latent dimensions. The other components of the forward-inverse model, i.e. f_ϕ^{init} , f_ϕ^{FM} , and f_ϕ^{IM} , are also implemented as feed-forward neural networks. For f_ϕ^{init} and f_ϕ^{FM} we use MLPs with three layers (64 \rightarrow 32 \rightarrow 16 feature neurons, tanh hidden activation). The inverse model f_ϕ^{IM} has an additional multiplicative layer, similar to the GATELORD output function o (16 features per layer), followed by a three-layered MLP (64 \rightarrow 32 \rightarrow 16 feature neurons, tanh hidden activation). We add this extra layer so that f_ϕ^{IM} has the same computational power to compute its outputs based on \mathbf{o}_t and \mathbf{h}_t as f_ϕ^{FM} . We modeled the skip network as a deep MLP with five layers (512 \rightarrow 256 \rightarrow 128 \rightarrow 64 \rightarrow 32 feature

^{D.1}This chapter is based on the supplementary material of:

Gumbsch, C., Adam, M., Elsner, B., Martius, G. & Butz, M. V. (2022). Developing Hierarchical Anticipations via Neural Network-based Event Segmentation. *IEEE International Conference on Development and Learning (ICDL 2022)*, 1–8.

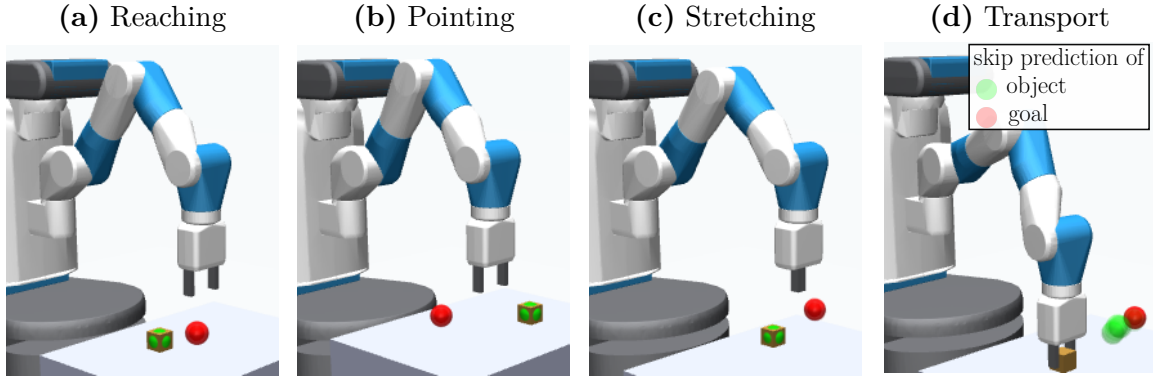


Figure D.1: Event boundary predictions for objects and goals: Exemplary skip predictions at $t = 2$ shown for reaching (a), pointing (b), and stretching (c). Skips at $t = 12$ shown for a transport event (d). A circle shows one skip-predicted position of the object (green) or the goal (red). The predictions of all 10 random seeds are overlaid.

neurons, tanh hidden activation). Networks predicting Gaussian distributions (i.e. f_ϕ^{FM} , f_ϕ^{IM} and f_θ^{skip}) use separate read-out layers for predicting the mean and predicting the covariance. The read-out layers for mean predictions have a linear activation function, whereas the read-out layers for the covariance matrix predictions use the Exponential Linear Unit (ELU) activation function shifted by 1 to avoid covariances ≤ 0 .

We trained and tested on datasets of 9.2k sequences using a batch size of 192. The forward-inverse model and the skip network were trained independently using Adam (learning rates $\eta_\phi = 0.0005$ and $\eta_\theta = 0.0001$, $\epsilon = 10^{-4}$) and gradient norm clipping (max= 0.1). The code for running our experiments can be found at <https://github.com/CognitiveModeling/HierarchicalGateLORD>.

D.2 Additional Experiments and Analysis

In Sec. 6.3.3 we showed that our system learns meaningful temporal abstractions for future hand positions. How are the skip predictions for the other two entities?

We exemplify the skip predictions for the position of the object and the goal as before by feeding the scripted sequence of $\mathcal{D}_{\text{script}}$ into the forward-inverse model to generate inputs for the skip network. Figure D.1 (a)-(c) shows the skip predictions for three exemplary sequences based on the inputs $(\mathbf{o}_2, \mathbf{h}_2)$. For all sequences, the skip network predicts that the object position (green) and the goal position (red) will not change (cf. Fig. 6.6 for hand predictions and the ground truth goal positions). This is reasonable for the given events. However, for reach-grasp-transport sequences, the object position changes once

the object is grasped (typically $t \leq 12$). As shown in exemplary fashion in Fig. D.1d, if we use $(\mathbf{o}_{12}, \mathbf{h}_{12})$ of a reach-grasp-transport sequence, the skip network actually predicts that the object will be close to the goal at the next event boundary.

E

THICK World Models: Supplementary Material^{E.1}

E.1 Pseudocode

Algorithm 2 outlines how THICK world models make temporal abstract predictions using both levels of the hierarchy (also visualized in Fig. 7.4). Blue parts are only needed for MBRL or MPC (see Sec. 7.3). For temporal abstract rollouts, which are used in THICK PlaNet, the process can be repeated K times by using the output states, i.e. $\mathbf{c}_{\tau(t)}$ and $\hat{\mathbf{z}}_{\tau(t)}^c$, as inputs again.

Algorithm 3 describes how to create input-target data for training the high-level world model. In continual learning environments with no early termination of an episode, we omit the red part.

Algorithm 4 describes the general main training and generation of behavior THICK world models. Red parts are only used for THICK PlaNet. Blue parts are only used for THICK Dreamer. In our zero-shot planning experiments using THICK PlaNet, we do not add new data to the replay buffer and only plan and execute actions during evaluation.

^{E.1}This chapter is based on the following manuscript, accepted for publication:

Gumbsch, C., Sajid, N., Martius, G. & Butz, M. V. (in press, 2024). Learning Hierarchical World Models with Adaptive Temporal Abstractions from Discrete Latent Dynamics. *The Twelfth International Conference on Learning Representations (ICLR 2024)*.

Algorithm 2 THICK Temporal Abstract Prediction

- 1: **input:** context \mathbf{c}_t , stochastic state \mathbf{z}_t
 - 2: $\hat{\mathbf{A}}_t \sim P_\theta(\hat{\mathbf{A}}_t \mid \mathbf{c}_t, \mathbf{z}_t)$ ▷ sample high-level action
 - 3: $\hat{\mathbf{z}}_{\tau(t)-1} \sim F_\theta(\hat{\mathbf{z}}_{\tau(t)-1} \mid \hat{\mathbf{A}}_t \mathbf{c}_t, \mathbf{z}_t)$ ▷ high-level state prediction
 - 4: $\hat{\mathbf{a}}_{\tau(t)-1} \sim F_\theta(\hat{\mathbf{a}}_{\tau(t)-1} \mid \hat{\mathbf{A}}_t \mathbf{c}_t, \mathbf{z}_t)$ ▷ high-level action prediction
 - 5: $\Delta\hat{\tau}(t) \sim F_\theta(\Delta\hat{\tau}(t) \mid \hat{\mathbf{A}}_t \mathbf{c}_t, \mathbf{z}_t)$ ▷ high-level time prediction
 - 6: $(\hat{r}_{t:\tau(t)}^\gamma \sim F_\theta(\hat{r}_{t:\tau(t)}^\gamma \mid \hat{\mathbf{A}}_t \mathbf{c}_t, \mathbf{z}_t)$ ▷ high-level reward prediction
 - 7: $\mathbf{c}_{\tau(t)} \leftarrow g_\phi(\hat{\mathbf{a}}_{\tau(t)-1}, \mathbf{c}_t, \hat{\mathbf{z}}_{\tau(t)-1})$ ▷ low-level context
 - 8: $\hat{\mathbf{z}}_{\tau(t)}^c \sim p_\phi^c(\hat{\mathbf{z}}_{\tau(t)-1}^c \mid \hat{\mathbf{a}}_{\tau(t)-1}, \mathbf{c}_{\tau(t)}, \hat{\mathbf{z}}_{\tau(t)-1})$ ▷ low-level coarse prior
 - 9: $\hat{r}_{\tau(t)}^c, \hat{\gamma}_{\tau(t)}^c \sim o_\phi^c(\hat{r}_{\tau(t)}^c, \hat{\gamma}_{\tau(t)}^c \mid \mathbf{c}_{\tau(t)}, \hat{\mathbf{z}}_{\tau(t)}^c)$ ▷ coarse reward & discount prediction
 - 10: **output:** $\mathbf{c}_{\tau(t)}, \hat{\mathbf{z}}_{\tau(t)}^c, \widehat{\Delta t}, \hat{r}_{t_\tau}^\gamma, \hat{r}_{\tau(t)}^c, \hat{\gamma}_{\tau(t)}^c$
-

Algorithm 3 THICK Training Data Generation

- 1: **input:** discount factor γ , sequences of contexts $\mathbf{c}_{1:T}$, stochastic states $\mathbf{z}_{1:T}$,
 - 2: actions $\mathbf{a}_{1:T}$, rewards $r_{1:T}$, and episode termination flags $d_{1:T}$
 - 3: **initialize:** train data $\mathcal{D} \leftarrow \{\}$, unassigned inputs $\mathcal{I} \leftarrow \{\}$
 - 4: **for** $\tau \leftarrow 1$ to T **do**
 - 5: **if** $\mathbf{c}_\tau \neq \mathbf{c}_{\tau-1}$ **or** $d_\tau = 1$ **then** ▷ context change or episode is over at time τ
 - 6: **for** $(\mathbf{c}_t, \mathbf{z}_t) \in \mathcal{I}$ **do**
 - 7: compute passed time $\Delta\tau \leftarrow \tau - t$
 - 8: compute accumulated rewards $r_{t:\tau} \leftarrow \sum_{\delta=1}^{\Delta\tau-1} \gamma^\delta r_{t+\delta}$
 - 9: add input-target tuple $((\mathbf{c}_t, \mathbf{z}_t), (\mathbf{z}_{\tau-1}, \mathbf{a}_{\tau-1}, \Delta t, r_{t:\tau}))$ to \mathcal{D}
 - 10: remove $(\mathbf{c}_t, \mathbf{z}_t)$ from \mathcal{I}
 - 11: add potential input $(\mathbf{c}_\tau, \mathbf{z}_\tau)$ to \mathcal{I}
 - 12: **output:** train data \mathcal{D}
-

Algorithm 4 THICK World Models

```
1: initialize neural networks and replay buffer
2:  $t^{\text{plan}} = -I$ 
3: for  $t \leftarrow 1$  to  $t^{\text{end}}$  do
4:   update low-level world model state  $\mathbf{s}_t \sim w_\phi(\mathbf{s}_t \mid \mathbf{s}_{t-1}, \mathbf{a}_{t-1})$ 
5:   // Behavior
6:   if  $\mathbf{c}_t \neq \mathbf{c}_{t-1} \wedge t \geq t^{\text{plan}} + I$  then
7:     plan subgoal  $\mathbf{z}_t^{\text{goal}}$  using MCTS and temporal abstract rollouts (Alg. 2)
8:      $t^{\text{plan}} \leftarrow t$ 
9:     plan new action  $\mathbf{a}_t$  using CEM given  $\mathbf{s}_t$  and  $\mathbf{z}_t^{\text{goal}}$  (Eq. 7.26)
10:    sample new action  $\mathbf{a}_t$  from actor  $\pi$  given  $\mathbf{s}_t$ 
11:    execute action  $\mathbf{a}_t$  in environment and observe  $\mathbf{r}_t, \mathbf{i}_t$  and  $d_t$ 
12:    add  $(\mathbf{i}_t, \mathbf{a}_t, r_t, d_t)$  to replay buffer
13:    // Train world models
14:    draw sequence batch  $\mathcal{B} \leftarrow (\mathbf{i}_{t':T}, \mathbf{a}_{t':T}, r_{t':T}, d_{t':T})$  from replay buffer
15:    embed batch in latent state  $\mathbf{s}_{t':T} \sim w_\phi(\mathbf{s}_{t':T} \mid \mathcal{B})$ 
16:    update low-level world model  $w_\phi$  using  $\mathcal{B}$  (Eq. 7.9)
17:    generate high-level training batch  $\mathcal{D}$  from  $(\mathbf{s}_{t':T}, \mathbf{a}_{t':T}, r_{t':T}, d_{t':T})$  (Alg. 3)
18:    update high-level world model  $W_\theta$  using  $\mathcal{D}$  (Eq. 7.19)
19:    // Train actor and critic
20:    imagine trajectory  $(\mathbf{s}_{t'':H}, \mathbf{a}_{t'':H}, r_{t'':H}, \gamma_{t'':H})$  using  $w_\phi$  from random start  $s_{t''} \in \mathcal{B}$ 
21:    make temporal abstract predictions for each  $\mathbf{s}_{t'':H}$  using  $W_\theta$  and  $w_\phi$  (Alg. 2)
22:    compute value  $V$  (Eq. 7.23)
23:    update critics  $v_\chi$  and  $v_\xi$  (Eq. 7.24)
24:    update actor  $\pi$ 
```

E.2 Hyperparameters

Table E.1: Hyperparameter choices. If there is only one centered value it counts for all suites. Otherwise different values are chosen for MiniHack (MH), VisualPinPad (VPP), or Multworld (MW).

Name	MH	Value VPP	MW
Low-Level World Model (C-RSSM or RSSM)			
Batches (size \times sequence length)		16×50	
Dimensions of \mathbf{c}_t		16	
Dimensions of \mathbf{h}_t		256	
Dimensions of \mathbf{z}_t		32×32	
MLP features per layer		256	
Sparsity loss scale β^{parse}	$1^{\text{E.2}}/10$	1	25
Prediction loss scale β^{pred}		1	
KL loss scale β^{KL}		1	
KL balancing β^{bal}		0.8	
Output heads o_ϕ for	i_t, γ_t, r_t	i_t, r_t	i_t, r_t
Prioritize ends in replay	yes	no	no
Learning rate		0.0001	
High-Level World Model			
Q_θ & P_θ number of layers \times features		3×200	
F_θ number of layers \times features		5×1024	
Number of actions \mathbf{A}_t	3	5	5
Use terminations d_t for segmentation	yes	no	no
Loss for training $F_\theta^{\hat{a}}(\hat{\mathbf{a}}_{\tau(t)-1} \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t)$	CCE	CCE	NLL
Action prediction loss scale $\zeta^{\mathbf{a}_{\tau(t)-1}}$	1	1	0.1
State prediction loss scale $\zeta^{\mathbf{z}_{\tau(t)-1}}$		1	
Time prediction loss scale $\zeta^{\Delta\tau(t)}$	1	1	0.1
Reward prediction loss scale $\zeta^{\tau_{t:\tau(t)}^\gamma}$		1	
KL balancing ζ^{bal}		0.8	
Learning rate		0.0001	
Thick Dreamer			
Imagination horizon H	15	15	
Value estimate balance ψ	0.9	0.9	
λ -target of V_t^λ	0.95	0.95	
Long-horizon critic v_χ layers \times features	4×400	4×400	
Long-horizon critic v_χ learning rate	0.0002	0.0002	
Thick PlaNet			
CEM planning horizon H			12
Long-horizon scale κ			0.025
MCTS simulations			100
MCTS discount			0.997
Common			
Optimizer		Adam	
MLP activation functions		ELU	
Discount γ		0.99	

^{E.2}Unlike the other MiniHack problems, MiniHack-Corridor tasks are fully deterministic, which is why we use a lower factor of $\beta^{\text{parse}} = 1$ here.

World model learning hyperparameters For optimizing the world models, i.e. our THICK world models and the baseline models in Dreamer and Director, we use the default DreamerV2 hyperparameters (Hafner et al., 2020a), except for minor variations. Specifically, we reduced the size of the model by setting the feature size of the RSSM and the dimensionality of \mathbf{h}_t to 256. As we show in Suppl. E.5.7 model size does not strongly affect performance in our setting. Additionally, for THICK Dreamer and Dreamer we did not employ layer normalization for the GRU within the RSSM, because in pre-tests this showed increased robustness for both approaches.

MBRL hyperparameters For training the actor and critic in THICK Dreamer and Dreamer we use the default hyperparameters of DreamerV2. For Director we mostly used its default hyperparameters (Hafner et al., 2022), however, we made some minor adjustments to the training frequency to ensure a fair comparison. Director performs one training update every 16 policy steps instead of every 5 steps in DreamerV2. This was done to reduce wall-clock time but decreases sample efficiency (Hafner et al., 2022). We increase the update frequency (16 \rightarrow 5) in order to fairly compare sample efficiency between approaches.

MPC hyperparameters For MPC with CEM we use the hyperparameters of PlaNet (Hafner et al., 2019b). For high-level planning with MCTS, we use MuZero’s (Schrittwieser et al., 2020) implementation, with mostly the same hyperparameters. However, intuitively we would not expect multiple predictions to reach a goal. Thus, we decrease the number of simulations to $S = 100$.

Differences between environments The main difference between MiniHack and the other environments is that in MiniHack episodes can terminate based on the success of the task or the death of the agent. VisualPinPad and Multiworld are continual learning environments without early episode termination. As is customary with the use of DreamerV2, for environments that do not feature early episode termination, we do not predict discounts γ_t , nor do we prioritize sampling subsequences around episode termination from the replay buffer. Additionally, we do not treat episode terminations as context changes. For action prediction, we use the Categorical Cross Entropy Loss (CCE) for predicting discrete actions (Minihack and VisualPinPad), and scale down the high-level prediction loss for predicting actions and elapsed time when training purely on task-free offline data (Multiworld). Lastly, the sparsity loss scale β^{sparse} was tuned for each suite.

Hyperparameter search For determining the sparsity loss scale β^{sparse} , the value estimate balance ψ , and the long-horizon planning scale κ , we ran a grid search using three random seeds and using two tasks of each suite (MiniHack: KeyRoom-Fixed-S5, WandOfDeath-Advances; MiniHack-Corridor: KeyCorridor-4, KeyCorridor-8, Visual Pin Pad: VisualPinPadFour, VisualPinPadFive; Multiworld: Door, PickUp). We determined the best hyperparameter value for each suite depending on task performance

and a qualitative inspection of the high-level predictions (see Suppl. E.5.8). For simplicity and to demonstrate robustness, we used the same values for each suite, whenever it was reasonable.

How to tune When tuning THICK world models for a new task, we recommend mainly searching over the sparsity loss scale $\beta^{\text{sparse}} \in \{1, 5, 10, 25, 50\}$. Typically, one random seed is sufficient to determine which β^{sparse} leads to few, but not too few, context changes.

E.3 THICK World Models: Implementation Details

E.3.1 THICK Design Choices

Here, we explain some design choices in more detail here.

High-level targets Our goal is to learn a high-level world model that predict situations in which latent generative factors are assumed to change, e.g. a door openings or object manipulations. In addition to that, we want to use high-level outputs to predict future rewards and reconstruct images at some time $\tau(t)$. Thus, we at least need the context $\mathbf{c}_{\tau(t)}$ and the stochastic state $\mathbf{z}_{\tau(t)}$ to make these reconstructions via the coarse processing pathway (see Eq. 7.8). There are two potential ways to predict $\mathbf{c}_{\tau(t)}$ and $\mathbf{z}_{\tau(t)}$, either predict the state *before* or *after* the context transition.

We predict the states *before* the context transition. Our main reasoning is that the prediction of context-altering situations presents two challenges: (1.) learning in which situation such transitions are currently possible and (2.) how these transitions affect the latent generative factors. The C-RSSM already learns to encode *ii*). Therefore, to reduce redundancy and simplify the challenge, we train the high-level model to only learn (1.) and then prompt the low-level model for (2.). One example from MiniHack would be predicting the agent standing in front of closed door and performing the a door-opening-action. We believe this is a simpler prediction compared to predicting the ego-centric view of the agent after opening a door and looking into a (potentially unknown) new room.

Coarse predictions for contextual learning We want the context \mathbf{c}_t to encode latent information that is necessary for prediction and reconstruction. If we omit the coarse prior predictions (Eq. 7.4) and coarse output reconstructions (Eq. 7.8) the C-RSSM would have no incentive to encode prediction-relevant latent information in \mathbf{c}_t . Instead, it could purely utilize \mathbf{h}_t and avoid a sparsity penalty in Eq. 7.9 via $\mathcal{L}^{\text{sparse}}$ by never updating \mathbf{c}_t . Completely omitting \mathbf{h} in the C-RSSM impedes learning, as we show in our ablations in Suppl. E.5.8. Thus, we instead add the coarse processing pathway. Via the coarse predictions, the C-RSSM needs to encode latent factors in \mathbf{c}_t in order to reduce the KL-loss (Eq. E.8) and prediction loss (Eq. E.7).

Coarse predictions to omit \mathbf{h}_t The high-level model attempts to predict a future

state of the system. The full latent state would contain the deterministic component \mathbf{h} . However, for the high-level model it would be very challenging to predict the unregularized high-dimensional deterministic hidden state \mathbf{h} many time steps in the future. The coarse pathway of the C-RSSM allows updates of the context dynamics \mathbf{c}_t , predictions of the stochastic states \mathbf{z}_t^c , and reconstructions of external variables without the deterministic hidden state \mathbf{h}_t . Thus, it is advantageous that the C-RSSM can make predictions without \mathbf{h} . After a high-level prediction, we can feed the high-level outputs $(\hat{\mathbf{z}}_{\tau(t)-1}, \hat{\mathbf{a}}_{\tau(t)-1})$ into the low-level world model. This brings many advantages: For example, this allows us to predict rewards or discounts/episode termination at particular states after a high-level prediction, which we use in THICK Dreamer in and THICK PlaNet (see Sec. 7.3). Furthermore, we can reconstruct images to visualize predictions as shown in Sec. 7.4.2. Additionally, we can continue with low-level rollouts after a high-level prediction, which is a feature we have not yet utilized.

E.3.2 GATELORD

We want the context code \mathbf{c}_t to change only sparsely in time. Thus, we implement the discrete context dynamics g_ϕ as a GATELORD cell (details in Chap. 5). For the sake of completeness, we briefly review GATELORD and our slightly changed notation here. GATELORD is an RNN designed to maintain sparsely changing latent states \mathbf{c}_t .^{E.3} To realize this inductive bias, GATELORD uses two subnetworks g_ϕ^p and g_ϕ^g that control \mathbf{c}_t updates via an internal update gate $\mathbf{\Lambda}_t$. GATELORD can be summarized as

$$\hat{\mathbf{c}}_t = g_\phi^p(\mathbf{a}_{t-1}, \mathbf{c}_{t-1}, \mathbf{z}_{t-1}) \quad (\text{candidate proposal}) \quad (\text{E.1})$$

$$\mathbf{\Lambda}_t = g_\phi^g(\mathbf{a}_{t-1}, \mathbf{c}_{t-1}, \mathbf{z}_{t-1}) \quad (\text{update gate}) \quad (\text{E.2})$$

$$\mathbf{c}_t = \mathbf{\Lambda}_t \odot \hat{\mathbf{c}}_t + (1 - \mathbf{\Lambda}_t) \odot \mathbf{c}_{t-1} \quad (\text{context update}) \quad (\text{E.3})$$

with \odot denoting the Hadamard product. We use the action \mathbf{a}_{t-1} and the last stochastic state \mathbf{z}_{t-1} as the cell inputs. Based on this cell input and the last context \mathbf{c}_{t-1} , GATELORD proposes a new context $\hat{\mathbf{c}}_t$ via its proposal subnetwork g_ϕ^p (Eq. E.1). Whether the context is updated depends on an update gate $\mathbf{\Lambda}_t \in [0, 1]^m$ (Eq. E.3). This update gate $\mathbf{\Lambda}_t$ is the output of the gating subnetwork g_ϕ^g (Eq. E.2) which uses a rectified tanh activation function (ReTanh), with $\text{ReTanh}(x) := \max(0, \tanh(x))$. This ensures that the gate activations are $\in [0, 1]^m$. Note that to compute $\mathbf{\Lambda}_t$, the subnetwork g_ϕ^g internally samples from a Gaussian distribution before applying the ReTanh function to improve robustness. Thus, the context updates are a stochastic process.

^{E.3}In previous chapters, we referred to the latent state of GATELORD, as well as to other RNNs' latent states as \mathbf{h}_t . Since C-RSSM maintains a GRU latent state as well as the latent state of GATELORD, we now refer to the latter as \mathbf{c}_t to distinguish the two.

Originally, GATELORD used a subnetwork to compute the cell output using multiplicative gating. We omit this here and instead feed the output to the GRU cell f_ϕ as shown in Fig. 7.2 (right).

The centralized gate Λ_t of GATELORD makes it easy to determine context changes, i.e. $\mathbf{c}_t \neq \mathbf{c}_{t-1}$. Since all context updates depend on Λ_t , we know that the context changed if $\Lambda_t > \mathbf{0}$. This is an advantage over other RNNs that use multiple gates for sparsely changing latent states. We use this measure to determine context changes when building the world model hierarchy.

E.3.3 C-RSSM Loss

The loss of the C-RSSM (Eq. 7.9) is composed of three parts: the prediction loss $\mathcal{L}^{\text{pred}}$, the KL loss \mathcal{L}^{KL} , and the sparsity loss $\mathcal{L}^{\text{sparse}}$. Except for the sparsity loss, we adapt these loss terms from the RSSM. However we always need to account for the coarse prediction pathways of the C-RSSM.

We define the prediction loss $\mathcal{L}^{\text{pred}}$ as

$$\mathcal{L}^{\text{pred}}(\phi) = \frac{1}{T} \sum_{t=1}^T \left[\sum_{y \in \{\mathbf{i}_t, r_t, \gamma_t\}} -\log o_\phi(y | \mathbf{s}_t) - \log o_\phi^c(y | \mathbf{c}_t, \mathbf{z}_t) \right]. \quad (\text{E.4})$$

Equations in red are exclusive to the C-RSSM. Thus, the network is trained to minimize the negative log likelihood for predicting the images \mathbf{i}_t , rewards r_t and future discounts γ_t . Here, we account for both precise predictions over the output heads o_ϕ (Eq. 7.7), and coarse predictions over the output heads o_ϕ^c (Eq. 7.8). Following the DreamerV2 codebase (Hafner et al., 2020a), in continual learning environments when there is no early episode termination, we do not predict the discount γ_t , and instead use a fixed discount $\gamma = 0.99$.

The C-RSSM predicts two prior distributions for the next stochastic state $\hat{\mathbf{z}}_t$: fine predictions using the full state (Eq. 7.5) and coarse predictions based only on the context, last action and stochastic state (Eq. 7.4). We need to account for both types of prediction in the KL loss \mathcal{L}^{KL} with

$$\begin{aligned} \mathcal{L}^{\text{KL}}(\phi) = \frac{1}{T} \sum_{t=1}^T & \text{KL} \left[q_\phi(\mathbf{z}_t | \mathbf{h}_t, \mathbf{i}_t) || p_\phi^h(\hat{\mathbf{z}}_t | \mathbf{h}_t) \right] \\ & + \text{KL} \left[q_\phi(\mathbf{z}_t | \mathbf{h}_t, \mathbf{i}_t) || p_\phi^c(\hat{\mathbf{z}}_t^c | \mathbf{a}_{t-1}, \mathbf{c}_t, \mathbf{z}_{t-1}) \right]. \end{aligned} \quad (\text{E.5})$$

Thus, we want to minimize the divergence between both the fine prior p_ϕ^h and the approximate posterior q_ϕ , as well as the divergence between the coarse prior p_ϕ^c and q_ϕ . As in DreamerV2 (Hafner et al., 2020a), we use KL-balancing, which scales the loss

contribution of the prior p_ϕ of each KL divergence by a factor $\beta^{\text{bal}} = 0.8$, and of the posterior q_ϕ by $1 - \beta^{\text{bal}}$. This enables faster learning of the prior to avoid that the posterior is regularized towards an untrained prior.

We take the sparsity loss $\mathcal{L}^{\text{sparse}}$ from GATELORD (see Sec. 5.3) which is an L_0 -regularization of the context changes $\Delta \mathbf{c}_t$. This is implemented as

$$\mathcal{L}^{\text{sparse}}(\phi) = \frac{1}{TJ} \sum_{t=1}^T \sum_{j=1}^J \|\Delta c_t^j\|_0 = \frac{1}{TJ} \sum_{t=1}^T \sum_{j=1}^J \Theta(\Lambda_t^j) \quad (\text{E.6})$$

where J is the dimensionality of the context \mathbf{c}_t and $\Theta(\cdot)$ denotes the Heaviside step function. That is, an L_0 -regularization of the context changes is implemented as the binarization of the update gates Λ_t (Eq. E.3). We estimate the gradient of the Heaviside step function using the straight-through estimator (Bengio et al., 2013).

E.3.4 High-level World Model Training

The high-level world model with parameters θ is trained to minimize both the prediction loss $\mathfrak{L}^{\text{pred}}$, of predicting the next context change state, and the action loss $\mathfrak{L}^{\mathbf{A}}$, which is the divergence of the prior and posterior high-level action distributions.

For every high-level target $Y \in \{\mathbf{a}_{\tau(t)-1}, \mathbf{z}_{\tau(t)-1}, \Delta\tau(t), r_{t:\tau(t)}^\gamma\}$, we use an appropriate prediction loss and use a weighted sum in $\mathfrak{L}^{\text{pred}}$. For the to-be-predicted action $\mathbf{a}_{\tau(t)-1}$, the time passed $\Delta\tau(t)$, and the rewards $r_{t:\tau(t)}^\gamma$, we use the negative log-likelihood (NLL). For the prediction of stochastic state $\mathbf{z}_{\tau(t)-1}$, we know the underlying distribution that generated the target variable, i.e. the posterior $q_\phi(\cdot)$. Thus, we can use the KL divergence as a loss for high-level state predictions. Overall, we get

$$\begin{aligned} \mathfrak{L}^{\text{pred}}(\theta) = \frac{1}{T} \sum_{t=1}^T \left[\sum_{Y \in \{\mathbf{a}_{\tau(t)-1}, \Delta\tau(t), r_{t:\tau(t)}^\gamma\}} -\zeta^Y \log F_\theta(Y | \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \right. \\ \left. + \zeta^z \text{KL} \left[q_\phi(\mathbf{z}_{\tau(t)-1} | \mathbf{c}_{\tau(t)-1}, \mathbf{h}_{\tau(t)-1}, \mathbf{i}_{\tau(t)-1}) || F_\theta^{\hat{\mathbf{z}}}(\hat{\mathbf{z}}_{\tau(t)-1} | \mathbf{A}_t, \mathbf{c}_t, \mathbf{z}_t) \right] \right]. \end{aligned} \quad (\text{E.7})$$

Hyperparameters $\zeta^Y \in \{\zeta^{\mathbf{a}}, \zeta^z, \zeta^{\Delta\tau(t)}, \zeta^{r^\gamma}\}$ can be used to scale individual loss terms. By default, we set $\zeta^Y = 1$ for all loss terms. When training the network on task-free exploration, i.e. during zero-shot MPC as described in Sec. 7.4.4, we found that predicting actions $\mathbf{a}_{\tau(t)-1}$ at context changes and the elapsed time $\Delta\tau(t)$ was challenging. To mitigate this, during task-free exploration, we set $\zeta^{\mathbf{a}_{\tau(t)-1}} = 0.1$ and $\zeta^{\Delta\tau(t)} = 0.1$.

The action loss $\mathfrak{L}^{\mathbf{A}}$ drives the system to minimize the divergence between the posterior high-level action distribution $Q_\theta(\mathbf{A}_t | \mathbf{c}_t, \mathbf{z}_t, \mathbf{c}_{\tau(t)}, \mathbf{z}_{\tau(t)})$, and the prior distribution $P_\theta(\hat{\mathbf{A}}_t |$

$\mathbf{c}_t, \mathbf{z}_t$) with

$$\mathcal{L}^{\mathbf{A}}(\theta) = \frac{1}{T} \sum_{t=1}^T \text{KL} \left[Q_{\theta}(\mathbf{A}_t \mid \mathbf{c}_t, \mathbf{z}_t, \mathbf{c}_{\tau(t)}, \mathbf{z}_{\tau(t)}) \parallel P_{\theta}(\hat{\mathbf{A}}_t \mid \mathbf{c}_t, \mathbf{z}_t) \right]. \quad (\text{E.8})$$

Like the KL loss \mathcal{L}^{KL} on the low level (see Suppl. E.3.3), we use KL balancing (Hafner et al., 2020a) to scale the prior part by $\zeta^{\text{bal}} = 0.8$ and the posterior part by $1 - \zeta^{\text{bal}}$.

E.3.5 THICK Dreamer: Details

THICK Dreamer estimates the overall value $V(\mathbf{s}_t)$ of a state \mathbf{s}_t as a mixture of short- and long-horizon estimates (Eq. 7.23) using critics v_{ξ} and v_{χ} , respectively. Like DreamerV2, we stabilize critic training by using a copy of the critics during value estimation (in 7.20 and Eq. 7.22). The copy is updated every 100 updates.

E.3.6 THICK PlaNet: Details

For planning on the high level we use a MCTS implementation based on MuZero (Schrittwieser et al., 2020). We only replan on the high level if the context changes, i.e. $\mathbf{c}_t \neq \mathbf{c}_{t-1}$. Since all subgoals $\mathbf{z}_t^{\text{goal}}$ are situations that lead to context changes, no additional criterion for subgoal completion is needed. Upon reaching a subgoal, e.g. touching an object, the context can sometimes change for multiple subsequent time steps. This causes the high-level to replan multiple times in a row. To avoid a high computational cost and to allow smoother trajectories, we inhibit replanning for $I = 3$ time steps after setting a new subgoal. While this could potentially degrade performance in dynamic environments, we found this to work well in Multiworld.

E.4 Environment Details

E.4.1 MiniHack

Here we provide a detailed explanation of all MiniHack problems we considered. In all settings, we restricted the action space to the minimum number of actions needed to solve the task. In all tasks the agent receives a sparse reward of 1 when exiting the room and a small punishment of -0.01 when performing an action that has no effect, e.g. moving against a wall. In the easier tasks (WaterCrossing-Ring, KeyRoom-Fixed-S5, KeyCorridor) the agent is allowed 200 time steps to solve the task. In all other tasks the time limit is set to 400 time steps. For aesthetic reasons, we use different characters in different levels.

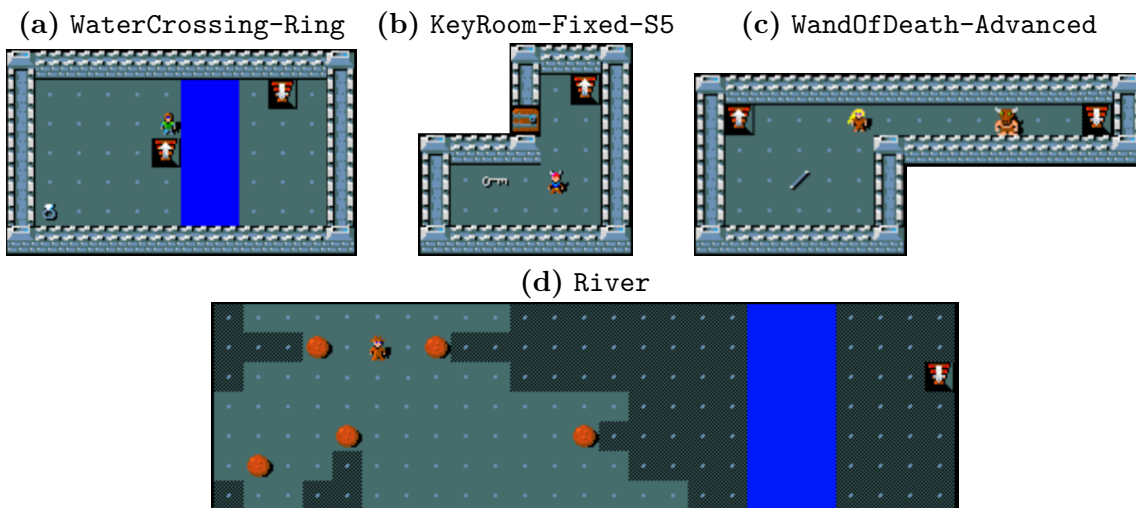


Figure E.1: MiniHack environments: Staircases with an upwards facing arrows mark the starting point of the agents. Staircases with downward facing arrows are the exits that need to be reached. In (a), (b), and (d) start points and exits are randomized.

`WaterCrossing-Ring` is a newly designed, simple level in which an agent needs to fetch a randomly placed ring of levitation and float over a river to get to the goal (Fig. E.1a). When a ring is picked up in our tasks, it is automatically worn.^{E.4} The level is inspired by `LavaCross-Levitate-Ring-PickUp` from the MiniHack benchmark suite, where a river of deadly lava blocks the exit. However, we found that Dreamer struggles to learn this task because of the early terminations when entering the lava.

`KeyRoom-Fixed-S5` is a benchmark task, in which an agent spawns in a room at a random position and has to fetch a randomly placed key to open a door and enter a smaller room with a randomly located exit (Fig. E.1b). The door position is fixed. In all our tasks, using the key opens the door from any grid cell adjacent to the door, even diagonally.

`KeyCorridor-N` is a novel task, in which an agent starts in front of a locked door in the top left corner of a corridor. In the bottom right corner of the corridor is the key for the door. We vary the length N of the corridor to systematically manipulate the task horizon.

`WandOfDeath-Advanced` is based on the `WandOfDeath` benchmark tasks, in which an exit is guarded by a minotaur, which instantly kills the agent upon contact. The agent needs

^{E.4}Usually, to wear a ring in MiniHack a sequence of actions needs to be performed: `PUTON` \rightarrow `RING` \rightarrow `RIGHT`, for putting the ring on the right finger. We simplify this by automatically applying the action sequence when the ring is picked up.

to pick up a wand to attack and kill the monster. Thereby, the agent needs to carefully select the direction of the attack, because if the attack bounces off a wall, it kills the agent instead. `WandOfDeath` comes in multiple levels of difficulty. `WandOfDeath-Advanced` (Fig. E.1c) is a self-created level layout, designed to be more challenging than `WandOfDeath-Medium` but not as difficult as `WandOfDeath-Hard`. In `WandOfDeath-Medium` the agent can only walk horizontally and the location of the wand is fixed. In `WandOfDeath-Hard` the map is very large, which makes this a hard exploration problem. Our version is of intermediate difficulty, where the number of accessible grids (28) is roughly the same as in `WandOfDeath-Medium` (27), while the randomly placed wand needs to be found first.

`River` is a benchmark task, in which an agent needs get to an exit on the other side of a river (Fig. E.1d). In order to cross the river the agent needs to push boulders into the water to form a small land bridge. To solve the task the agent needs to move at least two randomly placed boulders into the river.

`EscapeRoom` is a difficult new problem designed by us, which combines the challenges of many other problems (Fig. E.5a). Via `EscapeRoom` we test the ability to learn to execute a complex event sequence of five subgoals. However, the task can be learned without extensive exploration or large action spaces. The agent starts in a small room and the goal is to unlock a door and escape. However, in order to obtain the key, the agent must (1.) pick up a ring of levitation and (2.) float over a small patch of water into a corridor. In the corridor, the agent can (3.) exchange the ring of levitation for a key. To get back to the door in the first room, the agent must (4.) push a boulder into the water. Afterwards, the agent can (5.) unlock the door and exit the room. When levitating, the agent is too light to push the boulder. In `EscapeRoom`, the agent can only carry one item at the time, and picking up a second item results in dropping the first one.

E.4.2 Multiworld

In Multiworld we use tasks that have previously been used to study visual reinforcement learning (Nair et al., 2018; Pong et al., 2020). All tasks in Multiworld use different action spaces and camera viewpoints for their pixel-based observation, shown in Fig. E.2. In `Pusher` the 2-dimensional actions control the x - and y -movement of the endeffector, whereas the gripper is fixed. In `Door` the robot is equipped with a hook instead of a gripper and the 3-dimensional action controls x -, y -, and z -movement. In `PickUp` the 3-dimensional action controls the y - and z -movement and the gripper opening. We binarized the gripper opening to prevent accidental object drops. In all tasks, the goal positions are fixed. In `Pusher` and `PickUp` goals are visible in the video frames. In `Door` the goal is to fully open the door. For `Pusher-Dense` and `PickUp` we compute the reward

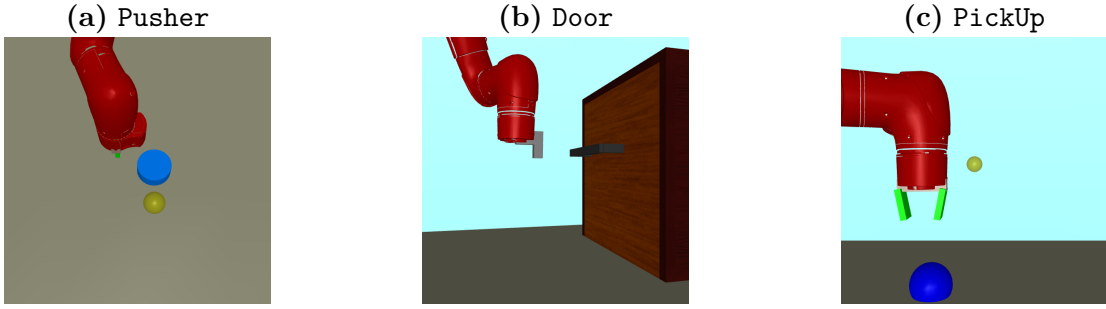


Figure E.2: Multiworld environments: Goal positions for the objects are shown in yellow.

r_t for every time step t as

$$r_t = 1 - \frac{\delta_t}{\delta_1}, \quad (\text{E.9})$$

where δ_t is the Euclidean distance between object and goal at time t . For **Pusher-Sparse** the agent received a reward of $r_t = 1$ when the distance between puck and goal $\delta_t < 0.025$, otherwise $r_t = 0$. For **Door** the reward r_t is the current angle of the door joint.

E.5 Extended Results and Experiment Details

E.5.1 Analysis of Contexts and Predictions

In this section, we provide further examples of high- and low-level predictions and context codes \mathbf{c}_t .

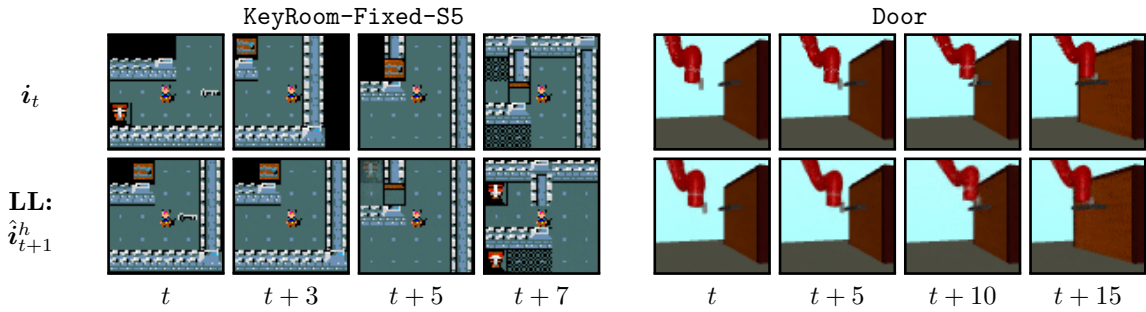


Figure E.3: Low-level predictions. The low-level predictions accurately predict the next frames (cf. Fig. 7.5 for high-level predictions and contexts \mathbf{c}_t of the same sequences).

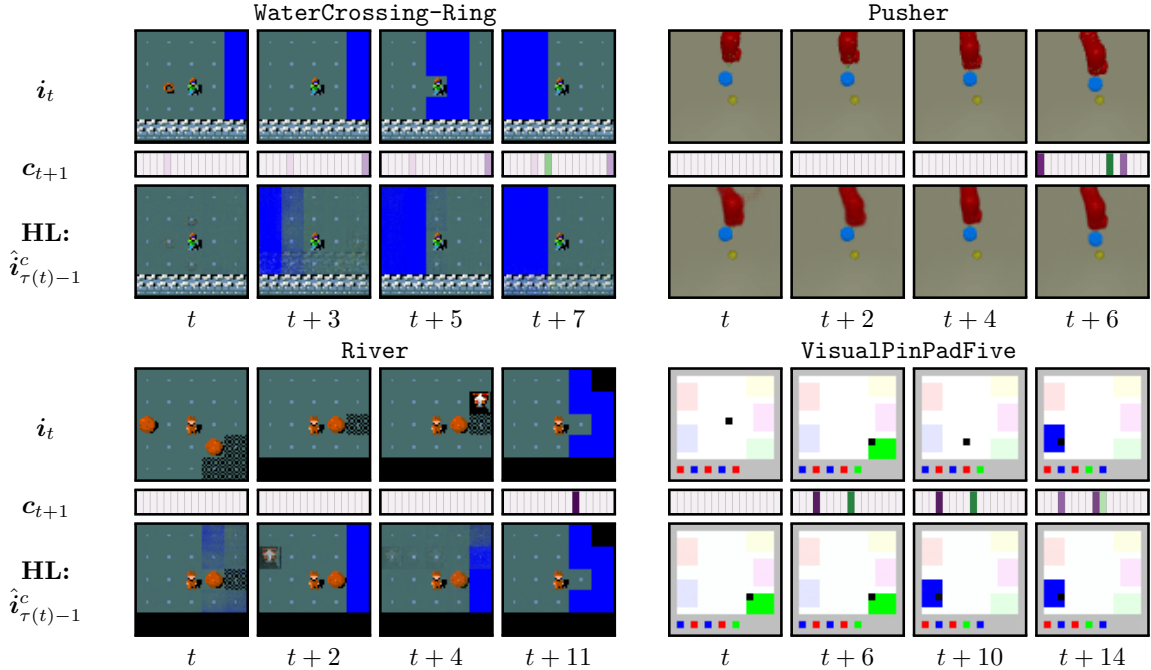


Figure E.4: Exemplary context changes. We show the input images i_t , 16-dim. contexts c_{t+1} and reconstructions $\hat{i}_{\tau(t)-1}^c$ of high-level predictions. Contexts are visualized relative to the first context c_t of the sequences, i.e. we plot $c_{t'} - c_t$ for every time t' .

C-RSSM predictions Figure E.3 visualizes the low-level predictions for two example sequences. The low-level world model predicts the immediate next state and its reconstructions are more accurate than the abstract high-level predictions (cf. Fig. 7.5).

Examples of context changes Figure E.4 displays four example sequences with the corresponding contexts c_t and high-level predictions. For *WaterCrossing-Ring* the context changes when stepping on the ring, picking it up, or arriving on the other side of the shore. In *Pusher* the context changes when the robot moves the puck. In *River* the context changes when pushing a boulder into water. In *VisualPinPadFive* the context changes when stepping on a pad.

High-level actions We analyze the high-level actions A_t in more detail for the *EscapeRoom* problem. *EscapeRoom* is a challenging MiniHack level, designed to contain diverse agent-environment interactions, shown in Fig. E.5a and described in detail in Suppl. E.4.1. To illustrate the emerging high-level action representations of THICK Dreamer, we show inputs i_t and image reconstructions $\hat{i}_{\tau(t)-1}^c$ and predicted low-level actions $\hat{a}_{\tau(t)-1}$ for all high-level actions A_t in Fig. E.5b for one exemplary sequence.

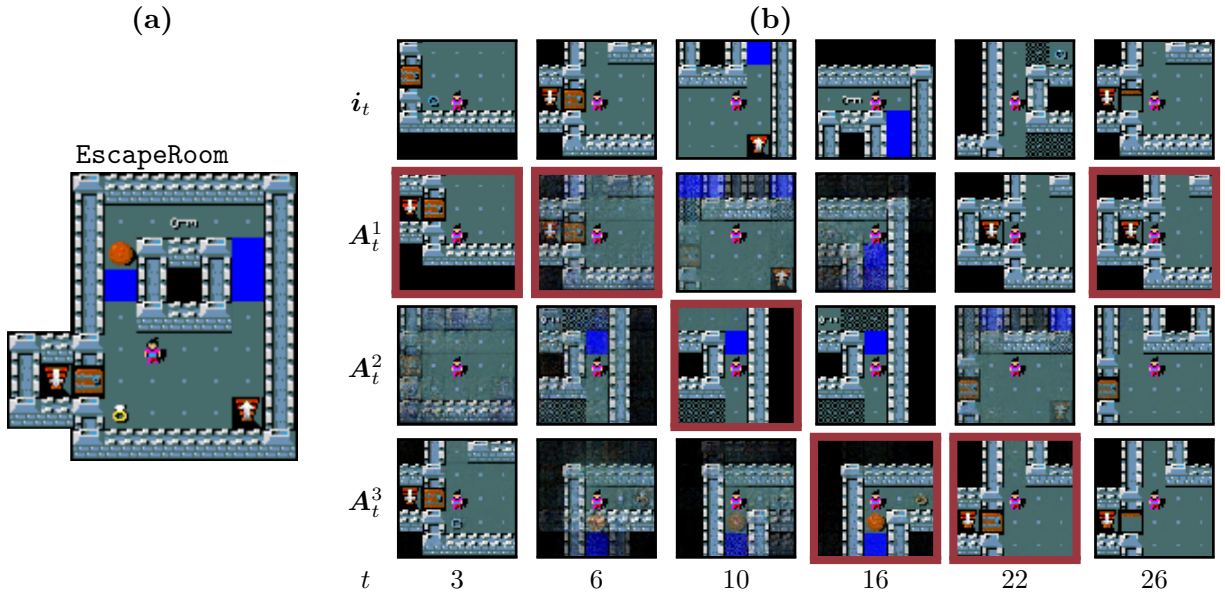


Figure E.5: Example sequence of high-level action predictions: (a) In EscapeRoom an agent needs to pick up a ring of levitation, hover over a patch of water to get to a key, exchange the ring for key, push a boulder into the water, and use the key to unlock a door. (b) Visualization of the high-level actions for one exemplary sequence. The top row shows the input image i_t . Image reconstructions $\hat{i}_{\tau(t)-1}^c$ and low-level action predictions $\hat{a}_{\tau(t)-1}$ are shown for all three high-level actions A_t . Red outlines depict which action \hat{A}_t was sampled.

At specific time steps, the three possible high-level actions A_t encode particular agent-environment interactions: A_t^1 encodes picking up the ring of levitation ($t = 3$) or exiting the level ($t \in \{22, 26\}$). A_t^2 encodes crossing the water after obtaining the ability to levitate ($t \in \{6, 10, 16\}$), either upwards ($t \in \{6, 10\}$) or downwards ($t = 16$). A_t^3 encodes pushing the boulder into water ($t \in \{6, 10, 16\}$) or walking in front of the door ($t = 22$). For all other time steps, the high-level actions produce identity predictions (e.g. A_6^1) or predictions that seem to encode average scene settings (cf. A_3^2 or A_{10}^1). These predictions account for unexpected context shifts, which can always occur with a small chance due to the randomness of sampling z_t and the stochastic update gates of GATELORD (see Suppl. E.3.2). The predicted low-level actions for these situations seem to be mostly random. The prior Q_ϕ (red frames and text in Fig. E.5b) typically samples reasonable high-level actions. However, occasionally the prior samples an action \hat{A}_t leading to an

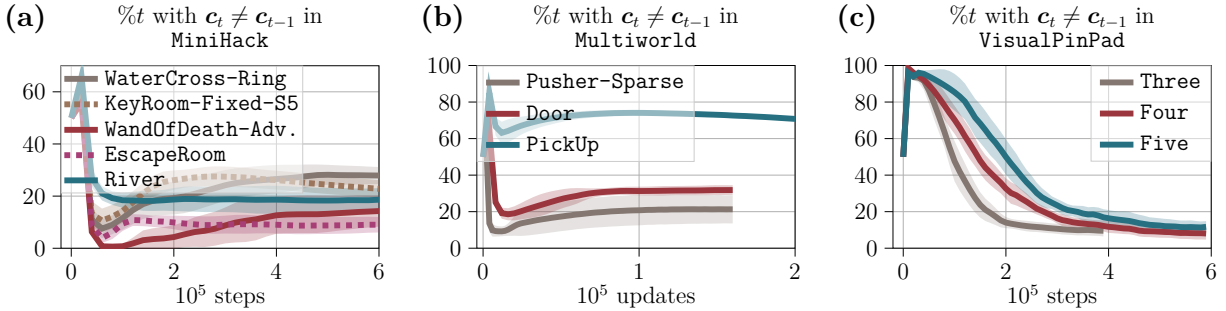


Figure E.6: Context changes over training: Each graphic plots the mean percentage of time steps per training batch for which the context \mathbf{c}_t changes in MiniHack (a), VisualPinPad (b), and Multiworld (c). Shaded areas depict the standard deviation.

identity or average prediction (e.g. $t = 6$) due to the randomness of the process.

Quantifying context changes To quantify the changes in context, we plot the mean percentage of time steps with context changes (i.e. $\mathbf{c}_t \neq \mathbf{c}_{t-1}$) over the course of training in Fig. E.6. Importantly, context changes are somewhat consistent within the same task, but, as expected, can vary between tasks in the same suite despite using the same hyperparameter β^{sparse} . Additionally, we analyze the time between context changes for some MiniHack tasks. We plot the histogram of time gaps between context changes in Fig. E.7, illustrating how different tasks show different distributions of context durations.

Task-relevance of contexts Lastly, we analyze whether context changes occur in task-relevant situations for some MiniHack problems. For this, we generated rollouts of the fully trained policy and identify points t^* , that we consider to be crucial for solving the task. For WandOfDeath-Adv., WaterCross-Ring, and KeyRoom-Fixed-S5 we take the time points t^* before picking up an item. For EscapeRoom, we use points in time t^* when the agent stands in front of a movable boulder blocking the path to the exit. We compute the mean percentage of context changes occurring around t^* (± 1 step) over 10 sequences and take the average over all 7 randomly seeded models. The results are shown in Table E.2. The C-RSSM tends to update its context with a high probability at the identified situations. This suggests that task-relevant aspects, such as item-pickups or boulder pushes, are encoded in the contexts.

E.5.2 MBRL in MiniHack: Experiment Details and Extended Results

In Fig. E.8 we plot the success rate of THICK Dreamer, DreamerV2, and Director for additional MiniHack tasks not shown in the main paper.

To investigate the effect of task horizon, we compare the performance gain of THICK

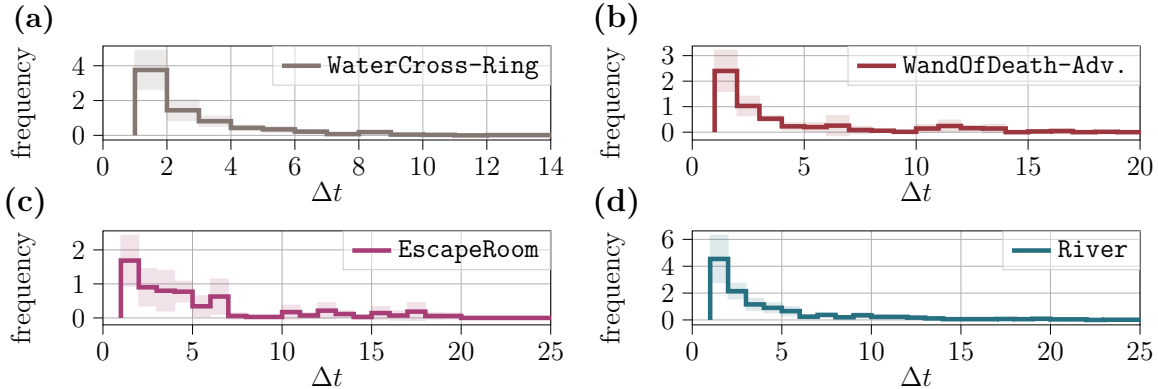


Figure E.7: Context duration: Each graphic plots a histogram of the mean number of time steps Δt between two consecutive context changes during an episode (over 10 episodes, max 50 steps) for different MiniHack tasks (7 seeds). Shaded areas depict the standard deviation.

Table E.2: Task-relevance of context changes. We list the mean percentage of context changes $c_{t^*} \neq c_{t^*-1}$ for fully trained policies (7 seeds) at crucial task-relevant times t^* in 10 sequences (\pm denotes standard deviation). See text for criterion of t^* .

	WaterCross	KeyRoom-Fixed-S5	WandOfDeath-Adv.	EscapeRoom
%	97.1 (\pm 4.9)	91.4 (\pm 6.9)	91.4 (\pm 1.5)	88.57 (\pm 15.7)

Dreamer over Dreamer for different corridor lengths in `KeyCorridor`. To compute improvements, we subtract the mean success rate and returns of Dreamer from the corresponding measure of THICK Dreamer (visualized in Fig. 7.7f–7.7g). For corridor lengths of 6 onward, the improvements of THICK Dreamer over Dreamer tend to increase with corridor length. However, for a corridor length of 11 most runs fail to discover the reward (see Fig. E.8g), which dampens the improvement in performance. The inability to solve these tasks seems to be due to inadequate exploration. Our results in VisualPinPad indicate that if exploration is addressed, then the performance gain of THICK Dreamer also holds for longer task horizons.

E.5.3 MBRL in MiniHack: Director

Director (Hafner et al., 2022) shows strong performance in the `KeyRoom-Fixed-S5` task. However, Director does not learn the other MiniHack tasks that we considered (Fig. 7.7).

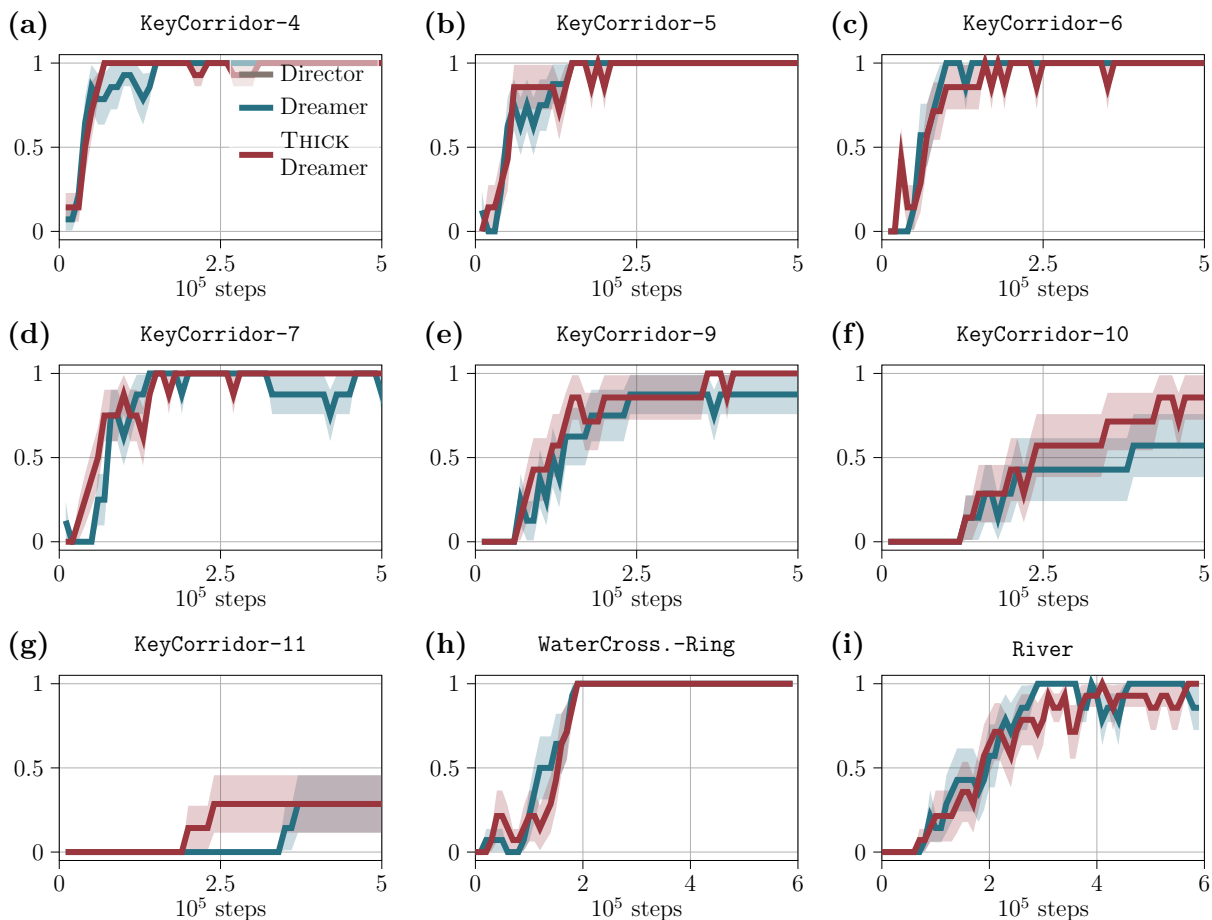


Figure E.8: MiniHack success for more tasks: We plot the mean evaluation success rate (7 seeds, \pm standard error).

Which crucial aspects are different across tasks and what causes Director to fail? We identify two key problems when applying Director in MiniHack, namely (1.) Director’s **reliance on diverse initial data** and (2.) problems with **specifying unobservable goal information**.

Diversity of initial data Director trains a goal encoder on its replay buffer from which it samples when training a goal-conditioned policy. We hypothesize that if early in training not enough diverse data is collected, this is reflected in the goal space. As a result, the manager (high-level) does not set meaningful goals for the worker (low-level) and learning is severely slowed down or stuck.

We analyze this aspect by training Director on variants of the **KeyRoom-Fixed-S5** problem. By default, the initial positions of agent, key, and goal within the second room

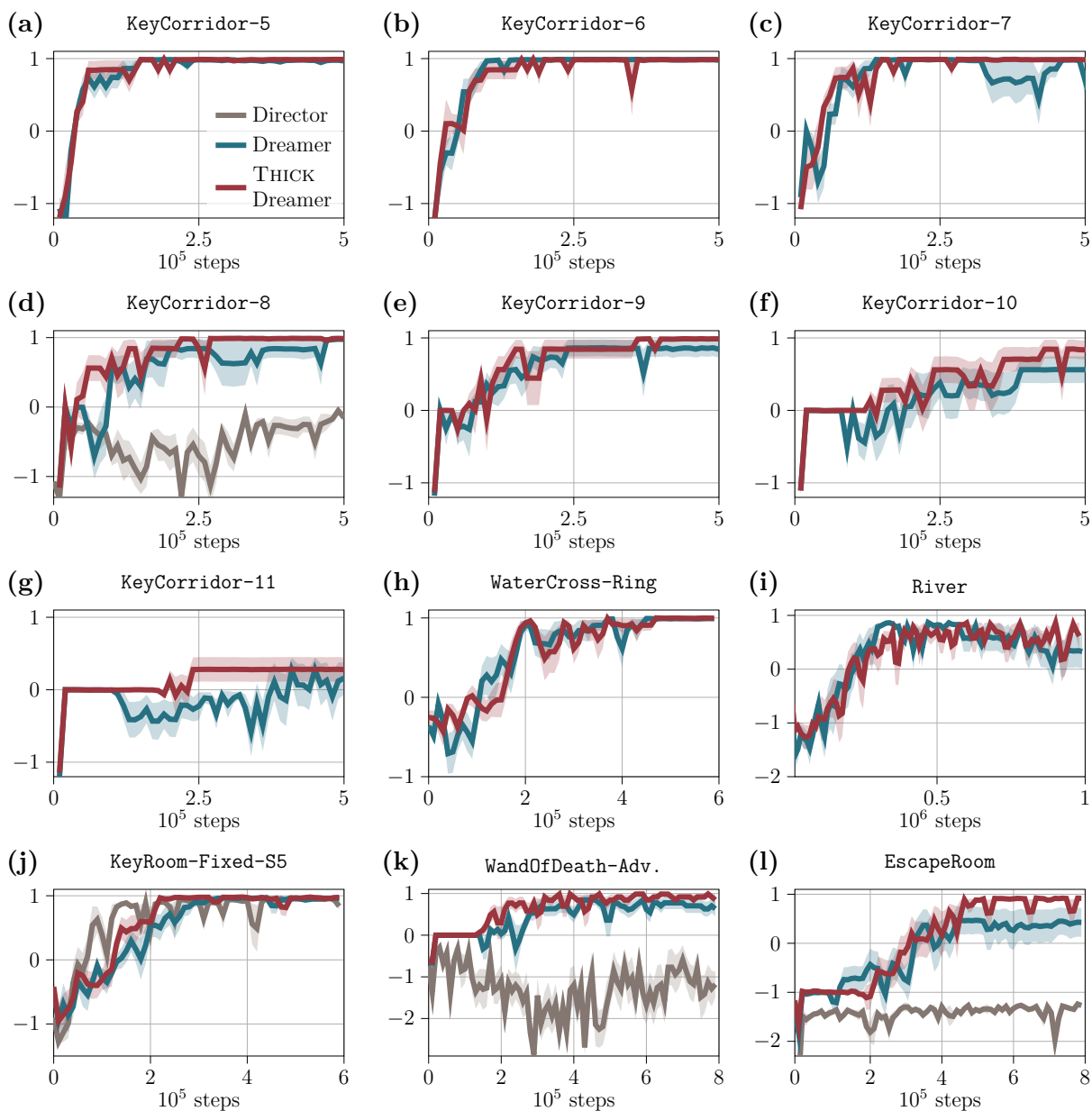


Figure E.9: MiniHack rewards: We plot the mean evaluation returns (7 seeds, \pm standard error).

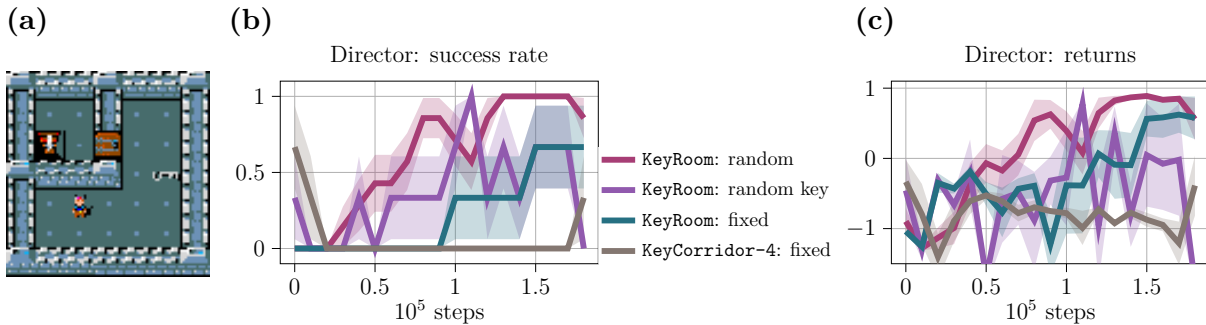


Figure E.10: Director in key-door-environments: We test Director in variants of the `KeyRoom-Fixed-S5` problem, with either fixed spawn positions of agent, key, and goal (i.e. `KeyRoom: fixed`, shown in **a**), random spawns points of the key (i.e. `KeyRoom: random key`), fully randomized spawn points (`KeyRoom: random`) or the similar `KeyCorridor-4` problem. We plot mean evaluation success rate (**b**) and mean evaluation returns (**c**) (7 seeds for `KeyRoom: random`, 3 seeds otherwise, \pm standard error).

are randomized in `KeyRoom-Fixed-S5`. We create additional variants of the task where either all entities spawn at fixed positions (positions shown in Fig. E.10a) or only the initial position of the key is randomized. Additionally, we train Director in the `KeyCorridor-4` task, which is very similar to `KeyRoom-Fixed-S5` with fixed spawn points but of much smaller size (8 grid corridor vs. two rooms with 16 and 4 grids). Thus, in `KeyCorridor-4` the observations show little diversity.

Figure E.10b plots evaluation success rates of Director in the `KeyRoom`-variants. Director needs more steps to solve the tasks when entities spawn at fixed positions. Director does not learn to solve `KeyCorridor-4` whereas with the same training it consistently learns to solve `KeyRoom-Fixed-S5`. Note that `KeyCorridor-4` is much smaller and has a shorter task horizon. A similar trend can be observed in the collected returns (Fig. E.10c).

Thus, we conclude that diversity in the initial observations drastically boosts Director’s performance. The ego-centric views of MiniHack often contain the same or similar observations, especially when traversing long corridors or empty rooms, e.g. in `KeyCorridor-8` or `WandOfDeath-Advanced`. This similarity in observations might impede Director’s learning in the MiniHack tasks we considered here.

Unobservable aspects of goals We hypothesize that a severe problem of Director could be specifying unobservable information in the goals. The RSSM encodes both observable and unobservable information within its deterministic latent state \mathbf{h}_t . If the unobservable information, e.g. item pick-ups in MiniHack, does not directly affect rewards or substantially influence image reconstruction, it might be encoded only locally in \mathbf{h}_t

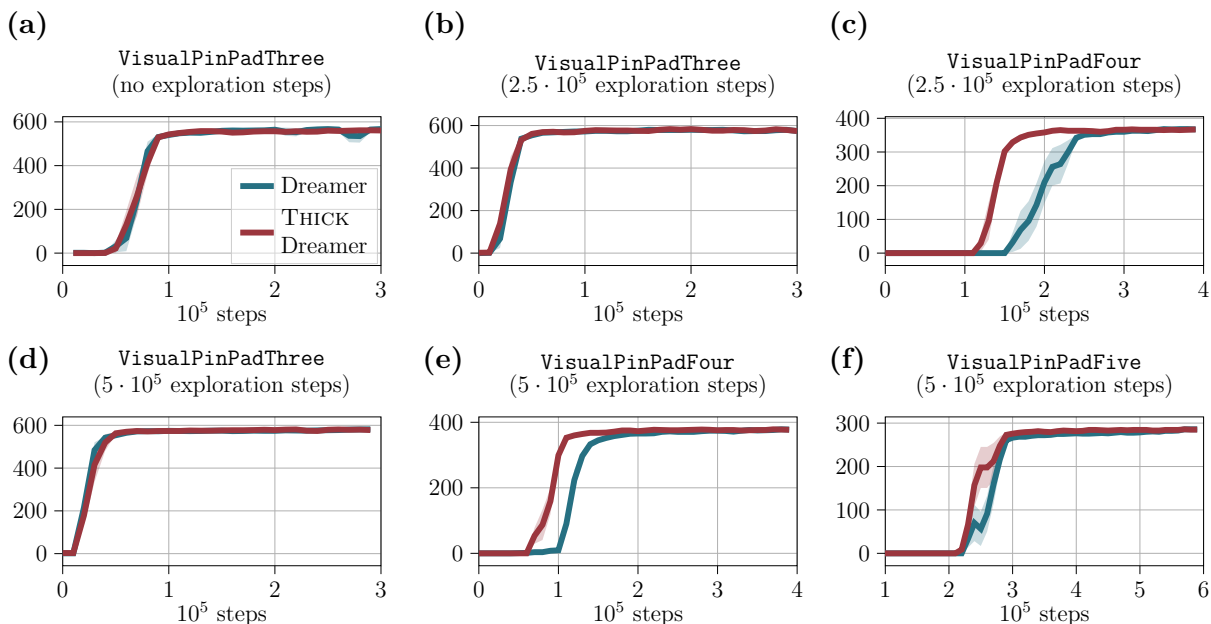


Figure E.11: Effects of exploration in VisualPinPad: We plot the mean evaluation returns (7 seeds for $5 \cdot 10^5$ exploration, 5 seeds otherwise; \pm standard error).

and fade over time. In Dreamer this is not a problem because the policy can amplify task-relevant information in \mathbf{h}_t . Director, however, compresses \mathbf{h}_t into a low-dimensional goal-encoding. Thereby, task-relevant latent information could get lost. Note that all novel tasks proposed in Hafner et al. (2022), in which Director shows strong performance, avoid long-horizon memory, e.g. by coloring walls in a maze (Egocentric Ant Maze) or by providing a visible history of past button presses (VisualPinPad).

In smaller MiniHack tasks, e.g. KeyRoom-Fixed-S5, memory can sometimes be circumvented by specifying goals via observable information. For example, if both the key and door are visible, a goal would be the same observation without the key visible (picked up) and an open door. This creates a curriculum in which the worker can first learn from such simpler situations and later learn to pick up a key and open the door automatically from the high-level goal of an open door. In larger task spaces, e.g. KeyCorridor-8, Director never encounters such simpler situations to begin with.

E.5.4 MBRL in VisualPinPad: Experiment Details

For the VisualPinPad suite we generated offline training data to avoid the challenge of discovering very sparse rewards. For data collection, we used Plan2Explore (Sekar et al.,

2020) with the default settings of the DreamerV2 (Hafner et al., 2020a) codebase. We trained two randomly initialized models of Plan2Explore for $S \in \{0, 250k, 500k, 1M\}$ environment steps in each task. For each setting, we determined the model that achieved the highest overall returns during training. We initialized the replay buffer of all new models with the S samples.

Originally, VisualPinPad has more levels of difficulty. However, in VisualPinPadSix Plan2Explore did not receive any rewards during the 1M steps of exploration. In addition to that, the results in Hafner et al. (2022) suggest that Dreamer is also not able to discover the very sparse rewards of VisualPinPadSix on its own. Thus, we omitted VisualPinPadSix and more complicated levels.

E.5.5 MBRL in VisualPinPad: Effect of Exploration

We analyze the effect of the exploration data by varying the number of data points with which we initialize the replay buffers. For this, we consider exploration data collected by $S \in \{0, 250k, 500k\}$ environment steps of Plan2Explore and compare THICK Dreamer to Dreamer. In PinPadThree, Dreamer and THICK Dreamer always achieve the same performance regardless of available exploration data (cf. Fig. E.11a, Fig. E.11b, Fig. E.11d). Without exploration data, neither THICK Dreamer nor Dreamer manage to obtain rewards in PinPadFour and PinPadFive within 600k steps. Similarly, both methods do not discover rewards when initialized with 250k steps of exploration in PinPadFive. Thus, for the more complicated problems both THICK Dreamer and Dreamer need sufficient exploration. Whenever there is enough exploration data to learn the more complicated tasks, THICK Dreamer manages to achieve high rewards faster than Dreamer (see Fig. E.11c, Fig. E.11e, Fig. E.11f).

For the larger problems, i.e. PinPadFour and PinPadFive, we quantify the effect of exploration data on sample efficiency by determining the number of environment steps needed to reach a certain level of reward. We take 95% of the highest mean reward across all our experiments as a threshold.^{E.5} Table E.3 shows the number of environment steps needed for THICK Dreamer and Dreamer reach this threshold for particular environments over exploration data. In sum, a medium amount of exploration data (500k) enables the fastest time to reach the threshold. THICK Dreamer reaches the reward threshold faster than Dreamer in all experiments. This advantage increases with level difficulty.

Table E.3: Sample efficiency in VisualPinPad. We list the number of environment steps needed for THICK Dreamer and Dreamer to reach a reward threshold (95% of max. reward) during evaluation for particular environments over amount of exploration data.

		PinPadFour		
exploration data	250k	500k	1M	
THICK Dreamer	200k	120k	140k	
Dreamer	280k	180k	200k	
difference	80k	60k	60k	

		PinPadFive		
exploration data	250k	500k	1M	
THICK Dreamer	/	260k	340k	
Dreamer	/	360k	590k	
difference	/	100k	250k	

E.5.6 MPC: Experiment Details and Extended Results

Experiment details To study zero-shot planning, we generated offline datasets of 1M environment steps for each task. For data collection, we used Plan2Explore in the same way as described in Suppl. E.5.4. After determining one dataset for every task, we trained the models purely on this data.

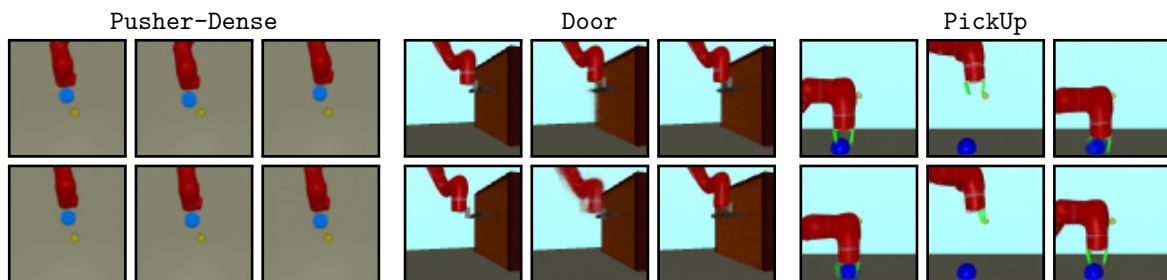


Figure E.12: Subgoals during hierarchical planning of THICK PlaNet reconstructed from z_1^{goal} in the first time step. The subgoals are typically pushing the puck (Pusher), moving to the door handle (Door), or grasping the ball (PickUp). For PickUp the system sometimes fails to find a reasonable subgoal (center).

^{E.5}This corresponds to a mean reward of 359 for PinPadFour and 274 for PinPadFive.

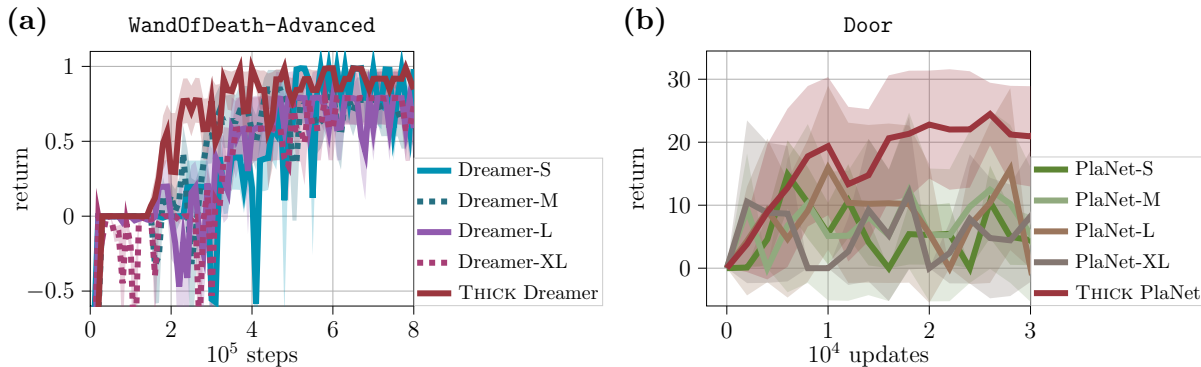


Figure E.13: Effect of scaling model size: We modify the number of hidden units per layer and dimensionality of the deterministic hidden state \mathbf{h}_t by a factor (S = 0.5, M = 1, L = 2, XL = 4) for Dreamer (a) or PlaNets (b) and compare to the unmodified THICK counterparts. Each graphic plots the mean returns over training (5 random seeds for ablations). Shaded areas depict the standard error in (a) or standard deviation in (b).

Subgoal reconstructions Besides boosting performance for long-horizon tasks, THICK PlaNets provides the additional advantage that the subgoals proposed by the high-level network can directly be reconstructed into images through the low-level output heads o_ϕ^c . The resulting goal images are easily interpretable by humans. Figure E.12 shows exemplary goals selected by the high-level planner in the first time step of an episode. Thus, the behavior of THICK PlaNets is much more explainable than simply performing MPC in a flat world model.

E.5.7 MBRL & MPC: Scaling Models vs. Hierarchy

An alternative approach towards improving world models, besides to adding hierarchies, is scaling up model size (Hafner et al., 2023; Deng et al., 2023). Could simply increasing the capacity of the world model improve performance of Dreamer or PlaNets similar to our approach of incorporating hierarchical abstraction?

We investigate this in the WandOfDeath-Adv. task for MBRL and in Door for zero-shot MPC. We increase the RSSM model capacity by scaling up the number of hidden units per layer (256 before) and the dimensionality of the deterministic hidden state \mathbf{h}_t (256 before) by different factors (factors: S = 0.5, M = 1, L = 2, XL = 4). Unlike the model scaling in Hafner et al. (2023), we did not increase the dimensionality of \mathbf{z}_t since both investigated environments are visually rather simple. Figure E.13 plots the returns of the different model sizes over environment steps. In both tasks, increasing the size of the model did

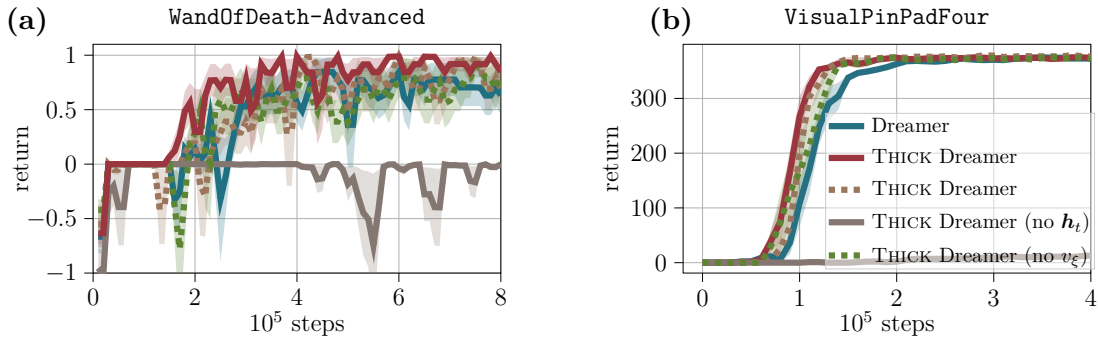


Figure E.14: C-RSSM ablations in THICK Dreamer. Each graphic plots the mean returns over training. We compare Dreamer, THICK Dreamer (7 seeds), and various ablations (5 seeds each): Dreamer with C-RSSM (C-RSSM Dreamer), Dreamer using only the coarse processing pathway of C-RSSM (C-RSSM Dreamer no h_t), and THICK Dreamer with only one critic (THICK Dreamer no v_ξ). Shaded areas depict standard error.

not improve Dreamer (Fig. E.13a) or PlaNet (Fig. E.13b). Thus, for the investigated setups, scaling up model size does not bring the same advantages as our THICK hierarchy.

E.5.8 Ablations and Hyperparameters

Ablations We ablate various components of the C-RSSM and THICK Dreamer within a MBRL setup. We evaluate the resulting systems using the two exemplary tasks of MiniHack-WandOfDeath-Adv. and VisualPinPadFour. Figure E.14 plots the returns of the ablated systems over environment steps. Using the C-RSSM in DreamerV2 results in roughly the same performance (WandOfDeath-Advances) or slightly better performance (VisualPinPadFour) than using the RSSM (i.e. Dreamer). However, removing the deterministic latent state h_t and the precise processing pathway from the C-RSSM (i.e. C-RSSM Dreamer without h) impedes the system from learning the tasks.^{E.6} Omitting v_ξ , and only using one critic v_χ for both the short- and long-horizon returns (Eq. 7.23), slightly degrades the performance of THICK Dreamer.

Hyperparameter β^{sparse} Next, we compare the effect of sparsity loss scale β^{sparse} on THICK Dreamer in VisualPinPadFour and on THICK PlaNet in Multiworld-Door.

Figure E.15a plots the mean returns of THICK Dreamer for different values for β^{sparse} in VisualPinPadFour. Figure E.15b shows the percentage of time steps with context

^{E.6}For this ablation we picked higher sparsity regularization β^{sparse} for both tasks ($\beta^{\text{sparse}} = 50$ for WandOfDeath-Advances, $\beta^{\text{sparse}} = 10$ for VisualPinPadFour), such that the number of time steps with open gate roughly matches that of the C-RSSM.

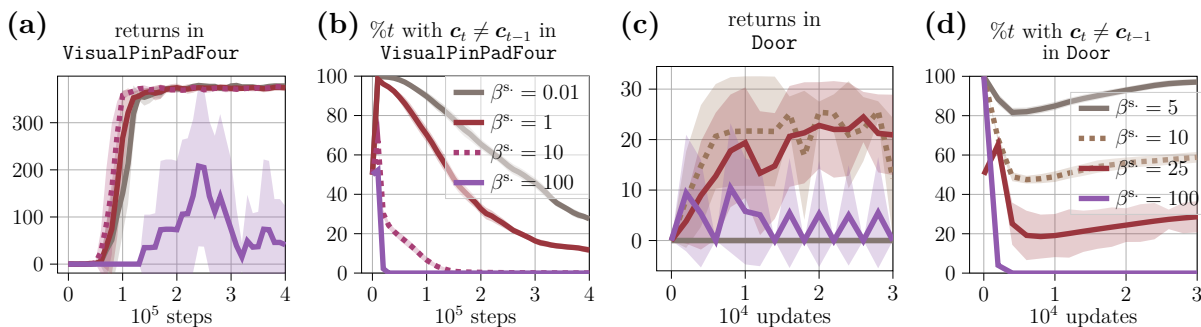


Figure E.15: Effect of sparsity on THICK. We plot the mean return for THICK Dreamer in `VisualPinPadFour` (a) and the mean zero-shot planning returns of THICK PlaNet in `Door` (c) for different values of the hyperparameter β^{sparse} (5 random seeds). Additionally, we plot the percentage of time steps with context changes over training time for both tasks (b, d). We test different ranges of β^{sparse} for the different tasks. Shaded areas depict the standard deviation.

changes over training. For THICK Dreamer, regularizing context changes too little is not as detrimental as overly regularizing context changes. If the contexts are weakly regularized, i.e. small β^{sparse} , then the context changes in most time steps. As a result, the high-level learns an identity mapping, and during a temporal abstract prediction, the network simply predicts the next state at time $t + 1$ (see Alg. 2). Stronger regularization boosts sample efficiency of learning long-horizon behavior. This is even true if, at some point after the behavior is sufficiently learned, the context is no longer adapted (e.g. $\beta^{\text{sparse}} = 10$). However, overly strong regularization, which prohibits context changes early during training, impedes the learning of the task (e.g. $\beta^{\text{sparse}} = 100$). In this case, the high-level predictions are essentially average state predictions, which only contribute noise for learning the critic. THICK Dreamer is very robust to the choice of β^{sparse} in `VisualPinPadFour`.

Figure E.15c plots zero-shot planning performance of THICK PlaNet for different values for β^{sparse} , with the percentage of context changes shown in Fig. E.15d. For THICK PlaNet both too strong as well as too weak regularization degrade performance. However, strongly regularizing the network towards sparse context changes is slightly less detrimental for THICK PlaNet than a weak sparsity regularization (cf. $\beta^{\text{sparse}} = 100$ and $\beta^{\text{sparse}} = 5$). For weak sparsity regularization the context changes in every time step, which prevents the high level from finding a useful subgoal sequence during planning. As a result, the low-level might be guided into the wrong direction by the proposed subgoals.

Hyperparameter ψ THICK Dreamer introduces a new hyperparameter ψ which

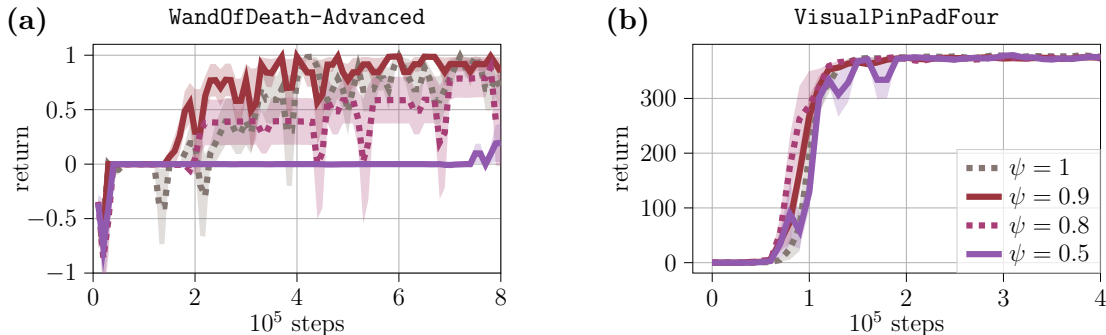


Figure E.16: Effect of hyperparameter ψ . Each graphic plots the mean returns for THICK Dreamer over training steps for different values of the hyperparameter ψ (5 random seeds). Shaded areas depict standard error.

balances the influence of the short-horizon value estimates V^λ and long-horizon value estimates V^{long} on the overall value V (Eq. 7.23). Figure E.16 shows how ψ affects task performance. Only considering short-horizon value estimates, i.e. $\psi = 1$, results in less sample efficient learning than taking small amounts of long-horizon value estimates into consideration, i.e. $\psi = 0.9$ for `WandOfDeath-Advanced` and $0.8 \leq \psi \leq 0.9$ for `VisualPinPadFour`. However, relying too strongly on long-horizon estimates, i.e. $\psi = 0.5$, impedes policy learning. This effect is less pronounced for very long-horizon tasks such as `VisualPinPadFour`. We set $\psi = 0.9$ in all experiments.

Hyperparameter κ THICK PlaNet introduces the hyperparameter κ , which scales the influence of the subgoal proximity on the reward estimate of the low-level planner (Eq. 7.26). We analyze the effect of κ on the performance of THICK PlaNet in `Multiworld-Door`, shown in Fig. E.17. Incentivizing subgoal proximity too strongly, i.e. $\kappa = 1$, can result in the agent getting stuck at a subgoal. This reduces overall task performance. Ignoring the subgoal, i.e. $\kappa = 0$, also decreases performance for long-horizon tasks such as `Door`. In `Door`, THICK PlaNet works well across a wide range of κ .

Replanning strategy THICK PlaNet proposes new goals on the high-level upon context transitions. We do this mainly to save computational cost from running MCTS at the high-level at every time step. Figure E.18 compares the effect of high-level planning in every time step to replanning upon context transitions in `Door`. The returns seem mostly the same. Thus, replanning at every time step is as effective as setting new subgoals only upon context transitions and can be applied if computational efficiency is not a concern.

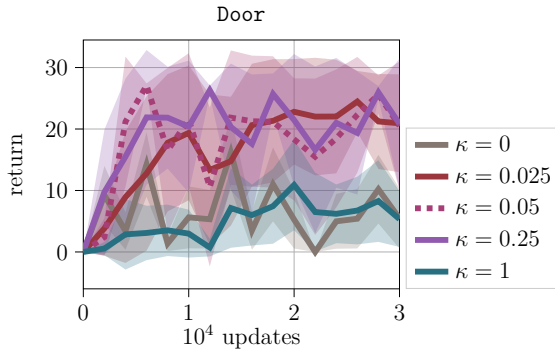


Figure E.17: Effect of hyperparameter κ . We plot mean returns of THICK PlaNet for zero-shot MPC for different values of the hyperparameter κ balancing external and subgoal-reaching rewards (5 seeds, \pm standard deviation).

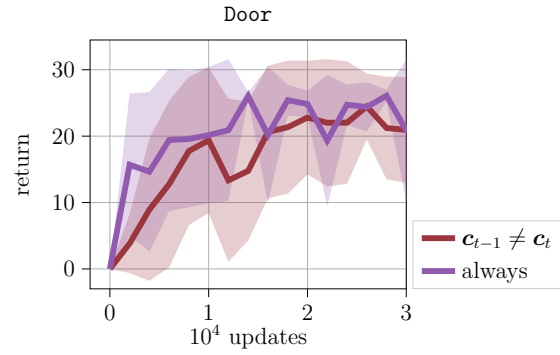


Figure E.18: Effect of replanning. We plot mean returns of THICK PlaNet for zero-shot MPC when only replanning upon context transition or when planning on every step (5 seeds, \pm standard deviation).

Bibliography

- Adam, M. & Elsner, B. (2018). Action effects foster 11-month-olds' prediction of action goals for a non-human agent. *Infant Behavior and Development*, 53, 49–55.
- Adam, M. & Elsner, B. (2020). The impact of salient action effects on 6-, 7-, and 11-month-olds' goal-predictive gaze shifts for a human grasping action. *PLOS ONE*, 15(10), 1–18.
- Adam, M., Gumbsch, C., Butz, M. V., & Elsner, B. (2021). The impact of action effects on infants' predictive gaze shifts for a non-human grasping action at 7, 11, and 18 months. *Frontiers in Psychology*, 12.
- Adam, M., Reitenbach, I., & Elsner, B. (2017). Agency cues and 11-month-olds' and adults' anticipation of action goals. *Cognitive Development*, 43, 37–48.
- Adam, M., Reitenbach, I., Papenmeier, F., Gredebäck, G., Elsner, C., & Elsner, B. (2016). Goal saliency boosts infants' action prediction for human manual actions, but not for mechanical claws. *Infant Behavior and Development*, 44, 29–37.
- Akakzia, A., Colas, C., Oudeyer, P.-Y., Chetouani, M., & Sigaud, O. (2021). Grounding language to autonomously-acquired skills via goal generation. In *The Ninth International Conference on Learning Representation, ICLR'21*.
- Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. (2017). Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30.
- Ba, J., Mnih, V., & Kavukcuoglu, K. (2015). Multiple object recognition with visual attention. In *The Third International Conference on Learning Representations, ICLR'15*.
- Bacon, P.-L., Harb, J., & Precup, D. (2017). The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31 (pp. 1726–1734).

- Badre, D. & D’esposito, M. (2009). Is the rostro-caudal axis of the frontal lobe hierarchical? *Nature Reviews Neuroscience*, 10(9), 659–669.
- Bagatella, M., Olšák, M., Rolínek, M., & Martius, G. (2021). Planning from pixels in environments with combinatorially hard search spaces. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 24707–24718).
- Bahdanau, D., Cho, K. H., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *The Third International Conference on Learning Representations*, ICLR’15.
- Balaguer, J., Spiers, H., Hassabis, D., & Summerfield, C. (2016). Neural mechanisms of hierarchical planning in a virtual subway network. *Neuron*, 90(4), 893–903.
- Baldwin, D. A. & Kosie, J. E. (2021). How does the mind render streaming experience as events? *Topics in Cognitive Science*, 13(1), 79–105.
- Barlow, H. B. et al. (1961). Possible principles underlying the transformation of sensory messages. *Sensory Communication*, 1(01), 217–233.
- Barsalou, L. W. (2008). Grounded cognition. *Annual Review of Psychology*, 59(1), 617–645.
- Basgol, H., Ayhan, I., & Ugur, E. (2024). Predictive event segmentation and representation with neural networks: A self-supervised model assessed by psychological experiments. *Cognitive Systems Research*, 83, 101167.
- Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint*. <https://arxiv.org/abs/1806.01261>.
- Bellman, R. (1957). A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5), 679–684.
- Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015a). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28.
- Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., & Lin, Z. (2015b). Towards biologically plausible deep learning. *arXiv preprint*. <https://arxiv.org/abs/1502.04156>.

- Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint*. <https://arxiv.org/abs/1308.3432>.
- Bertsekas, D. (2012). *Dynamic Programming and Optimal Control: Volume I*, volume 4. Athena scientific.
- Beyer, H.-G. & Schwefel, H.-P. (2002). Evolution strategies—a comprehensive introduction. *Natural computing*, 1, 3–52.
- Bezdek, M., Nguyen, T., Gershman, S. J., Bobick, A. F., Braver, T. S., & Zacks, J. M. (2022). Uncertainty-driven updating enables human-like segmentation and categorization of naturalistic activity. *PsyArXiv preprint*. <https://doi.org/10.31234/osf.io/pt6hx>.
- Bhui, R., Lai, L., & Gershman, S. J. (2021). Resource-rational decision making. *Current Opinion in Behavioral Sciences*, 41, 15–21.
- Binz, M. & Schulz, E. (2023). Using cognitive psychology to understand gpt-3. *Proceedings of the National Academy of Sciences*, 120(6), e2218523120.
- Biro, S. (2013). The role of the efficiency of novel actions in infants’ goal anticipation. *Journal of Experimental Child Psychology*, 116(2), 415–427.
- Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Berlin, Heidelberg: Springer-Verlag.
- Botvinick, M. (2007). Multilevel structure in behaviour and in the brain: a model of fuster’s hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 362(1485), 1615–1626.
- Botvinick, M. (2008). Hierarchical models of behavior and prefrontal function. *Trends in Cognitive Sciences*, 12(5), 201–208.
- Botvinick, M. & Weinstein, A. (2014). Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1655), 20130480.
- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI gym. *arXiv preprint*. <https://arxiv.org/abs/arXiv:1606.01540>.

- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33 (pp. 1877–1901).
- Brunec, I. K., Bellana, B., Ozubko, J. D., Man, V., Robin, J., Liu, Z.-X., Grady, C., Rosenbaum, R. S., Winocur, G., Barense, M. D., & Moscovitch, M. (2018). Multiple scales of representation along the hippocampal anteroposterior axis in humans. *Current Biology*, 28(13), 2129–2135.e6.
- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv preprint*. <https://arxiv.org/abs/2303.12712>.
- Buschhoff, L. M. S., Akata, E., Bethge, M., & Schulz, E. (2023). Have we built machines that think like people? *arXiv preprint*. <https://arxiv.org/abs/2311.16093>.
- Butz, M. V. (2016). Towards a unified sub-symbolic computational theory of cognition. *Frontiers in Psychology*, 7(925).
- Butz, M. V. (2017). Which structures are out there. In *Philosophy and Predictive Processing* chapter 8. MIND Group.
- Butz, M. V. (2022). Resourceful event-predictive inference: The nature of cognitive effort. *Frontiers in Psychology*, 13.
- Butz, M. V., Achimova, A., Bilkey, D., & Knott, A. (2021). Event-predictive cognition: A root for conceptual human thought. *Topics in Cognitive Science*, 13(1), 10–24.
- Butz, M. V., Bilkey, D., Humaidan, D., Knott, A., & Otte, S. (2019). Learning, planning, and control in a monolithic neural event inference architecture. *Neural Networks*, 117, 135–144.
- Butz, M. V. & Kutter, E. F. (2017). *How the Mind Comes into Being*. Oxford University Press.
- Campos, V., Jou, B., Giró-i Nieto, X., Torres, J., & Chang, S.-F. (2018). Skip rnn: Learning to skip state updates in recurrent neural networks. In *The Sixth International Conference on Learning Representations, ICLR'18*.
- Cannon, E. N. & Woodward, A. L. (2012). Infants generate goal-based action predictions. *Developmental Science*, 15(2), 292–298.

- Cannon, E. N., Woodward, A. L., Gredebäck, G., von Hofsten, C., & Turek, C. (2012). Action production influences 12-month-old infants' attention to others' actions. *Developmental Science*, 15(1), 35–42.
- Chen, C., Wu, Y.-F., Yoon, J., & Ahn, S. (2022). Transdreamer: Reinforcement learning with transformer world models. *arXiv preprint*. <https://arxiv.org/abs/2202.09481>.
- Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. In *2016 Conference on Empirical Methods in Natural Language Processing* (pp. 551–561).
- Chevalier-Boisvert, M., Bahdanau, D., Lahlou, S., Willems, L., Saharia, C., Nguyen, T. H., & Bengio, Y. (2018a). BabyAI: A platform to study the sample efficiency of grounded language learning. In *The Sixth International Conference on Learning Representations*, ICLR'18.
- Chevalier-Boisvert, M., Willems, L., & Pal, S. (2018b). Minimalistic gridworld environment for openai gym. <https://github.com/maximecb/gym-minigrid>. Accessed: 03.08.2021.
- Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing*, volume 31.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint*. <https://arxiv.org/abs/1412.3555>.
- Clark, A. (2015). *Surfing Uncertainty: Prediction, Action, and the Embodied Mind*. Oxford University Press.
- Cohen, A., Ivry, R. I., & Keele, S. W. (1990). Attention and structure in sequence learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 16(1), 17.
- Colombo, M. & Wright, C. (2021). First principles in the life sciences: The free-energy principle, organicism, and mechanism. *Synthese*, 198, 3463–3488.
- Cooper, R. & Shallice, T. (2000). Contention scheduling and the control of routine activities. *Cognitive Neuropsychology*, 17(4), 297–338.

- Cooper, R. P. (2021). Action production and event perception as routine sequential behaviors. *Topics in Cognitive Science*, 13(1), 63–78.
- Copete, J. L., Nagai, Y., & Asada, M. (2016). Motor development facilitates the prediction of others’ actions through sensorimotor predictive learning. In *2016 Joint IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)* (pp. 223–229).
- Da Costa, L., Lanillos, P., Sajid, N., Friston, K., & Khan, S. (2022). How active inference could help revolutionise robotics. *Entropy*, 24(3).
- Da Costa, L., Parr, T., Sajid, N., Veselic, S., Neacsu, V., & Friston, K. (2020). Active inference on discrete state-spaces: A synthesis. *Journal of Mathematical Psychology*, 99, 102447.
- Dayan, P. (2009). Goal-directed control and its antipodes. *Neural Networks*, 22(3), 213–219.
- Degrave, J., Felici, F., Buchli, J., Neunert, M., Tracey, B., Carpanese, F., Ewalds, T., Hafner, R., Abdolmaleki, A., de Las Casas, D., et al. (2022). Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897), 414–419.
- Dekker, R. B., Otto, F., & Summerfield, C. (2022). Curriculum learning for human compositional generalization. *Proceedings of the National Academy of Sciences*, 119(41), e2205582119.
- Deng, F., Park, J., & Ahn, S. (2023). Facing off world model backbones: Rnns, transformers, and s4. In *Advances in Neural Information Processing Systems*, volume 37.
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint*. <https://arxiv.org/abs/1810.04805>.
- Dolan, R. J. & Dayan, P. (2013). Goals and habits in the brain. *Neuron*, 80(2), 312–325.
- Doncieux, S., Filliat, D., Díaz-Rodríguez, N., Hospedales, T., Duro, R., Coninx, A., Roijers, D. M., Girard, B., Perrin, N., & Sigaud, O. (2018). Open-ended learning: A conceptual framework based on representational redescription. *Frontiers in Neurobotics*, 12.
- DuBrow, S. & Davachi, L. (2013). The influence of context boundaries on memory for the sequential order of events. *Journal of Experimental Psychology: General*, 142(4), 1277.

- Duncker, K. & Lees, L. S. (1945). On problem-solving. *Psychological Monographs*, 58(5).
- Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2), 179–211.
- Elsner, B. & Adam, M. (2021). Infants’ goal prediction for simple action events: The role of experience and agency cues. *Topics in Cognitive Science*, 13(1), 45–62.
- Elsner, B. & Hommel, B. (2001). Effect anticipation and action control. *Journal of Experimental Psychology: Human Perception and Performance*, 27(1), 229–240.
- Eppe, M., Gumbsch, C., Kerzel, M., Nguyen, P. D., Butz, M. V., & Wermter, S. (2022). Intelligent problem-solving as integrated hierarchical reinforcement learning. *Nature Machine Intelligence*, 4, 11–20.
- Eysenbach, B., Gupta, A., Ibarz, J., & Levine, S. (2018). Diversity is all you need: Learning skills without a reward function. In *The Sixth International Conference on Learning Representations*, ICLR’18.
- Falck-Ytter, T., Gredebäck, G., & von Hofsten, C. (2006). Infants predict other people’s action goals. *Nature Neuroscience*, 9(7), 878–879.
- Fantz, R. L. (1958). Pattern vision in young infants. *The Psychological Record*, 8, 43–47.
- Fitch, W. T. & Martins, M. D. (2014). Hierarchical processing in music, language, and action: Lashley revisited. *Annals of the New York Academy of Sciences*, 1316(1), 87–104.
- Flanagan, J. R. & Johansson, R. S. (2003). Action plans used in action observation. *Nature*, 424(6950), 769–771.
- Fodor, J. A. (2001). Language, thought and compositionality. *Mind & Language*, 16(1), 1–15.
- Fountas, Z., Sajid, N., Mediano, P., & Friston, K. (2020). Deep active inference agents using monte-carlo methods. In *Advances in Neural Information Processing Systems*, volume 33 (pp. 11662–11675).
- Frankland, S. M. & Greene, J. D. (2020). Concepts and compositionality: In search of the brain’s language of thought. *Annual Review of Psychology*, 71, 273–303.
- Franklin, N. T. & Frank, M. J. (2020). Generalizing to generalize: Humans flexibly switch between compositional and conjunctive structures during reinforcement learning. *PLOS Computational Biology*, 16(4), 1–33.

- Franklin, N. T., Norman, K. A., Ranganath, C., Zacks, J. M., & Gershman, S. J. (2020). Structured event memory: A neuro-symbolic model of event cognition. *Psychological Review*, 127(3), 327–361.
- Friston, K. (2009). The free-energy principle: a rough guide to the brain? *Trends in Cognitive Sciences*, 13(7), 293–301.
- Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2), 127–138.
- Friston, K., FitzGerald, T., Rigoli, F., Schwartenbeck, P., O’Doherty, J., & Pezzulo, G. (2016). Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68, 862–879.
- Friston, K., Kilner, J., & Harrison, L. (2006). A free energy principle for the brain. *Journal of Physiology*, 100(1-3), 70–87.
- Friston, K., Mattout, J., & Kilner, J. (2011). Action understanding and active inference. *Biological Cybernetics*, 104, 137–160.
- Friston, K., Moran, R. J., Nagai, Y., Taniguchi, T., Gomi, H., & Tenenbaum, J. (2021). World model learning and inference. *Neural Networks*, 144, 573–590.
- Friston, K., Rigoli, F., Ognibene, D., Mathys, C., FitzGerald, T., & Pezzulo, G. (2015). Active inference and epistemic value. *Cognitive Neuroscience*, 6, 187–214.
- Friston, K. J., Daunizeau, J., & Kiebel, S. J. (2009). Reinforcement learning or active inference? *PLOS ONE*, 4(7), 1–13.
- Friston, K. J., Daunizeau, J., Kilner, J., & Kiebel, S. J. (2010). Action and behavior: a free-energy formulation. *Biological Cybernetics*, 102, 227–260.
- Friston, K. J., Rosch, R., Parr, T., Price, C., & Bowman, H. (2018). Deep temporal models and active inference. *Neuroscience & Biobehavioral Reviews*, 90, 486–501.
- Ganglmayer, K., Attig, M., Daum, M. M., & Paulus, M. (2019). Infants’ perception of goal-directed actions: A multi-lab replication reveals that infants anticipate paths and not goals. *Infant Behavior and Development*, 57, 101340.
- Gärdenfors, P. (2014). *The Geometry of Meaning: Semantics Based on Conceptual Spaces*. Cambridge, MA: MIT Press.

- Gentner, D. (2006). Analogical reasoning, psychology of. *Encyclopedia of Cognitive Science*, (pp. 106–112).
- Gentner, D., Brem, S., Ferguson, R. W., Markman, A. B., B, B., Levidow, Wolff, P., & Forbus, K. D. (1997). Analogical reasoning and conceptual change: A case study of johannes kepler. *Journal of the Learning Sciences*, 6(1), 3–40.
- Gentner, D. & Maravilla, F. (2017). Analogical reasoning. *International Handbook of Thinking and Reasoning*, (pp. 186–203).
- Gershman, S. J., Horvitz, E. J., & Tenenbaum, J. B. (2015). Computational rationality: A converging paradigm for intelligence in brains, minds, and machines. *Science*, 349(6245), 273–278.
- Gick, M. L. & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, 12(3), 306–355.
- Glad, T. & Ljung, L. (2018). *Control theory*. CRC press.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. <http://www.deeplearningbook.org>.
- Goyal, A. & Bengio, Y. (2022). Inductive biases for deep learning of higher-level cognition. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 478(2266), 20210068.
- Goyal, A., Didolkar, A. R., Ke, N. R., Blundell, C., Beaudoin, P., Heess, N., Mozer, M. C., & Bengio, Y. (2021a). Neural production systems. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 25673–25687).
- Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., & Schölkopf, B. (2021b). Recurrent independent mechanisms. In *The Nineth International Conference on Learning Representations, ICLR'21*.
- Graves, A., Wayne, G., & Danihelka, I. (2014). Neural turing machines. *arXiv preprint*. <https://arxiv.org/abs/1410.5401>.
- Gredebäck, G. & Falck-Ytter, T. (2015). Eye movements during action observation. *Perspectives on Psychological Science*, 10(5), 591–598.
- Gredebäck, G., Johnson, S., & von Hofsten, C. (2010). Eye tracking in infancy research. *Developmental Neuropsychology*, 35(1), 1–19.

- Gredebäck, G. & Melinder, A. (2010). Infants’ understanding of everyday social interactions: A dual process account. *Cognition*, 114(2), 197–206.
- Greff, K., Van Steenkiste, S., & Schmidhuber, J. (2020). On the binding problem in artificial neural networks. *arXiv preprint*. <https://arxiv.org/abs/2012.05208>.
- Gruber, R., Schiestl, M., Boeckle, M., Frohnwieser, A., Miller, R., Gray, R. D., Clayton, N. S., & Taylor, A. H. (2019). New caledonian crows use mental representations to solve metatool problems. *Current Biology*, 29(4), 686–692.
- Gu, A., Dao, T., Ermon, S., Rudra, A., & Ré, C. (2020). Hippo: Recurrent memory with optimal polynomial projections. In *Advances in Neural Information Processing Systems*, volume 33 (pp. 1474–1487).
- Gu, A., Goel, K., & Re, C. (2021). Efficiently modeling long sequences with structured state spaces. In *The Ninth International Conference on Learning Representations, ICLR’21*.
- Gumbsch, C., Adam, M., Elsner, B., & Butz, M. V. (2021a). Emergent goal-anticipatory gaze in infants via event-predictive learning and inference. *Cognitive Science*, 45(8), e13016.
- Gumbsch, C., Adam, M., Elsner, B., Martius, G., & Butz, M. V. (2022). Developing hierarchical anticipations via neural network-based event segmentation. In *2022 IEEE International Conference on Development and Learning (ICDL)* (pp. 1–8): IEEE.
- Gumbsch, C., Butz, M. V., & Martius, G. (2019). Autonomous identification and goal-directed invocation of event-predictive behavioral primitives. *IEEE Transactions on Cognitive and Developmental Systems*, 13(2), 298–311.
- Gumbsch, C., Butz, M. V., & Martius, G. (2021b). Sparsely changing latent states for prediction and planning in partially observable domains. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 17518–17531).
- Gumbsch, C., Sajid, N., Martius, G., & Butz, M. V. (2024, to appear). Learning hierarchical world models with adaptive temporal abstractions from discrete latent dynamics. In *International Conference on Learning Representations*.
- Gürtler, N., Büchler, D., & Martius, G. (2021). Hierarchical reinforcement learning with timed subgoals. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 21732–21743).

- Ha, D., Dai, A. M., & Le, Q. V. (2016). Hypernetworks. In *The Fourth International Conference on Learning Representations*, ICLR'16.
- Ha, D. & Schmidhuber, J. (2018). World models. *arXiv preprint*. <https://arxiv.org/abs/1803.10122>.
- Hafner, D., Lee, K.-H., Fischer, I., & Abbeel, P. (2022). Deep hierarchical planning from pixels. In *Advances in Neural Information Processing*, volume 35 (pp. 26091–26104).
- Hafner, D., Lillicrap, T., Ba, J., & Norouzi, M. (2019a). Dream to control: Learning behaviors by latent imagination. In *The Seventh International Conference on Learning Representations*, ICLR'19.
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., & Davidson, J. (2019b). Learning latent dynamics for planning from pixels. In *Proceedings of the 37th International Conference on Machine Learning* (pp. 2555–2565).
- Hafner, D., Lillicrap, T. P., Norouzi, M., & Ba, J. (2020a). Mastering atari with discrete world models. In *The Eighth International Conference on Learning Representations*, ICLR'20.
- Hafner, D., Ortega, P. A., Ba, J., Parr, T., Friston, K., & Heess, N. (2020b). Action and perception as divergence minimization. *arXiv preprint*. <https://arxiv.org/abs/2009.01791>.
- Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint*. <https://arxiv.org/abs/2301.04104>.
- Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1), 335–346.
- Hartvigsen, T., Sen, C., Kong, X., & Rundensteiner, E. (2020). Learning to selectively update state neurons in recurrent networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (pp. 485–494).
- Hausknecht, M. J. & Stone, P. (2015). Deep recurrent Q-learning for partially observable MDPs. *arXiv preprint*. <https://arxiv.org/abs/1507.06527>.
- Hauskrecht, M. (2000). Value-function approximations for partially observable markov decision processes. *Journal of Artificial Intelligence Research*, 13, 33–94.

- Haynes, J.-D., Wisniewski, D., Gorgen, K., Momennejad, I., & Reverberi, C. (2015). FMRI decoding of intentions: Compositionality, hierarchy and prospective memory. In *The 3rd International Winter Conference on Brain-Computer Interface* (pp. 1–3).
- Heald, J. B., Lengyel, M., & Wolpert, D. M. (2021). Contextual inference underlies the learning of sensorimotor repertoires. *Nature*, 600, 489–493.
- Heald, J. B., Lengyel, M., & Wolpert, D. M. (2023). Contextual inference in learning and memory. *Trends in Cognitive Sciences*, 27(1), 43–64.
- Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02), 107–116.
- Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
- Hohwy, J. (2013). *The Predictive Mind*. Oxford University Press.
- Hommel, B. (2009). Action control according to TEC (theory of event coding). *Psychological Research PRPF*, 73(4), 512–526.
- Hommel, B. (2019). Theory of event coding (TEC) v2.0: Representing and controlling perception and action. *Attention, Perception, & Psychophysics*, 81, 2139–2154.
- Hommel, B., Musseler, J., Aschersleben, G., & Prinz, W. (2001). The theory of event coding (TEC): A framework for perception and action planning. *Behavioral and Brain Sciences*, 24(5), 849–878.
- Howard, R. A. (1964). System analysis of semi-markov processes. *IEEE Transactions on Military Electronics*, 8(2), 114–124.
- Humaidan, D., Otte, S., & Butz, M. V. (2020). Fostering event compression using gated surprise. In *International Conference on Artificial Neural Networks, ICANN’20* (pp. 155–167).
- Humaidan, D., Otte, S., Gumbsch, C., Wu, C. M., & Butz, M. V. (2021). Latent event-predictive encodings through counterfactual regularization. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 43 (pp. 1726–1732).
- Igl, M., Zintgraf, L., Le, T. A., Wood, F., & Whiteson, S. (2018). Deep variational reinforcement learning for POMDPs. In *Proceedings of the 36th International Conference on Machine Learning* (pp. 2117–2126).

- Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., & Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, 25(2), 328–373.
- Ito, T., Klinger, T., Schultz, D., Murray, J., Cole, M., & Rigotti, M. (2022). Compositional generalization through abstract representations in human and artificial neural networks. In *Advances in Neural Information Processing Systems*, volume 35 (pp. 32225–32239).
- Jain, A. K., Sujit, S., Joshi, S., Michalski, V., Hafner, D., & Ebrahimi Kahou, S. (2022). Learning robust dynamics through variational sparse gating. In *Advances in Neural Information Processing Systems*, volume 35 (pp. 1612–1626).
- Jang, E., Gu, S., & Poole, B. (2017). Categorical reparameterization with gumbel-softmax. In *The Fifth International Conference on Learning Representations*, ICLR’17.
- Jayaraman, D., Ebert, F., Efros, A., & Levine, S. (2019). Time-agnostic prediction: Predicting predictable video frames. In *The Seventh International Conference on Learning Representations*, ICLR’19.
- Johansson, R. S., Westling, G., Bäckström, A., & Flanagan, J. R. (2001). Eye–hand coordination in object manipulation. *Journal of Neuroscience*, 21(17), 6917–6932.
- Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1), 99–134.
- Kahneman, D. (1973). *Attention and Effort*. Citeseer.
- Kanakogi, Y. & Itakura, S. (2011). Developmental correspondence between action prediction and motor ability in early infancy. *Nature Communications*, 2, 341.
- Keele, S. W., Cohen, A., & Ivry, R. (1990). Motor programs: Concepts and issues. *Attention and Performance*, 13, 77–110.
- Khetarpal, K., Riemer, M., Rish, I., & Precup, D. (2022). Towards continual reinforcement learning: A review and perspectives. *Journal of Artificial Intelligence Research*, 75, 1401–1476.
- Kiebel, S. J., Daunizeau, J., & Friston, K. J. (2008). A hierarchy of time-scales and the brain. *PLOS Computational Biology*, 4(11), 1–12.
- Kim, T., Ahn, S., & Bengio, Y. (2019). Variational temporal abstraction. In *Advances in Neural Information Processing Systems*, volume 32.

- Kingma, D. P. & Ba, J. (2015). Adam: A method for stochastic optimization. In *The Third International Conference for Learning Representations*, ICLR'15.
- Kingma, D. P. & Welling, M. (2014). Auto-encoding variational bayes. In *The Fourth International Conference on Learning Representations*, ICLR'14.
- Kipf, T., Elsayed, G. F., Mahendran, A., Stone, A., Sabour, S., Heigold, G., Jonchkowski, R., Dosovitskiy, A., & Greff, K. (2021). Conditional object-centric learning from video. In *The Nineth International Conference on Learning Representations*, ICLR'21.
- Kirk, R., Zhang, A., Grefenstette, E., & Rocktäschel, T. (2023). A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76, 201–264.
- Knott, A. (2012). *Sensorimotor Cognition and Natural Language Syntax*. MIT press.
- Konda, V. & Tsitsiklis, J. (1999). Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, volume 12.
- Koutnik, J., Greff, K., Gomez, F., & Schmidhuber, J. (2014). A Clockwork RNN. In *Proceedings of the 31st International Conference on Machine Learning* (pp. 1863–1871).
- Krogh-Jespersen, S. & Woodward, A. L. (2014). Making smart social judgments takes time: Infants' recruitment of goal information when generating action predictions. *PLOS ONE*, 9(5), e98085.
- Krueger, D. & Memisevic, R. (2015). Regularizing RNNs by stabilizing activations. *arXiv preprint arXiv:1511.08400*. <https://arxiv.org/abs/1511.08400>.
- Kuperberg, G. R. (2021). Tea with milk? a hierarchical generative framework of sequential event comprehension. *Topics in Cognitive Science*, 13(1), 256–298.
- Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., & Rocktäschel, T. (2020). The nethack learning environment. In *Advances in Neural Information Processing Systems*, volume 33 (pp. 7671–7684).
- Lake, B. M., Linzen, T., & Baroni, M. (2019). Human few-shot learning of compositional instructions. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 41 (pp. 611–617).
- Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, e253.

- Lamb, Alex M, A. G., Zhang, Y., Zhang, S., Courville, A. C., & Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems*, volume 29.
- Land, M., Mennie, N., & Rusted, J. (1999). The roles of vision and eye movements in the control of activities of daily living. *Perception*, 28(11), 1311–1328.
- Lashley, K. (1951). The problem of serial order in behavior. In *Cerebral mechanisms in behavior; the Hixon Symposium* (pp. 112–136). Wiley.
- LeCun, Y. (2022). A path towards autonomous machine intelligence. Open Review, version 0.9. 2, 2022-06-27, <https://openreview.net/pdf?id=BZ5a1r-kVsf>.
- LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, volume 2 (pp. 396–404).
- Lee, E.-K., Brown-Schmidt, S., & Watson, D. G. (2013). Ways of looking ahead: Hierarchical planning in language production. *Cognition*, 129(3), 544–562.
- Lee, T. S. & Mumford, D. (2003). Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7), 1434–1448.
- Levine, D., Hirsh-Pasek, K., Pace, A., & Michnick Golinkoff, R. (2017). A goal bias in action: The boundaries adults perceive in events align with sites of actor intent. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 43(6), 916.
- Levine, S., Kumar, A., Tucker, G., & Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint*. <https://arxiv.org/abs/2005.01643>.
- Levy, A., Konidaris, G., Platt, R., & Saenko, K. (2019). Learning multi-level hierarchies with hindsight. In *The Seventh International Conference on Learning Representations, ICLR'19*.
- Li, Z., He, D., Tian, F., Chen, W., Qin, T., Wang, L., & Liu, T. (2018). Towards binary-valued gates for robust LSTM training. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 (pp. 2995–3004).
- Lieder, F. & Griffiths, T. L. (2020). Resource-rational analysis: Understanding human cognition as the optimal use of limited computational resources. *Behavioral and Brain Sciences*, 43, e1.

- Lin, J., Du, Y., Watkins, O., Hafner, D., Abbeel, P., Klein, D., & Dragan, A. (2023). Learning to model the world with language. *arXiv preprint*. <https://arxiv.org/abs/2308.01399>.
- Lin, Y., Stavans, M., & Baillargeon, R. (2022). Infants’ physical reasoning and the cognitive architecture that supports it. *Cambridge Handbook of Cognitive Development*, (pp. 168–194).
- Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., & Kipf, T. (2020). Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems*, volume 33 (pp. 11525–11538).
- Louizos, C., Welling, M., & Kingma, D. P. (2018). Learning sparse neural networks through L_0 regularization. In *The Sixth International Conference on Learning Representations*, ICLR’18.
- Lovejoy, W. S. (1991). Computationally feasible bounds for partially observed markov decision processes. *Operations Research*, 39(1), 162–175.
- Lu, L. (2020). Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5), 1671–1706.
- Madan, K., Ke, N. R., Goyal, A., Schölkopf, B., & Bengio, Y. (2021). Fast and slow learning of recurrent independent mechanisms. In *The Nineth International Conference on Learning Representations*, ICLR’21.
- Maddison, C. J., Mnih, A., & Teh, Y. W. (2017). The concrete distribution: A continuous relaxation of discrete random variables. In *The Fifth International Conference on Learning Representations*, ICLR’17.
- Marković, D., Stojić, H., Schwöbel, S., & Kiebel, S. J. (2021). An empirical evaluation of active inference in multi-armed bandits. *Neural Networks*, 144, 229–246.
- Mattar, M. G. & Lengyel, M. (2022). Planning in the brain. *Neuron*, 110(6), 914–934.
- Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., & Pathak, D. (2021). Discovering and achieving goals via world models. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 24379–24391).
- Micheli, V., Alonso, E., & Fleuret, F. (2023). Transformers are sample-efficient world models. In *The Eleventh International Conference on Learning Representations*, ICLR’23.

- Michotte, A. (1964). *The Perception of Causality*, volume 21. Routledge.
- Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2), 81–97.
- Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the Structure of Behavior*. Holt, Rinehart and Winston.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.
- Mohajerin, N. & Waslander, S. L. (2017). State initialization for recurrent neural network modeling of time-series data. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 2330–2337).
- Nachum, O., Gu, S. S., Lee, H., & Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31.
- Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., & Levine, S. (2018). Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, volume 31.
- Nasiriany, S., Pong, V., Lin, S., & Levine, S. (2019). Planning with goal-conditioned policies. In *Advances in Neural Information Processing Systems*, volume 32 (pp. 14843–14854).
- Nee, D. E. & Brown, J. W. (2012). Rostral–caudal gradients of abstraction revealed by multi-variate pattern analysis of working memory. *NeuroImage*, 63(3), 1285–1294.
- Neil, D., Pfeiffer, M., & Liu, S.-C. (2016). Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances In Neural Information Processing Systems*, volume 29 (pp. 3882–3890).
- Neitz, A., Parascandolo, G., Bauer, S., & Schölkopf, B. (2018). Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. In *Advances in Neural Information Processing Systems*, volume 31.
- Newell, A., Shaw, J. C., & Simon, H. A. (1959). Report on a general problem solving program. In *IFIP congress*, volume 256 (pp.64).
- Newtonson, D. (1973). Attribution and the unit of perception of ongoing behavior. *Journal of Personality and Social Psychology*, 28(1), 28—38.

- Newton, D. & Engquist, G. (1976). The perceptual organization of ongoing behavior. *Journal of Experimental Social Psychology*, 12(5), 436–450.
- Ni, T., Eysenbach, B., & Salakhutdinov, R. (2022). Recurrent model-free RL can be a strong baseline for many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 (pp. 16691–16723).
- Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3), 139–154. Special Issue: Dynamic Decision Making.
- O’Doherty, J. P., Cockburn, J., & Pauli, W. M. (2017). Learning, reward, and decision making. *Annual Review of Psychology*, 68(1), 73–100.
- Otte, S., Schmitt, T., Friston, K., & Butz, M. V. (2017). Inferring adaptive goal-directed behavior within recurrent neural networks. In *International Conference on Artificial Neural Networks, ICANN’17* (pp. 227–235).
- Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M., Heess, N., & Hadsell, R. (2020). Stabilizing transformers for reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research* (pp. 7487–7498).
- Parr, T., Pezzulo, G., & Friston, K. J. (2022). *Active inference: the free energy principle in mind, brain, and behavior*. The MIT Press.
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 (pp. 1310–1318).
- Pateria, S., Subagdja, B., Tan, A.-h., & Quek, C. (2021). Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5), 1–35.
- Patil, V. P., Hofmarcher, M., Dinu, M.-C., Dorfer, M., Blies, P. M., Brandstetter, J., Arjona-Medina, J., & Hochreiter, S. (2021). Align-RUDDER: Learning from few demonstrations by reward redistribution. In *The Nineth International Conference on Learning Representations, ICLR’21*.
- Paulus, M. (2011). How infants relate looker and object: evidence for a perceptual learning account of gaze following in infancy. *Developmental Science*, 14(6), 1301–1310.

- Perkins, D. N. & Salomon, G. (1992). Transfer of learning. In *International Encyclopedia of Education* (pp. 6452–6457). Pergamon Press.
- Peters, J., Janzing, D., & Schölkopf, B. (2017). *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT press.
- Peters, J. & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21(4), 682–697.
- Pezzulo, G., Rigoli, F., & Friston, K. (2015). Active inference, homeostatic regulation and adaptive behavioural control. *Progress in Neurobiology*, 134, 17–35.
- Pezzulo, G., Rigoli, F., & Friston, K. J. (2018). Hierarchical active inference: A theory of motivated control. *Trends in Cognitive Sciences*, 22(4), 294–306.
- Pinneri, C., Sawant, S., Blaes, S., Achterhold, J., Stueckler, J., Rolinek, M., & Martius, G. (2021a). Sample-efficient cross-entropy method for real-time planning. In *Proceedings of the 2020 Conference on Robot Learning*, volume 155 (pp. 1049–1065).
- Pinneri, C., Sawant, S., Blaes, S., & Martius, G. (2021b). Extracting strong policies for robotics tasks from zero-order trajectory optimizers. In *The Nineth International Conference on Learning Representations, ICLR’21*.
- Pitis, Silviu Creager, E. & Garg, A. (2020). Counterfactual data augmentation using locally factored dynamics. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 3976–3990).
- Pitis, S., Creager, E., Mandlkar, A., & Garg, A. (2022). Mocoda: Model-based counterfactual data augmentation. In *Advances in Neural Information Processing Systems*, volume 35 (pp. 18143–18156).
- Pong, V., Dalal, M., Lin, S., Nair, A., Bahl, S., & Levine, S. (2020). Skew-fit: State-covering self-supervised reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 (pp. 7783–7792).
- Pong, V., Dalal, M., Lin, S., & Nair, A. V. (2018). Multiworld: Multitask environments for RL. <https://github.com/vitchyr/multiworld>. Accessed: 23.01.2023.
- Precup, D. (2000). *Temporal abstraction in reinforcement learning*. PhD thesis, University of Massachusetts Amherst.

- Puterman, M. L. (1990). Markov decision processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science* chapter 8, (pp. 331–434). Elsevier.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. OpenAI blog <https://insightcivic.s3.us-east-1.amazonaws.com/language-models.pdf>. Accessed: 11.12.2023.
- Radvansky, G., Copeland, D., & Zwaan, R. (2005). A novel study: Investigating the structure of narrative and autobiographical memories. *Memory*, 13(8), 796–814. PMID: 16298889.
- Radvansky, G. A. & Copeland, D. E. (2000). Functionality and spatial relations in memory and language. *Memory & Cognition*, 28(6), 987–992.
- Radvansky, G. A. & Zacks, J. M. (2014). *Event Cognition*. Oxford University Press.
- Reynolds, J. R., Zacks, J. M., & Braver, T. S. (2007). A computational model of event segmentation from perceptual prediction. *Cognitive Science*, 31(4), 613–643.
- Richmond, L. L. & Zacks, J. M. (2017). Constructing experience: Event models from perception to action. *Trends in Cognitive Sciences*, 21(12), 962–980.
- Roberts, D. (2013). Thick concepts. *Philosophy Compass*, 8(8), 677–688.
- Robine, J., Höftmann, M., Uelwer, T., & Harmeling, S. (2023). Transformer-based world models are happy with 100k interactions. In *The Eleventh International Conference on Learning Representations, ICLR'23*.
- Rohe, T. & Noppeney, U. (2015). Cortical hierarchies perform bayesian causal inference in multisensory perception. *PLOS Biology*, 13(2), e1002073.
- Rosenbaum, D. A., Cohen, R. G., Jax, S. A., Weiss, D. J., & Van Der Wel, R. (2007). The problem of serial order in behavior: Lashley's legacy. *Human movement science*, 26(4), 525–554.
- Rougier, N. P., Noelle, D. C., Braver, T. S., Cohen, J. D., & O'Reilly, R. C. (2005). Prefrontal cortex and flexible cognitive control: Rules without symbols. *Proceedings of the National Academy of Sciences*, 102(20), 7338–7343.
- Rubinstein, R. & Davidson, W. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, 1, 127–190.

- Sajid, N., Ball, P. J., Parr, T., & Friston, K. J. (2021a). Active Inference: Demystified and Compared. *Neural Computation*, 33(3), 674–712.
- Sajid, N., Tigas, P., Zakharov, A., Fountas, Z., & Friston, K. (2021b). Exploration and preference satisfaction trade-off in reward-free learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*.
- Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Kuttler, H., Grefenstette, E., & Rocktäschel, T. (2021). Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Neural Information Processing Systems Datasets and Benchmarks Track*.
- Sancaktar, C., Blaes, S., & Martius, G. (2022). Curious exploration via structured world models yields zero-shot object manipulation. In *Advances in Neural Information Processing Systems*, volume 35 (pp. 24170–24183).
- Sancaktar, C., van Gerven, M. A., & Lanillos, P. (2020). End-to-end pixel-based deep active inference for body perception and action. In *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)* (pp. 1–8).
- Saxena, V., Ba, J., & Hafner, D. (2021). Clockwork variational autoencoders. In *Advances in Neural Information Processing Systems*, volume 34 (pp. 29246–29257).
- Schaal, S. (2003). Dynamic movement primitives - a framework for motor control in humans and humanoid robots. In *The International Symposium on Adaptive Motion of Animals and Machines*.
- Schank, R. C. & Abelson, R. P. (1975). Scripts, plans, and knowledge. In *IJCAI*, volume 75 (pp. 151–157).
- Schapiro, A. C., Rogers, T. T., Cordova, N. I., Turk-Browne, N. B., & Botvinick, M. M. (2013). Neural representations of events arise from temporal community structure. *Nature Neuroscience*, 16(4), 486–492.
- Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2), 234–242.
- Schmidhuber, J. (2022). Annotated history of modern ai and deep learning. *arXiv preprint*. <https://arxiv.org/abs/2212.11279>.

- Schölkopf, B. (2019). Causality for machine learning. *arXiv preprint*. <https://arxiv.org/abs/1911.10500>.
- Schölkopf, B., Locatello, F., Bauer, S., Ke, N. R., Kalchbrenner, N., Goyal, A., & Bengio, Y. (2021). Toward causal representation learning. *Proceedings of the IEEE*, 109(5), 612–634.
- Scholz, F., Gumbsch, C., Otte, S., & Butz, M. V. (2022). Inference of affordances and active motor control in simulated agents. *Frontiers in Neurorobotics*, 16.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839), 604–609.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint*. <https://arxiv.org/abs/1707.06347>.
- Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(5306), 1593–1599.
- Schwöbel, S., Kiebel, S., & Marković, D. (2018). Active Inference, Belief Propagation, and the Bethe Approximation. *Neural Computation*, 30(9), 2530–2567.
- Seitzer, M., Horn, M., Zadaianchuk, A., Zietlow, D., Xiao, T., Simon-Gabriel, C., He, T., Zhang, Z., Schölkopf, B., Brox, T., et al. (2023). Bridging the gap to real-world object-centric learning. In *The Eleventh International Conference on Learning Representations, ICLR'23*.
- Seitzer, M., Schölkopf, B., & Martius, G. (2021). Causal influence detection for improving efficiency in reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35.
- Seitzer, M., Tavakoli, A., Antic, D., & Martius, G. (2022). On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *The Tenth International Conference on Learning Representations, ICLR'22*.
- Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., & Pathak, D. (2020). Planning to explore via self-supervised world models. In *Proceedings of the 38th International Conference on Machine Learning* (pp. 8583–8592).
- Shahnazian, D., Senoussi, M., Krebs, R. M., Verguts, T., & Holroyd, C. B. (2022). Neural representations of task context and temporal order during action sequence execution. *Topics in Cognitive Science*, 14(2), 223–240.

- Shenhav, A., Musslick, S., Lieder, F., Kool, W., Griffiths, T. L., Cohen, J. D., & Botvinick, M. M. (2017). Toward a rational and mechanistic account of mental effort. *Annual Review of Neuroscience*, 40(1), 99–124. PMID: 28375769.
- Shin, Y. S. & DuBrow, S. (2021). Structuring memory through inference-based event segmentation. *Topics in Cognitive Science*, 13, 106–127.
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587), 484–489.
- Simon, H. A. (1997). *Models of Bounded Rationality: Empirically Grounded Economic Reason*, volume 3. MIT press.
- Sims, C. R. (2018). Efficient coding explains the universal law of generalization in human perception. *Science*, 360(6389), 652–656.
- Smith, J. T., Warrington, A., & Linderman, S. (2022). Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, ICLR'22.
- Solway, A., Diuk, C., Córdova, N., Yee, D., Barto, A. G., Niv, Y., & Botvinick, M. M. (2014). Optimal behavioral hierarchy. *PLOS Computational Biology*, 10(8), 1–10.
- Speer, N. K., Swallow, K. M., & Zacks, J. M. (2003). Activation of human motion processing areas during event perception. *Cognitive, Affective, & Behavioral Neuroscience*, 3(4), 335–345.
- Speer, N. K., Zacks, J. M., & Reynolds, J. R. (2007). Human brain activity time-locked to narrative event boundaries. *Psychological Science*, 18(5), 449–455.
- Stock, A. & Stock, C. (2004). A short history of ideo-motor action. *Psychological Research*, 68(2-3), 176–188.
- Sutton, R. (2019). The bitter lesson. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>. Accessed: 11.12.2023.
- Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3, 9–44.
- Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4), 160–163.

- Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An Introduction*. Cambridge, MA: MIT press, second edition edition.
- Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999a). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12.
- Sutton, R. S., Precup, D., & Singh, S. (1999b). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1), 181–211.
- Taniguchi, T., Ugur, E., Hoffmann, M., Jamone, L., Nagai, T., Rosman, B., Matsuka, T., Iwahashi, N., Oztog, E., Piater, J., & Wörgötter, F. (2019). Symbol emergence in cognitive developmental systems: A survey. *IEEE Transactions on Cognitive and Developmental Systems*, 11(4), 494–516.
- Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological Review*, 55(4), 189–208.
- Tolman, E. C. & Honzik, C. H. (1930). Introduction and removal of reward, and maze performance in rats. *University of California Publications in Psychology*, 4, 257–275.
- Tomov, M. S., Yagati, S., Kumar, A., Yang, W., & Gershman, S. J. (2020). Discovery of hierarchical representations for efficient planning. *PLOS computational biology*, 16(4), e1007594.
- Toussaint, M. A., Allen, K. R., Smith, K. A., & Tenenbaum, J. B. (2018). Differentiable physics and stable modes for tool-use and manipulation planning. *Proceedings of the Robotics: Science and Systems*.
- Trabasso, T. & van den Broek, P. (1985). Causal thinking and the representation of narrative events. *Journal of Memory and Language*, 24(5), 612–630.
- Traub, M., Otte, S., Menge, T., Karlbauer, M., Thuemmel, J., & Butz, M. V. (2023). Learning what and where: Disentangling location and identity tracking without supervision. In *The Eleventh International Conference on Learning Representations, ICLR’23*.
- van Seijen, H., Whiteson, S., & Kester, L. (2014). Efficient abstraction selection in reinforcement learning. *Computational Intelligence*, 30(4), 657–699.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

- Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., & Kavukcuoglu, K. (2017). Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 (pp. 3540–3549).
- Vlastelica, M., Blaes, S., Pinneri, C., & Martius, G. (2022). Risk-averse zero-order trajectory optimization. In *Proceedings of the 5th Conference on Robot Learning*, volume 164 (pp. 444–454).
- Washburn, M. F. (1916). *Movement and Mental Imagery: Outlines of a Motor Theory of the Complexer Mental Processes*. Houghton Mifflin.
- Watson, J. B. (1920). Is thinking merely the action of language mechanisms? *British Journal of Psychology*, 11, 87–104.
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022). Emergent abilities of large language models. *Transactions on Machine Learning Research*.
- Wiener, J. M. & Mallot, H. A. (2003). ‘Fine-to-coarse’ route planning and navigation in regionalized environments. *Spatial Cognition and Computation*, 3(4), 331–358.
- Wierstra, D., Foerster, A., Peters, J., & Schmidhuber, J. (2007). Solving deep memory POMDPs with recurrent policy gradients. In *International Conference on Artificial Neural Networks, ICANN’17* (pp. 697–706).
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4), 229–256.
- Zacks, J. M. (2010). How we organize our experience into events. *Psychological Science Agenda*, 24(4).
- Zacks, J. M., Braver, T. S., Sheridan, M. A., Donaldson, D. I., Snyder, A. Z., Ollinger, J. M., Buckner, R. L., & Raichle, M. E. (2001a). Human brain activity time-locked to perceptual event boundaries. *Nature Neuroscience*, 4(6), 651–655.
- Zacks, J. M., Speer, N. K., Swallow, K. M., Braver, T. S., & Reynolds, J. R. (2007). Event perception: a mind-brain perspective. *Psychological Bulletin*, 133(2), 273–293.
- Zacks, J. M. & Swallow, K. M. (2007). Event segmentation. *Current Directions in Psychological Science*, 16(2), 80–84.

- Zacks, J. M. & Tversky, B. (2001). Event structure in perception and conception. *Psychological Bulletin*, 127(1), 3–21.
- Zacks, J. M., Tversky, B., & Iyer, G. (2001b). Perceiving, remembering, and communicating structure in events. *Journal of Experimental Psychology: General*, 130(1), 29.
- Zadaianchuk, A., Seitzer, M., & Martius, G. (2023). Object-centric learning for real-world videos by predicting temporal feature similarities. In *Advances in Neural Information Processing Systems*, volume 36.
- Zakharov, A., Guo, Q., & Fountas, Z. (2022a). Long-horizon video prediction using a dynamic latent hierarchy. *arXiv preprint*. <https://arxiv.org/abs/2212.14376>.
- Zakharov, A., Guo, Q., & Fountas, Z. (2022b). Variational predictive routing with nested subjective timescales. In *The Tenth International Conference on Learning Representations*, ICLR'22.
- Zech, P., Renaudo, E., Haller, S., Zhang, X., & Piater, J. (2019). Action representations in robotics: A taxonomy and systematic classification. *The International Journal of Robotics Research*, 38(5), 518–562.
- Zheng, J., Schjetnan, A. G., Yebra, M., Gomes, B. A., Mosher, C. P., Kalia, S. K., Valiante, T. A., Mamelak, A. N., Kreiman, G., & Rutishauser, U. (2022). Neurons detect cognitive boundaries to structure episodic memories in humans. *Nature Neuroscience*, 25(3), 358–368.
- Zhu, P., Li, X., & Poupart, P. (2017). On improving deep reinforcement learning for POMDPs. *arXiv preprint*. <https://arxiv.org/abs/1804.06309>.
- Zénon, A., Solopchuk, O., & Pezzulo, G. (2019). An information-theoretic perspective on the costs of cognition. *Neuropsychologia*, 123, 5–18.

Acknowledgments

First of all, I would like to express my gratitude to my supervisors Martin V. Butz and Georg Martius. Thank you not only for giving me the opportunity to pursue my doctoral degree but also for giving me the freedom to explore ideas while also providing fantastic scientific guidance.

Specifically, I would like to thank Martin V. Butz for ongoing enormous support over many years and for consistently encouraging me to expand my skills and horizons. Your ability to recognize interdisciplinary connections has profoundly shaped the way I think about problems.

I am very grateful to Georg Martius for providing great support and a fantastic working environment. I have greatly benefited from your guidance and your ability to give constructive feedback on various levels, from technical questions to conceptual approaches. Your combination of enthusiasm and scientific rigor has deeply inspired me.

I have had the privilege of working in two amazing research groups. Many thanks to all my friends and colleagues at the Autonomous Learning Group and the Neuro-Cognitive Modeling Group for creating excellent and friendly working environments. In particular, I want to thank Cansu Sancaktar, Marco Bagatella, Andrii Zadaianchuk, Fedor Scholz, Maximilian Mittenbühler, and Johanna Theuer, who helped me by proofreading this thesis and providing feedback.

Many thanks go to my collaborators outside of Tübingen, without whom the presented research would not have been possible. I would like to thank Birgit Elsner and Maurits Adam for the fruitful interdisciplinary exchange and for providing invaluable expertise on developmental psychology. Many thanks to Manfred Eppe for all the stimulating discussions, our excellent collaboration, and the support beyond that. Thanks to Noor Sajid for the great collaboration and the pleasure of sharing and combining our perspectives on hierarchical predictions.

I would also like to thank Charley Wu for being on my thesis advisory committee and always providing helpful and constructive feedback and encouragement.

I would like to thank the DFG SPP “The Active Self” for providing financial support and thank their coordinators for organizing interesting events and funding my lab visits to both Nijmegen and Hamburg. Furthermore, I am grateful for the financial support of IMPRS-IS and would like to thank Leila Masri and Sara Sorce for their enthusiasm in coordinating the program.

Last but not least, many thanks go to my friends and family, especially my parents Susanne and Peter Gumbsch who have provided support, stability and assurance in so many ways for as long as I can remember. And a very special thanks goes to my wife Lea Hölz for always caring for me, helping me through stressful deadlines, adjusting to unusual working hours, and for countless other things I cannot compress into a single paragraph. Thank you for your love and unlimited personal support.

THIS THESIS WAS TYPESET using \LaTeX , originally developed by Leslie Lamport and based on Donald Knuth's \TeX . The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (x11) license, and can be found online at github.com/suchow/Dissertate or from its author, Jordan Suchow, at suchow@post.harvard.edu.