Learning Hierarchical World Models with Adaptive Temporal Abstractions from Discrete Latent Dynamics

Christian Gumbsch
University of Tübingen &
Max Planck Institute for Intelligent Systems
Tübingen, Germany
`christian.gumbsch@uni-tuebingen.de`

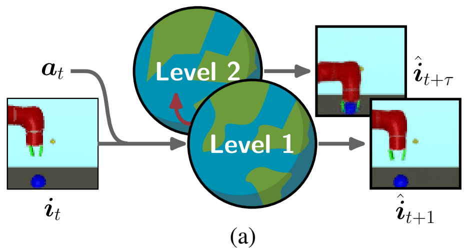
Noor Sajid
WCHN, University College London
London, UK
`noor.sajid.18@ucl.ac.uk`

Georg Martius
Max Planck Institute for Intelligent Systems
Tübingen, Germany
`georg.martius@tuebingen.mpg.de`

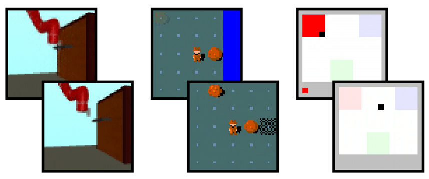
Martin V. Butz
University of Tübingen
Tübingen, Germany
`martin.butz@uni-tuebingen.de`

Abstract

Hierarchical world models have the potential to significantly improve model-based reinforcement learning (MBRL) and planning by enabling reasoning across multiple time scales. Nonetheless, the majority of state-of-the-art MBRL methods still employ flat, non-hierarchical models. The challenge lies in learning suitable hierarchical abstractions. We propose Temporal Hierarchies from Invariant Context Kernels (THICK), an algorithm that learns a world model hierarchy based on discrete latent dynamics. The lower level of the THICK world model selectively updates parts of its latent state sparsely in time, forming invariant contexts. The higher level is trained exclusively to predict situations involving these sparse context state changes. Our experiments demonstrate that THICK learns categorical, interpretable, temporal abstractions on the high level while maintaining precise low-level predictions. Furthermore, we show that the developing hierarchical predictive model can seamlessly enhance the abilities of MBRL or planning methods. We believe that THICK-like, hierarchical world models will be key for developing more sophisticated agents capable of exploring, planning, and reasoning about the future across multiple time scales.



(a)



(b)

Figure 1: **THICK world models** predict on two levels. (a) Level 1 predicts the next input ($t+1$). Level 2 predicts a future input ($t+\tau$) that is expected to change an otherwise constant latent state. (b) Exemplary situations (bottom) with hierarchical predictions (top) for: opening a door (`Multiworld-Door`), pushing a boulder into a river (`Minihack-River`), and activating a pad (`VisualPinPadThree`).

1 Introduction

The intricate hierarchical representations formed in our brains[1–6] through sensorimotor experience serve as a useful blueprint for enhancing artificial agents' planning capabilities via hierarchical world models [7, 8]. A hierarchy facilitates engagement at multiple complexity levels [9–12], contingent on the context. Humans can plan their behavior on various time scales and flexibly switch between them, such as picking up a pen for writing an invitation when organizing a party. Accordingly, the integration of hierarchical models into MBRL can foster more structured, context-dependent representations that bolster planning efficiency and effectiveness in complex, real-world tasks.

Despite significant advancements in equipping MBRL agents with the capacity to learn world models, i.e., generative, task-agnostic forward models that encode an agent's interaction with its environment from high-dimensional inputs [13–17], these models often lack an explicit hierarchical structure. Consequently, they are restricted to predictions on predefined time scales, substantially hampering their capability for long-horizon planning. The main challenge is formalizing suitable methods to learn higher-level abstractions [18–21]. Seeing that events, and even instances of the same event, typically have different time durations, fixed nested time scales are inadequate. Conversely, learning temporal abstractions from rewards like hierarchical RL (HRL) [19, 22] tethers the abstractions to a specific task, whereas world models ideally should maintain a degree of task-agnosticism.

We present a deep learning architecture that learns hierarchical world models, which we call **Temporal Hierarchies from Invariant Context Kernels** (THICK[1]). THICK adaptively discovers higher-level time scales by guiding the 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. Thus, we distill a high-level world model from discrete low-level latent state updates. A depiction of THICK's world model can be found in Fig. 1a.

We make the following key contributions:

- We **encode context-sensitive dynamics** by introducing a Context-specific Recurrent State Space Model (C-RSSM), which enhances Dreamer's [14, 16] Recurrent State Space Model (RSSM) with a sparsely changing latent factor, labeled *context*.
- We introduce a higher level predictive processing module, which learns a **hierarchical world model** with flexible time scales based on sparse context changes at the lower level.
- We enable the learning of discrete **higher level actions** by disambiguating low-level dynamics, thus enabling goal-directed high-level planning.
- We demonstrate the effectiveness of THICK in two **planning** scenarios: i) using THICK's hierarchical predictions to enhance **MBRL in long-horizon tasks**, and ii) **hierarchical model-predictive planning** (MPC) using THICK's high-level predictions to set subgoals.

2 Method

2.1 C-RSSM World Model

The RSSM proposed in Hafner et al. [14] is a recurrent neural network (RNN) world model that is commonly used for model-based reinforcement learning [15–17, 24–26]. RSSM embeds input images $\boldsymbol{i}_t$ and actions $\boldsymbol{a}_t$ into a latent state $\boldsymbol{s}_t$, predicting dynamics exclusively within this state. Inherently, all aspects of this latent state evolve continuously. However, to establish a hierarchy of world models, we require sparse latent state changes. We propose **Context-specific RSSM** (C-RSSM), which integrates a sparsely changing latent state $\boldsymbol{c}_t$ as context and a coarse prediction pathway into RSSM, as illustrated in Fig. 2. The components of C-RSSM with trainable parameters ϕ are:

Latent state:	$\boldsymbol{s}_t \leftarrow [\boldsymbol{c}_t, \boldsymbol{h}_t, \boldsymbol{z}_t]$	(1)	Posterior:	$\boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}_t \mid \boldsymbol{c}_t, \boldsymbol{h}_t, \boldsymbol{i}_t)$	(4)
Coarse Dyn.:	$\boldsymbol{c}_t = g_\phi(\boldsymbol{a}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{z}_{t-1})$	(2)	Coa. Prior:	$\hat{\boldsymbol{z}}^c_t \sim p^c_\phi(\hat{\boldsymbol{z}}^c_t \mid \boldsymbol{a}_{t-1}, \boldsymbol{c}_t, \boldsymbol{z}_{t-1})$	(5)
Precise Dyn.:	$\boldsymbol{h}_t = f_\phi(\boldsymbol{a}_{t-1}, \boldsymbol{c}_t, \boldsymbol{h}_{t-1}, \boldsymbol{z}_{t-1})$	(3)	Pre. Prior:	$\hat{\boldsymbol{z}}^h_t \sim p^h_\phi(\hat{\boldsymbol{z}}^h_t \mid \boldsymbol{c}_t, \boldsymbol{h}_t)$	(6)

Equations in red are exclusive to the C-RSSM.[2] We extend RSSM's latent state $\boldsymbol{s}_t$ to include three parts (Eq. 1): a stochastic state $\boldsymbol{z}_t$, a continuously updated, high-dimensional, deterministic state

[1]In philosophy, "thickness" refers to descriptions that fuse facts with contextual elements, while "thinness" denotes factual, neutral descriptions [23]. When adapted to world models, a THICK model presents an accurate representation of the world with contextual interpretation.

[2]Removing $\boldsymbol{c}$ in all black equations recovers the equations for the RSSM (Eqns. 1,3,4,6).

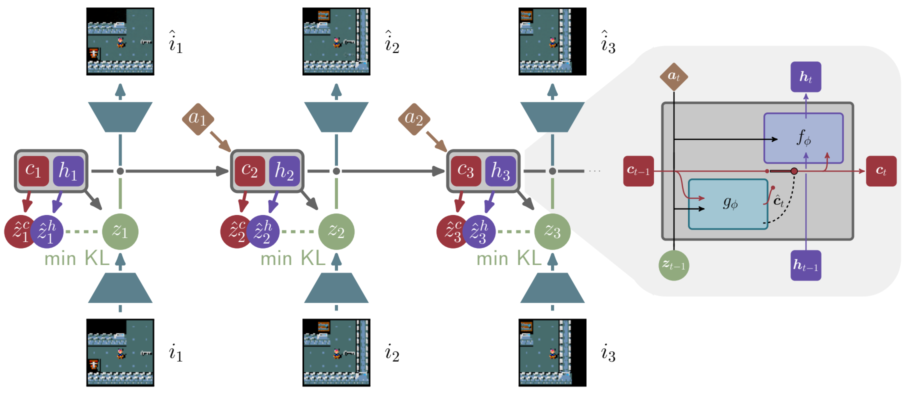


Figure 2: **C-RSSM world model**. Left: The C-RSSM predicts dynamics within a latent states with a stochastic part $\boldsymbol{z}_t$ and two deterministic parts $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ The network predicts the next stochastic state $\boldsymbol{z}_t$ via two pathways: It makes *coarse* predictions $\hat{z}^c_t$ based mainly on $\boldsymbol{c}_t$ and *precise* predictions $\hat{z}^h_t$ based on $\boldsymbol{h}_t$. Right: Internally, the sparsely changing context $\boldsymbol{c}_t$ is updated via a GateL0RD cell [27] g_ϕ with a sparsely-operated update gate. A GRU cell [28] f_ϕ is used to continuously change $\boldsymbol{h}_t$.

$\boldsymbol{h}_t$, and a discretely changing, low-dimensional context $\boldsymbol{c}_t$. At time t the C-RSSM first updates the context $\boldsymbol{c}_t$ (Eq. 2), where actual $\boldsymbol{c}_t$ changes only occur *sparsely in time*. Note that $\boldsymbol{c}_t$ updates do not depend on $\boldsymbol{h}_t$, thus creating a coarse processing pathway that is independent of $\boldsymbol{h}_t$. Next, C-RSSM updates $\boldsymbol{h}_t$ via a GRU [28] cell f_ϕ (Eq. 3). Using both deterministic components of its latent state, C-RSSM makes two predictions about the next stochastic state $\hat{\boldsymbol{z}}^h_t$: *i*) a *precise* prior based on its deterministic latent states (Eq. 6), and *ii*) a *coarse* prior $\hat{\boldsymbol{z}}^c_t$ based on the context, stochastic state, and action, ignoring $\boldsymbol{h}_t$ (Eq. 5). Given the input image $\boldsymbol{i}_t$, C-RSSM finally updates its posterior $\boldsymbol{z}_t$ (Eq. 4). Following DreamerV2 [16], we sample $\boldsymbol{z}_t$ from a vector of categorical distributions.

C-RSSM encodes its interactions with the world in its latent states $\boldsymbol{s}_t$. Besides encoding latent system dynamics, $\boldsymbol{s}_t$ is trained to predict observable variables y_t of the outside world. Two output heads o_ϕ generate precise and coarse predictions:

$$\text{Precise prediction:} \quad \hat{y}_t \sim o_\phi(\hat{y}_t \mid \boldsymbol{s}_t) \quad (7) \qquad \text{Coarse prediction:} \quad \hat{y}^c_t \sim o^c_\phi(\hat{y}^c_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t). \quad (8)$$

We follow DreamerV2 [16] and predict the input image $\boldsymbol{i}_t$, the reward r_t, and future reward discount γ_t [3], i.e. $y_t \leftarrow \{\boldsymbol{i}_t, r_t, \gamma_t\}$.

Sparse context updates The latent context code $\boldsymbol{c}_t$ should change sparsely in time, ideally at distinct, environment-specific, transition points. This is why the coarse dynamics g_ϕ (Eq. 2) are modeled by a GateL0RD cell [27]. Briefly, GateL0RD is a gated RNN that learns sparsely changing latent states $\boldsymbol{c}_t$ by introducing an update gate and an auxiliary loss term $\mathcal{L}^{\text{sparse}}$, which penalizes context changes $\boldsymbol{\Delta c}_t$ via L_0-norm regularization. GateL0RD is detailed in Suppl. D.1.

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

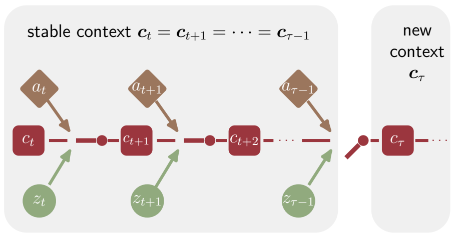
$$\mathcal{L}(\phi) = \mathbb{E}_{q_\phi}\left[\beta^{\text{pred}}\mathcal{L}^{\text{pred}}(\phi) + \beta^{\text{KL}}\mathcal{L}^{\text{KL}}(\phi) + \beta^{\text{sparse}}\mathcal{L}^{\text{sparse}}(\phi)\right], \tag{9}$$

including the prediction loss $\mathcal{L}^{\text{pred}}$, the KL loss $\mathcal{L}^{\text{KL}}$, and sparsity loss $\mathcal{L}^{\text{sparse}}$ with hyperparameters β^{pred}, β^{KL}, and β^{sparse} scaling their impact. The prediction loss $\mathcal{L}^{\text{pred}}$ drives the system to predict perceptions y via its output heads o_ϕ. Compared to RSSM, we also account for the coarse predictions (Eq. 8). The KL loss $\mathcal{L}^{\text{KL}}$ minimizes the KL divergence between coarse and fine prior predictions p^c_ϕ and p^h_ϕ and the approximate posterior q_ϕ. The sparsity loss $\mathcal{L}^{\text{sparse}}$ encourages consistency of context $\boldsymbol{c}_t$. The exact loss functions are provided in Suppl. D.2. We set β^{pred} and β^{KL} to DreamerV2 defaults [16] and modify the sparsity loss scale β^{sparse} depending on the scenario (see Suppl. B).

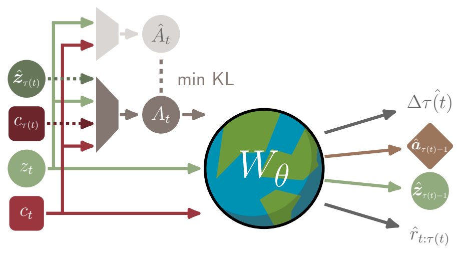
2.2 Hierarchical World Model

To learn a hierarchical world model, we leverage C-RSSM's discrete context $\boldsymbol{c}_t$ updates to create coarse predictions by means of our **Temporal Hierarchies from Invariant Context Kernels** (THICK)

[3] The discount γ_t is set to 0 if an episode terminates and set to a fixed value γ otherwise.



(a) Time series discretization



(b) Level 2 world model

Figure 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 a 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 z_t and c_t. High-level actions (A_t or $\hat{A}_t$) distinguish high-level outcomes.

algorithm (Figure 1a). C-RSSM's world model w_ϕ segments sequences into periods of stable context activity ($c_t = c_{t+1} = \cdots = c_{\tau-1}$), interspersed with sparse context updates (cf. Fig. 3a). THICK uses these discrete context dynamics as an adaptive timescale for training its high-level network W_θ. The core assumption is that states prompting low-level context updates coincide with crucial changes in latent generative factors. These key states are predicted by the high-level network W_θ, while 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 $i_{1:T}$, actions $a_{1:T}$, and episode termination flags $d_{1:T}$. The sequence is passed through the low-level model w_ϕ to obtain a sequence of contexts $c_{1:T}$. Targets are defined as all time steps τ with context changes, i.e., where $c_\tau \neq c_{\tau-1}$ or the episode ends. We define the function $\tau(\cdot)$ as

$$\tau(t) = \min\big(\{\tau \mid \tau > t \wedge (c_\tau \neq c_{\tau-1} \vee d_\tau = 1)\}\big). \tag{10}$$

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.

High-level targets We predict all variables at $\tau(t) \in \mathcal{T}$ that may cause a context change or are needed for planning across a context change: $\hat{z}_{\tau(t)-1}, \hat{a}_{\tau(t)-1}, \Delta\hat{\tau}(t), \hat{r}^\gamma_{t:\tau(t)}$ (cf. Fig. 3b). In particular, we predict the stochastic states $\hat{z}_{\tau(t)-1}$ and actions $\hat{a}_{\tau(t)-1}$ immediately before a context change at time $\tau(t)$, because both can cause an update of $c_{\tau(t)}$ (see Eq. 2). Intuitively, this means that observations, e.g. seeing something falling, as well as actions, e.g. catching something, could cause c_t to change. We furthermore predict the elapsed time $\Delta\tau(t)$ and the accumulated discounted reward $r^\gamma_{t:\tau(t)}$, which allow taking variable duration and rewards into account when evaluating potential outcomes following a high-level prediction:

$$\text{Elapsed time:} \quad \Delta\tau(t) = \tau(t) - t \qquad \text{Accumulated rewards:} \quad r^\gamma_{t:\tau(t)} = \sum_{\delta=1}^{\Delta\tau(t)-1} \gamma^\delta r_{t+\delta} \tag{11}$$

High-level inputs To predict high-level targets, we use the low-level stochastic state z_t and context 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. Accordingly, akin to actions on the low level, we create self-organizing high-level "actions" A_t, similar to skills or options [19]. A_t encode a categorical distribution over probable next context changes. To learn A_t, the high-level world model implements a *posterior* action encoder Q_θ and a *prior* action encoder P_θ (cf. Fig. 3b). Overall, the high-level world model W_θ can be formalized as

$$\text{Posterior: } A_t \sim Q_\theta(A_t \mid c_t, z_t, c_{\tau(t)}, z_{\tau(t)}) \tag{12}$$

$$\text{Action: } \hat{a}_{\tau(t)-1} \sim F^{\hat{a}}_\theta\big(\hat{a}_{\tau(t)-1} \mid A_t, c_t, z_t\big) \tag{13}$$

$$\text{State: } \hat{z}_{\tau(t)-1} \sim F^{\hat{z}}_\theta\big(\hat{z}_{\tau(t)-1} \mid A_t, c_t, z_t\big) \tag{14}$$

$$\text{Prior: } \hat{A}_t \sim P_\theta(\hat{A}_t \mid c_t, z_t) \tag{15}$$

$$\text{Time : } \Delta\hat{\tau}(t) \sim F^{\hat{\tau}}_\theta\big(\Delta\hat{\tau}(t) \mid A_t, c_t, z_t\big) \tag{16}$$

$$\text{Reward: } \hat{r}^\gamma_{t:\tau(t)} \sim F^{\hat{r}}_\theta\big(\hat{r}^\gamma_{t:\tau(t)} \mid A_t, c_t, z_t\big) \tag{17}$$

for learnable parameters θ. The posterior Q_θ receives not only c_t and z_t as its input but also privileged information about the actually encountered next context, i.e. $c_{\tau(t)}$ and $z_{\tau(t)}$ (Eq. 12), which leads to

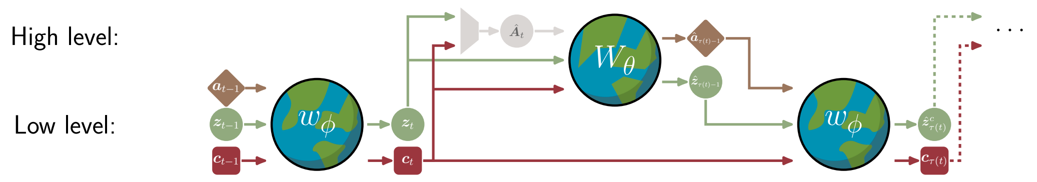


Figure 4: **Temporal abstract predictions of THICK world models**. From a low-level context $\boldsymbol{c}_t$ and stochastic state $\boldsymbol{z}_t$, the high level predicts a future stochastic state $\hat{\boldsymbol{z}}_{\tau(t)-1}$ as well as the action $\hat{\boldsymbol{a}}_{\tau(t)-1}$. With these predictions, the context $\boldsymbol{c}_{\tau(t)}$ is updated on the low level together with the coarse prior $\hat{\boldsymbol{z}}^c_{\tau(t)}$. This process can be repeated (dashed line) to create a temporal abstract roll-out.

the emergence of individualized, result-conditioned action encodings in $\boldsymbol{A}_t$. The prior P_θ learns a distribution over $\hat{\boldsymbol{A}}_t$ approximating the posterior without the privileged information (Eq. 15). During training, THICK samples the high-level action $\boldsymbol{A}_t$ from Q_θ. During evaluation we sample from the prior P_θ instead. We model $\hat{\boldsymbol{A}}_t$ and $\boldsymbol{A}_t$ as one-hot encoded categorical variables.

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

$$\mathfrak{L}(\theta) = \mathbb{E}\big[\alpha^{\text{pred}}\mathfrak{L}^{\text{pred}}(\theta) + \alpha^{\text{KL}}\mathfrak{L}^{\text{KL}}(\theta)\big], \tag{18}$$

with hyperparameters α^{pred} and α^{KL} scaling the prediction $\mathfrak{L}^{\text{pred}}$ and action $\mathfrak{L}^{\text{KL}}$ loss terms, respectively. The prediction loss is the summed negative log-likelihood of the high-level 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. D.3.

Summary Our THICK world model augments traditional flat world models by a high-level, which predicts situations where low-level context changes occur. This augmentation allows for seamless transitions between coarse, low-level and abstract, high-level predictions. Given a context $\boldsymbol{c}_t$, stochastic state $\boldsymbol{z}_t$, and sampled high-level action $\hat{\boldsymbol{A}}_t$, the high-level model W_θ predicts a scenario ($\hat{\boldsymbol{a}}_{\tau(t)-1}$, $\hat{\boldsymbol{z}}_{\tau(t)-1}$) immediately prior to the next context change. By feeding this prediction into C-RSSM, we can predict the new context $\boldsymbol{c}_{\tau(t)}$ (Eq. 2) and compute a coarse prior estimate of the corresponding stochastic state $\hat{\boldsymbol{z}}^c_{\tau(t)}$ (Eq. 5). To create longer *temporal abstract roll-outs*, we repeat this process by feeding $\boldsymbol{c}_{\tau(t)}$ and $\hat{\boldsymbol{z}}^c_{\tau(t)}$ again into W_θ. Fig. 4 visualizes this process.

2.3 Downstream applications of THICK world models

World models have been applied in many downstream tasks, including MBRL [13, 15–17], exploration [24, 29, 30], or model-predictive control (MPC) [14, 29, 30]. 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.

2.3.1 THICK Dreamer: MBRL with hierarchical rollouts

Dreamer [15–17] learns behavior by training an actor and a critic from "imagined" roll-outs of an RSSM world model. More specifically, the world model imagines a sequence of states $\boldsymbol{s}_{t:t+H}$ from a starting state $\boldsymbol{s}_t$ given an actor-generated action sequence $\boldsymbol{a}_{t:t+H}$. Dreamer computes the general λ−return $V^\lambda(\boldsymbol{s}_t)$ [31] for every $\boldsymbol{s}_t$ and the critic v_ξ is trained to regress $V^\lambda(\boldsymbol{s}_t)$.

In sparse reward tasks, one challenge is reward propagation for training the critic [32]. Here, Dreamer faces a difficult trade-off: Long roll-outs (large H) speed up 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 contexts $\boldsymbol{c}_{\tau(t)}$ and stochastic states $\boldsymbol{z}_{\tau(t)}$ of temporal abstract predictions. Like Dreamer, we first imagine a low-level roll-out of H states $\boldsymbol{s}_{t:t+H}$. For every time step t in the roll-out, we additionally predict a temporal abstract outcome $\boldsymbol{c}_{\tau(t)}$ and $\boldsymbol{z}_{\tau(t)}$ and estimate a long horizon value V^{long} as

$$V^{\text{long}}(\boldsymbol{s}_t) = \hat{r}^\gamma_{t:\tau(t)} + \gamma^{\Delta\hat{t}}\big(\hat{r}^c_{\tau(t)} + \hat{\gamma}^c_{\tau(t)} v_\chi(\hat{\boldsymbol{c}}_{\tau(t)}, \hat{\boldsymbol{z}}_{\tau(t)})\big), \tag{19}$$

with all variables predicted via the THICK world model and immediate rewards via Eq. 8 of C-RSSM given THICK's world model predictions (cf. also supplementary Alg. 1). We estimate the value of a

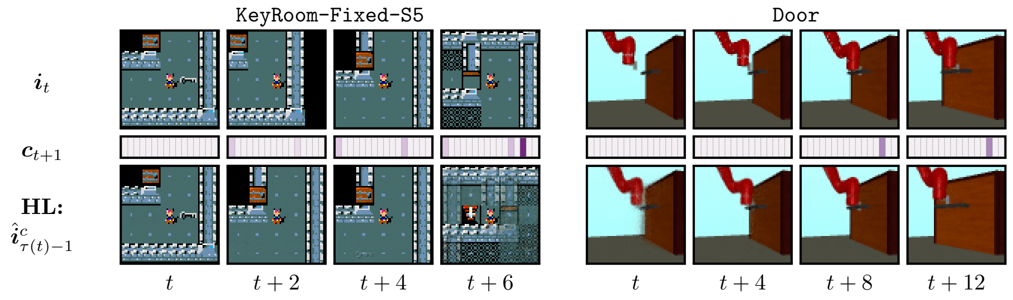


Figure 5: **Context changes**. We show the input images $\boldsymbol{i}_t$, 16-dim. contexts $\boldsymbol{c}_{t+1}$ and reconstructions $\hat{\boldsymbol{i}}^c_{\tau(t)-1}$ of the high-level predictions. For `KeyRoom` the context changes when picking up a key, opening a door (here from a diagonally adjacent grid) or exiting the room. In `Door` the context changes when the robot grabs the handle. The high level predicts the states before the next changes.

state $\boldsymbol{s}_t$ as a mixture of short- and long-horizon estimates with

$$V(\boldsymbol{s}_t) = \psi V^\lambda(\boldsymbol{s}_t) + (1-\psi) V^{\text{long}}(\boldsymbol{s}_t), \tag{20}$$

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^c_χ to regress the value estimate using a squared loss:

$$\mathcal{L}(\vartheta) = \mathbb{E}_{p_\phi}\Big[\sum_{t=1}^{H} \frac{1}{2}\Big(v_\vartheta(\boldsymbol{s}_t) - \text{sg}\big(V(\boldsymbol{s}_t)\big)\Big)^2\Big], \tag{21}$$

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

2.3.2 THICK PlaNet: Hierarchical MPC

The original RSSM was proposed in PlaNet [14] as a world model for MPC. PlaNet searches for the optimal action sequence $\boldsymbol{a}^*_{t:t+H}$ to maximize the predicted returns $\hat{r}_{t:t+H}$. To find $\boldsymbol{a}^*_{t:t+H}$ PlaNet employs zero-order trajectory optimization via the cross entropy method (CEM) [33]. Once $\boldsymbol{a}^*_{t:t+H}$ is identified, the initial action $\boldsymbol{a}^*_t$ is executed and the procedure is iteratively continued.

CEM optimizes randomly sampled trajectories. Sampling a good action sequence is exponentially harder for increasing task horizons. We hypothesize that on a high level 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 $\boldsymbol{A}_{t:K}$ with length K recursively as

$$R(\boldsymbol{A}_{t:K}) = \hat{r}^\gamma_{t:\tau(t)} + \gamma^{\Delta \hat{t}} \begin{cases} \hat{r}^c_{\tau(t)} + \hat{\gamma}^c_{\tau(t)} R(\boldsymbol{A}_{\tau(t):K}) & \text{for } t < K, \\ \hat{r}^c_{\tau(t)} & \text{for } t = K \end{cases} \tag{22}$$

with all variables predicted via a temporal abstract rollout (see Sec. 2.2). We search for the optimal sequence $\hat{\boldsymbol{A}}^*_{t:K}$ maximizing $R(\cdot)$ with Monte Carlo Tree Search [34]. Based on the first action $\hat{\boldsymbol{A}}^*_t$ we sample a subgoal $\hat{\boldsymbol{z}}^{\text{goal}}_t \sim F_\theta(\hat{\boldsymbol{z}}^{\text{goal}}_t | \hat{\boldsymbol{A}}^*_t, \boldsymbol{c}_t, \boldsymbol{z}_t)$. This subgoal is valid if it has not been reached or nothing drastically changes in the environment. Thus, we replan on the high level if the context changes. We apply CEM on the low level to reach $\boldsymbol{z}^{\text{goal}}_t$ while also maximizing task return with

$$\boldsymbol{a}^*_{t:t+H} = \underset{\boldsymbol{a}_{t:t+H}}{\arg\max} \sum_{t'=t}^{t+H} \hat{r}_{t'} + \kappa \, \text{sim}(\boldsymbol{z}_{t'}, \boldsymbol{z}^{\text{goal}}_t) \qquad \text{with} \quad \hat{r}_{t'} \sim o_\phi(\hat{r}_{t'} \mid \boldsymbol{s}_{t'}), \tag{23}$$

for a planning horizon H. The function $\text{sim}(\cdot)$ is a similarity measure between subgoal $\boldsymbol{z}^{\text{goal}}_t$ and a state $\boldsymbol{z}_t$. The hyperparameter κ controls the trade-off between external and internal reward. Previously, similarity between Gaussian distributed $\boldsymbol{z}_t$ of the RSSM was estimated using cosine similarity [25, 35]. However, for the categorically distributed states $\boldsymbol{z}_t$, the cosine similarity can be low even when they stem from the same distribution. Instead we use the cosine similarity of the logits, i.e.

$$\text{sim}(\boldsymbol{z}_t, \boldsymbol{z}^{\text{goal}}_t) = \frac{\boldsymbol{l}_t \cdot \boldsymbol{l}^{\text{goal}}_t}{\|\boldsymbol{l}_t\| \|\boldsymbol{l}^{\text{goal}}_t\|}, \tag{24}$$

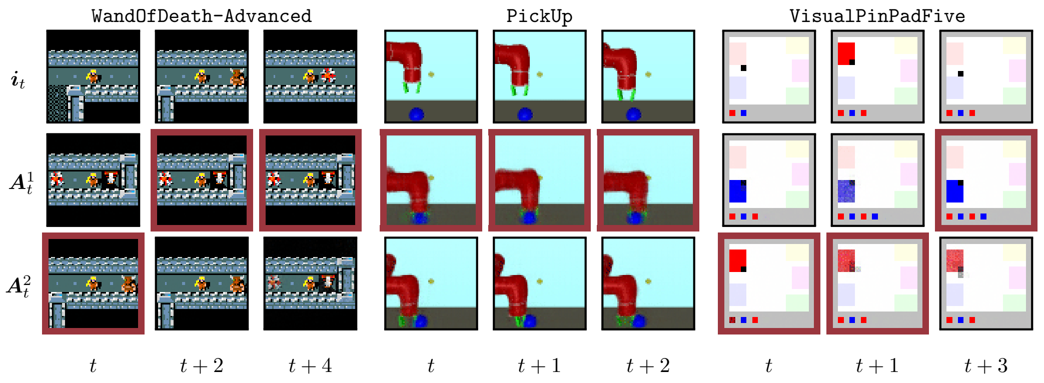


Figure 6: **Visualization of high-level actions $\boldsymbol{A}_t$.** The top row shows the input image $\boldsymbol{i}_t$. Image reconstructions $\hat{i}^c_{\tau(t)-1}$ are shown for two high-level actions $\boldsymbol{A}^1_t$ and $\boldsymbol{A}^2_t$. Red outlines depict which action $\hat{\boldsymbol{A}}_t$ was sampled. Exemplar learned high-level actions are: Exiting the dungeon or attacking a monster (left), grasping or pushing an object (center), stepping on a pad (right).

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

3 Results

We empirically evaluate THICK to answer the following questions:

- **Can THICK learn context-sensitive temporal abstractions?** We illustrate that the high-level world model can discern meaningful and explainable context-sensitive temporal abstractions across various scenarios (Sec. 3.1).
- **Can THICK's hierarchical predictions improve MBRL?** We demonstrate that THICK Dreamer, employing hierarchical roll-outs, achieves higher returns compared to Dreamer using flat roll-outs in long-horizon tasks with sparse rewards (Sec. 3.2).
- **Can THICK's world model be used to plan hierarchically?** We show that MPC with THICK world models is better at solving long-horizon tasks compared to flat world models (Sec. 3.3).

We evaluate our THICK world models in various scenarios. **MiniHack** [36] is a sandbox framework for designing RL environments based on Nethack [37]. We test our system on benchmark problems as well as newly created tasks. The problems in MiniHack have hierarchical structures in which subgoals need to 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 agent's 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.1.

VisualPinPad [35] is a suite of visual, long-horizon RL problems. Here an agent (black square) needs to activate a fixed sequence of pads by stepping on them to receive a sparse reward. We use three levels of difficulties based on the number of pads and target sequence length (three, four, five).

MultiWorld [38] is a suite of robotic manipulation tasks for visual reinforcement learning [39, 40]. 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.2.

3.1 Explainable Contexts and Hierarchical Predictions

First, we analyze the predictions of THICK world models across diverse tasks. Two example sequences are displayed in Fig. 5. In MiniHack, context alterations typically coincide with item collection, map changes, area exploration, or dungeon exits. In Multiworld, context changes occur due to object interactions or when workspace boundaries are reached. In Visual Pin Pad, stepping on pads can

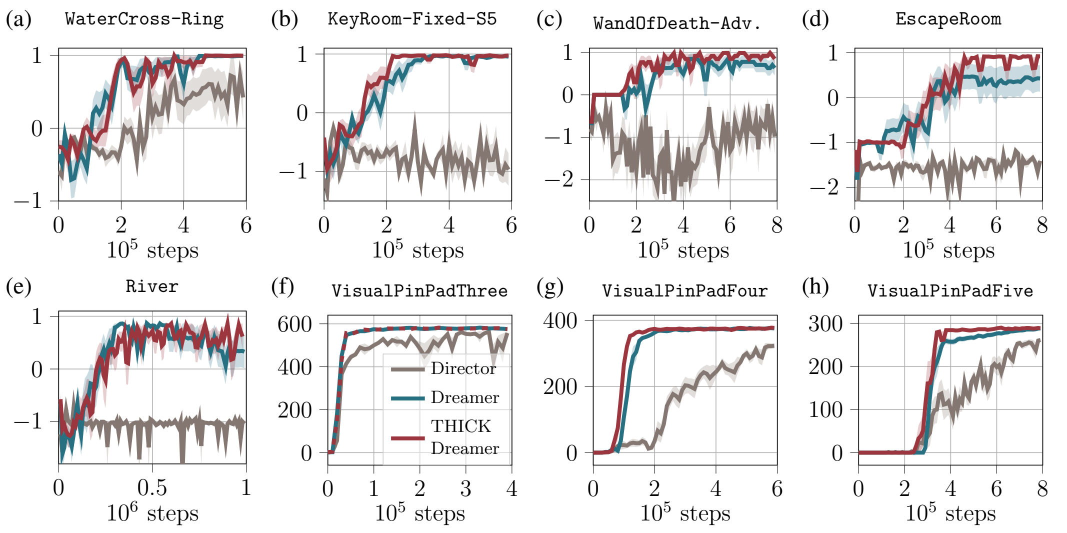


Figure 7: **MBRL results**. Each graphic plots the mean returns during evaluation for several MiniHack tasks (a-e) and for the Visual Pin Pad suite (f-h) using 7 seeds. Shaded areas depict standard error.

prompt context changes. The high-level model anticipates 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. 5, $t + 6$). Nevertheless, the low-level model consistently predicts the next frames accurately, as illustrated in Fig. 1b.

Abstract action representations $\boldsymbol{A}_t$ emerge on the high level, as illustrated in Fig. 6. These actions seem to categorically encode different agent-world interactions, e.g., grasping or pushing a ball in `PickUp`. The prior Q_θ learns to sample likely actions based on the likelihood of their outcomes (red frames in Fig. 6). If there are more actions $\boldsymbol{A}_t$ than necessary, different actions encode the same outcome. We provide more examples and analysis of the predictions and contexts in Suppl. F.1 and on our website.

3.2 Model-Based Reinforcement Learning

We investigate whether hierarchical roll-outs can improve MBRL in various MiniHack tasks. Here we compare THICK Dreamer, which uses hierarchical predictions to train a flat policy, to DreamerV2 [16] using non-hierarchical roll-outs. Additionally we compare against Director [35] another hierarchical method derived from Dreamer, which trains hierarchical policies using a flat world model. Fig. 15c–7e show that THICK Dreamer matches or outperforms flat Dreamer in all tasks in terms of sample efficiency or overall rewards. The advantage of THICK Dreamer is more pronounced in long-horizon tasks that require completing multiple subgoals to accomplish a task (e.g. picking up a ring to gain access to a key to open a door in `EscapeRoom` in contrast to the simpler problem of finding a key to open a door in `KeyRoom`). Director fails to learn most MiniHack tasks and only manages to solve the simplest task (`WaterCross-Ring`).

We further analyze the effect of task horizon on the performance in VisualPinPad. VisualPinPad poses two challenges: exploration and learning long-horizon behavior. To analyze the latter in isolation, we sidestep the challenge of discovering the sparse rewards by initially filling the replay buffer of all models with 1M data points collected from exploration with Plan2Explore [24] (details in Suppl. F.2). Fig. 7f–7h shows the performance of Director, Dreamer, and THICK Dreamer. THICK Dreamer matches Dreamer in `VisuaLPinPadThree` and outperforms its counterparts in terms of sample efficiency in the more challenging tasks.[4] Thus, the integration of hierarchical prediction for training a flat policy in THICK Dreamer seems to be better suited for learning long-horizon behavior than the hierarchical policies of Director and the non-hierarchical approach of Dreamer.

[4]Previously, Hafner et al. [35] reported that Director outperforms Dreamer in VisualPinPad. We hypothesize that this improvement mainly comes from more sophisticated exploration, which is not necessary in our setting.

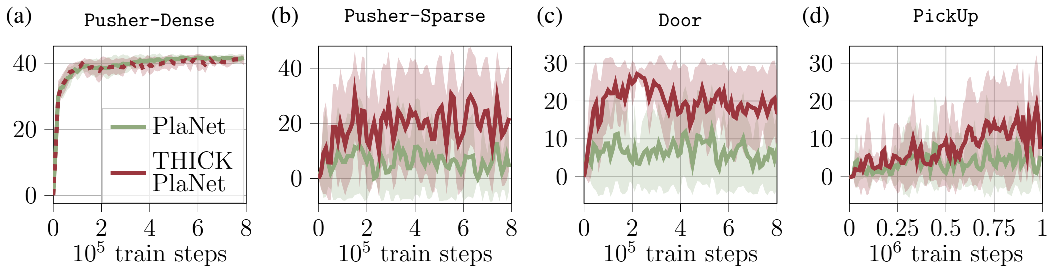


Figure 8: **MPC Multiworld results**. Each graphic plots the mean returns for zero-shot planning in Multiworld over training steps using 10 seeds. Shaded areas depict the standard deviation.

3.3 Zero-Shot Model-Predictive Control

Lastly, we analyze whether our hierarchical predictions are suitable for robust planning by comparing THICK PlaNet and PlaNet [14] in the Multiworld suite. We consider the challenging setup of MPC for models trained on an offline dataset of 1M samples collected by Plan2Explore [24]. Figure 8 shows the zero-shot performance over training steps. For `Pusher-Dense`, i.e. a short-horizon task[5] with dense rewards, there is no notable difference between applying THICK PlaNet or flat PlaNet. When rewards are sparse (`Pusher-Sparse`) or the task horizon is long (`Door` and `PickUp`), THICK PlaNet achieves higher returns than PlaNet. Additionally, the subgoals set by the high level, shown in Suppl. F.3, are easily interpretable, which makes the behavior of the system more explainable.

4 Related work

Sparsity in RNNs: Learning hierarchical RNNs from sparse activity was proposed in Schmidhuber [7], where a high level would become active based on low-level errors. Subsequently, there has been a lot of research on fostering sparsity in RNNs [27, 41–45], which we compare in Suppl. C.

Temporal abstract predictions: One main challenge for learning task-agnostic, temporal abstract predictions is segmenting a sequence into meaningful units. Discrete latent dynamics were previously used to model goal-anticipatory gaze behavior [46]. Alternative segmentation techniques are identifying easy-to-predict bottleneck states [47–49], using a set of fixed time scales [44, 50], segmenting unexpected prediction errors [51], or training regularized boundary detectors [52, 53] (further details in Suppl. C).

Hierarchical RL (HRL): HRL is an orthogonal research direction to the hierarchical world models of this work. In HRL a high-level policy either selects a low-level policy or provides goals or rewards for a low level [19, 22]. In contrast, our THICK Dreamer uses high-level predictions to train a flat RL agent. This allows faster credit assignment for tasks with sparse or delayed rewards, which was previously tackled using reward redistribution [54]. In common HRL approaches, the high level operates based on fixed time scales [35, 55–57] or task-dependently based on subgoal completion [58, 59]. In THICK world models, the high level is learned time- and task-independently purely from predictions and latent state regularization.

Transformer-based world models: Transformers [60] can also improve the learning of long-term dependencies in world models [61–63]. However, applying Transformers does not result in temporal abstract predictions that can serve as subgoals for hierarchical planning, necessary for THICK PLaNet. Importantly, THICK world models provide a practical implementation that allows both.

5 Conclusion

We have introduced C-RSSM and THICK—fully self-supervised learning methods that construct hierarchical world models. Our method expands previous RSSMs by context-conditioning them. The resulting context-conditioned C-RSSM learns to develop a sparse, dynamically evolving latent context. By imposing a sparsity objective, C-RSSM self-organizes its context codes and tends to update them only at critical time points, where unobservable, prediction-relevant properties of the environment change, including, for example, agent-object interactions or environmental shifts. As a result, C-RSSM tends to generate an approximation of a hierarchical, hidden Markov world model,

[5]Since the puck starts between the gripper and goal, the task can be solved by directly moving to the goal.

where the context conditions approximate lower-level Markov models. On the higher level, THICK tends to learn a self-organizing hidden Markov model, which is trained to anticipate context-altering transition states. To account for multiple possible lower-level context transitions, THICK develops categorical high-level action representations that both are interpretable and can encode a variety of outcomes—from controllable actions, like object interactions, to external environmental factors, such as entering a novel environmental area. As a result, THICK world models enable the generation of temporally abstract predictions, abstracting over unpredictable or unknown details.

Here, we employed THICK to establish a two-level hierarchy of world models. However, given THICK's segmentation approach, which is based on piecewise constant latent states, it could be applied at any hierarchical level. Consequently, by designing each level as a C-RSSM, THICK could potentially be extended to seamlessly build an N-level hierarchy of world models. Ideally, though, a recursive approach may be pursued in the future, such that THICK may be enabled to selectively mix lower- and higher-level contextual states.

Limitations One restriction of our method is that it relies on setting the hyperparameter β^{sparse}, which scales the sparsity loss and, thus, determines the high-level segmentation. Ideally, this hyperparameter should be tuned for every task. However, we found that the same value works well for all tasks of the same suite. Furthermore, our downstream applications have similar restrictions to whatever method they build upon, except for improving long-horizon learning. For example, if Dreamer never discovers a solution to a task, our THICK world models cannot break it down into components.

Future directions We have shown how to obtain hierarchical models with adaptive temporal abstraction and how these can improve both MBRL and MPC in long-horizon tasks. We see great potential of THICK world models as a tool to build sophisticated, hierarchical agents, that explore and plan their behavior across multiple time scales. For example, THICK world models could be used for hierarchical RL, by training a high-level policy based on temporal abstract roll-outs to select goals for a low-level policy, similar to approaches using goal-conditioned Dreamer [25, 35]. Another potential extension could be temporal abstract uncertainty-based exploration [24, 30], where the agent seeks out states that cause uncertainty for the high-level predictions.

Acknowledgments and Disclosure of Funding

We thank Karl Friston for the helpful discussion and we thank Marco Bagatella for the valuable feedback on the draft. We acknowledge the support from the German Federal Ministry of Education and Research through the Tübingen AI Center (FKZ: 01IS18039B). This research was funded by the German Research Foundation (DFG) within Priority-Program SPP 2134 – project "Development of the agentive self" (BU 1335/11-1, EL 253/8-1). Georg Martius and Martin Butz are members of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. We thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Christian Gumbsch. Noor Sajid is grateful for support from the Medical Research Council (MR/S502522/1) and the 2021-2022 Microsoft PhD Fellowship.

References

[1] Tai Sing Lee and David Mumford. Hierarchical bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448, 2003.

[2] Nicolas P Rougier, David C Noelle, Todd S Braver, Jonathan D Cohen, and Randall C O'Reilly. Prefrontal cortex and flexible cognitive control: Rules without symbols. *Proceedings of the National Academy of Sciences*, 102(20):7338–7343, 2005.

[3] Matthew Botvinick and Ari Weinstein. Model-based hierarchical reinforcement learning and human action control. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 369(1655):20130480, 2014.

[4] Tim Rohe and Uta Noppeney. Cortical hierarchies perform bayesian causal inference in multisensory perception. *PLoS biology*, 13(2):e1002073, 2015.

[5] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

[6] Karl J Friston, Richard Rosch, Thomas Parr, Cathy Price, and Howard Bowman. Deep temporal models and active inference. *Neuroscience & Biobehavioral Reviews*, 90:486–501, 2018.

[7] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

[8] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

[9] Karl Friston, Rosalyn J Moran, Yukie Nagai, Tadahiro Taniguchi, Hiroaki Gomi, and Josh Tenenbaum. World model learning and inference. *Neural Networks*, 2021.

[10] Gabriel A Radvansky and Jeffrey M Zacks. *Event cognition*. Oxford University Press, 2014.

[11] Martin V Butz. Toward a unified sub-symbolic computational theory of cognition. *Frontiers in psychology*, 7:925, 2016.

[12] Momchil S Tomov, Samyukta Yagati, Agni Kumar, Wanqian Yang, and Samuel J Gershman. Discovery of hierarchical representations for efficient planning. *PLoS computational biology*, 16(4):e1007594, 2020.

[13] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[14] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR, 2019.

[15] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[16] Danijar Hafner, Timothy P Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. In *International Conference on Learning Representations*, 2020.

[17] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023.

[18] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[19] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[20] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

[21] Harm van Seijen, Shimon Whiteson, and Leon Kester. Efficient abstraction selection in reinforcement learning. *Computational Intelligence*, 30(4):657–699, 2014.

[22] Manfred Eppe, Christian Gumbsch, Matthias Kerzel, Phuong DH Nguyen, Martin V Butz, and Stefan Wermter. Intelligent problem-solving as integrated hierarchical reinforcement learning. *Nature Machine Intelligence*, 4(1):11–20, 2022.

[23] Thomas M Scanlon. Thickness and theory. *The Journal of Philosophy*, 100(6):275–287, 2003.

[24] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pages 8583–8592. PMLR, 2020.

[25] Russell Mendonca, Oleh Rybkin, Kostas Daniilidis, Danijar Hafner, and Deepak Pathak. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391, 2021.

[26] Noor Sajid, Panagiotis Tigas, Alexey Zakharov, Zafeirios Fountas, and Karl Friston. Exploration and preference satisfaction trade-off in reward-free learning. In *ICML 2021 Workshop on Unsupervised Reinforcement Learning*, 2021.

[27] Christian Gumbsch, Martin V Butz, and Georg Martius. Sparsely changing latent states for prediction and planning in partially observable domains. *Advances in Neural Information Processing Systems*, 34:17518–17531, 2021.

[28] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. URL https://arxiv.org/abs/1412.3555.

[29] Marin Vlastelica, Sebastian Blaes, Cristina Pinneri, and Georg Martius. Risk-averse zero-order trajectory optimization. In *5th Annual Conference on Robot Learning*, 2021.

[30] Cansu Sancaktar, Sebastian Blaes, and Georg Martius. Curious exploration via structured world models yields zero-shot object manipulation. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., December 2022.

[31] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[32] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 07 2017.

[33] Reuven Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1:127–190, 1999.

[34] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[35] Danijar Hafner, Kuang-Huei Lee, Ian Fischer, and Pieter Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 35, 2022.

[36] Mikayel Samvelyan, Robert Kirk, Vitaly Kurin, Jack Parker-Holder, Minqi Jiang, Eric Hambro, Fabio Petroni, Heinrich Kuttler, Edward Grefenstette, and Tim Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research. In *Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[37] Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

[38] Vitchyr Pong, Murtaza Dalal, Steven Lin, and Ashvin V Nair. Multiworld: Multitask environments for rl, 2018. URL https://github.com/vitchyr/multiworld.

[39] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31, 2018.

[40] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: state-covering self-supervised reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7783–7792, 2020.

[41] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[42] Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021.

[43] Arnav Kumar Jain, Shivakanth Sujit, Shruti Joshi, Vincent Michalski, Danijar Hafner, and Samira Ebrahimi Kahou. Learning robust dynamics through variational sparse gating. *Advances in Neural Information Processing Systems*, 35:1612–1626, 2022.

[44] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A Clockwork RNN. In Eric P. Xing and Tony Jebara, editors, *International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 1863–1871, Bejing, China, 22–24 Jun 2014. PMLR. URL http://proceedings.mlr.press/v32/koutnik14.html.

[45] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances In Neural Information Processing Systems*, pages 3882–3890, 2016.

[46] Christian Gumbsch, Maurits Adam, Birgit Elsner, Georg Martius, and Martin V Butz. Developing hierarchical anticipations via neural network-based event segmentation. In *2022 IEEE International Conference on Development and Learning (ICDL)*, pages 1–8. IEEE, 2022.

[47] Alexander Neitz, Giambattista Parascandolo, Stefan Bauer, and Bernhard Schölkopf. Adaptive skip intervals: Temporal abstraction for recurrent dynamical models. *Advances in Neural Information Processing Systems*, 31, 2018.

[48] Dinesh Jayaraman, Frederik Ebert, Alexei Efros, and Sergey Levine. Time-agnostic prediction: Predicting predictable video frames. In *International Conference on Learning Representations*, 2019.

[49] Alexey Zakharov, Qinghai Guo, and Zafeirios Fountas. Variational predictive routing with nested subjective timescales. *arXiv preprint arXiv:2110.11236*, 2021.

[50] Vaibhav Saxena, Jimmy Ba, and Danijar Hafner. Clockwork variational autoencoders. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 29246–29257. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/f490d0af974fedf90cb0f1edce8e3dd5-Paper.pdf.

[51] Christian Gumbsch, Martin V. Butz, and Georg Martius. Autonomous identification and goal-directed invocation of event-predictive behavioral primitives. *IEEE Transactions on Cognitive and Developmental Systems*, 13(2):298–311, June 2019. doi: 10.1109/TCDS.2019.2925890. URL https://ieeexplore.ieee.org/document/8753716.

[52] Taesup Kim, Sungjin Ahn, and Yoshua Bengio. Variational temporal abstraction. *Advances in Neural Information Processing Systems*, 32, 2019.

[53] Alexey Zakharov, Qinghai Guo, and Zafeirios Fountas. Long-horizon video prediction using a dynamic latent hierarchy. *arXiv preprint arXiv:2212.14376*, 2022.

[54] Vihang Prakash Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M Blies, Johannes Brandstetter, Jose Arjona-Medina, and Sepp Hochreiter. Align-{rudder}: Learning from few demonstrations by reward redistribution. In *International Conference on Learning Representations*, 2021.

[55] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[56] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549, 2017.

[57] Nico Gürtler, Dieter Büchler, and Georg Martius. Hierarchical reinforcement learning with timed subgoals. *Advances in Neural Information Processing Systems*, 34:21732–21743, 2021.

[58] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[59] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In *Proceedings of International Conference on Learning Representations*, 2019.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[61] Chang Chen, Yi-Fu Wu, Jaesik Yoon, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models. *arXiv preprint arXiv:2202.09481*, 2022.

[62] Jan Robine, Marc Höftmann, Tobias Uelwer, and Stefan Harmeling. Transformer-based world models are happy with 100k interactions. In *The Eleventh International Conference on Learning Representations*, 2023.

[63] Vincent Micheli, Eloi Alonso, and François Fleuret. Transformers are sample-efficient world models. In *The Eleventh International Conference on Learning Representations*, 2023.

[64] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Supplementary Material for: Learning Hierarchical World Models with Adaptive Temporal Abstractions from Discrete Latent Dynamics

Contents

More supplementary material will be available on our website.

A Pseudocode

Algorithm 1 outlines how THICK world models make temporal abstract predictions using both levels of the hierarchy (also visualized in Fig. 4). Blue parts are only needed for MBRL or MPC (see Sec. 2.3). For temporal abstract rollouts, which are used in THICK PlaNet, the process can be repeated K times by using the output states, i.e. $\boldsymbol{c}_{\tau(t)}$ and $\hat{\boldsymbol{z}}^c_{\tau(t)}$, as inputs again.

Algorithm 1 THICK Temporal Abstract Prediction

```
 1: input: context $\boldsymbol{c}_t$, stochastic state $\boldsymbol{z}_t$
 2: $\hat{\boldsymbol{A}}_t \sim P_\theta(\hat{\boldsymbol{A}}_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t)$    ▷ sample high-level action
 3: $\hat{\boldsymbol{z}}_{\tau(t)-1} \sim F_\theta\big(\hat{\boldsymbol{z}}_{\tau(t)-1} \mid \hat{\boldsymbol{A}}_t \boldsymbol{c}_t, \boldsymbol{z}_t\big)$    ▷ high-level state prediction
 4: $\hat{\boldsymbol{a}}_{\tau(t)-1} \sim F_\theta\big(\hat{\boldsymbol{a}}_{\tau(t)-1} \mid \hat{\boldsymbol{A}}_t \boldsymbol{c}_t, \boldsymbol{z}_t\big)$    ▷ high-level action prediction
 5: $\Delta\hat{\tau}(t) \sim F_\theta\big(\Delta\hat{\tau}(t) \mid \hat{\boldsymbol{A}}_t \boldsymbol{c}_t, \boldsymbol{z}_t\big)$    ▷ high-level time prediction
 6: $(\hat{r}^\gamma_{t:\tau(t)} \sim F_\theta\big(\hat{r}^\gamma_{t:\tau(t)} \mid \hat{\boldsymbol{A}}_t \boldsymbol{c}_t, \boldsymbol{z}_t\big)$    ▷ high-level reward prediction
 7: $\boldsymbol{c}_{\tau(t)} \leftarrow g_\phi\big(\hat{\boldsymbol{a}}_{\tau(t)-1}, \boldsymbol{c}_t, \hat{\boldsymbol{z}}_{\tau(t)-1}\big)$    ▷ low-level context
 8: $\hat{\boldsymbol{z}}^c_{\tau(t)} \sim p^c_\phi\Big(\hat{\boldsymbol{z}}^{\boldsymbol{c}}_{\tau(t)-1} \mid \hat{\boldsymbol{a}}_{\tau(t)-1}, \boldsymbol{c}_{\tau(t)}, \hat{\boldsymbol{z}}_{\tau(t)-1}\Big)$    ▷ low-level coarse prior
 9: $\hat{r}^c_{\tau(t)}, \hat{\gamma}^c_{\tau(t)} \sim o^c_\phi\Big(\hat{r}^c_{\tau(t)}, \hat{\gamma}^c_{\tau(t)} \mid \boldsymbol{c}_{\tau(t)}, \hat{\boldsymbol{z}}^c_{\tau(t)}\Big)$    ▷ coarse reward & discount prediction
10: output: $\boldsymbol{c}_{\tau(t)}, \hat{\boldsymbol{z}}^c_{\tau(t)}, \widehat{\Delta t}, \hat{r}^\gamma_{t_\tau}$ $\hat{r}^c_{\tau(t)}, \hat{\gamma}^c_{\tau(t)}$
```

Algorithm 2 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 2 THICK Training Data Generation

```
 1: input: discount factor $\gamma$, sequences of contexts $\boldsymbol{c}_{1:T}$, stochastic states $\boldsymbol{z}_{1:T}$, actions $\boldsymbol{a}_{1:T}$,
 2:         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 $\boldsymbol{c}_\tau \neq \boldsymbol{c}_{\tau-1}$ or $d_\tau = 1$ then    ▷ context change or episode is over at time $\tau$
 6:         for $(\boldsymbol{c}_t, \boldsymbol{z}_t) \in \mathcal{I}$ do
 7:             compute passed time $\Delta\tau \leftarrow \tau - t$ and accumulated rewards $r_{t:\tau} \leftarrow \sum_{\delta=1}^{\Delta t-1} \gamma^\delta r_{t+\delta}$
 8:             add input-target tuple $\big((\boldsymbol{c}_t, \boldsymbol{z}_t), (\boldsymbol{z}_{\tau-1}, \boldsymbol{a}_{\tau-1}, \Delta t, r_{t:\tau})\big)$ to $\mathcal{D}$
 9:             remove $(\boldsymbol{c}_t, \boldsymbol{z}_t)$ from $\mathcal{I}$
10:     add potential input $(\boldsymbol{c}_\tau, \boldsymbol{z}_\tau)$ to $\mathcal{I}$
11: output: train data $\mathcal{D}$
```

Algorithm 3 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.

Algorithm 3 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 $\boldsymbol{s}_t \sim w_\phi(\boldsymbol{s}_t \mid \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$
5:      // Behavior
6:      if $\boldsymbol{c}_t \neq \boldsymbol{c}_{t-1} \wedge t \geq t^{\text{plan}} + I$ then
7:          plan subgoal $\boldsymbol{z}_t^{\text{goal}}$ using MCTS and temporal abstract rollouts (Alg. 1)
8:          $t^{\text{plan}} \leftarrow t$
9:      plan new action $\boldsymbol{a}_t$ using CEM given $\boldsymbol{s}_t$ and $\boldsymbol{z}_t^{\text{goal}}$ (Eq. 23)
10:     sample new action $\boldsymbol{a}_t$ from actor $\pi$ given $\boldsymbol{s}_t$
11:     execute action $\boldsymbol{a}_t$ in environment and observe $\boldsymbol{r}_t$, $\boldsymbol{i}_t$ and $d_t$
12:     add $(\boldsymbol{i}_t, \boldsymbol{a}_t, r_t, d_t)$ to replay buffer
13:     // Train world models
14:     draw sequence batch $\mathcal{B} \leftarrow (\boldsymbol{i}_{t':T}, \boldsymbol{a}_{t':T}, r_{t':T}, d_{t':T})$ from replay buffer
15:     embed batch in latent state $\boldsymbol{s}_{t':T} \sim w_\phi(\boldsymbol{s}_{t':T} \mid \mathcal{B})$
16:     update low-level world model $w_\phi$ using $\mathcal{B}$ (Eq. 9)
17:     generate high-level training batch $\mathcal{D}$ from $(\boldsymbol{s}_{t':T}, \boldsymbol{a}_{t':T}, r_{t':T}, d_{t':T})$ (Alg. 2)
18:     update high-level world model $W_\theta$ using $\mathcal{D}$ (Eq. 18)
19:     // Train actor and critic
20:     imagine trajectory $(\boldsymbol{s}_{t'':H}, \boldsymbol{a}_{t'':H}, r_{t'':H}, \gamma_{t'':H})$ using $w_\phi$ from random start $s_{t''} \in \mathcal{B}$
21:     make temporal abstract predictions for each $\boldsymbol{s}_{t'':H}$ using $W_\theta$ and $w_\phi$ (Alg. 1)
22:     compute value $V$ (Eq. 20)
23:     update critics $v_\chi$ and $v_\xi$ (Eq. 21)
24:     update actor $\pi$
```

B Hyperparameters

Name		**Value**	
	MH	VPP	MW
Low-Level World Model (or RSSM)			
Batches (size $\times$ sequence length)		16×50	
Dimensions of $\boldsymbol{c}_t$		16	
Dimensions of $\boldsymbol{h}_t$		256	
Dimensions of $\boldsymbol{z}_t$		32×32	
MLP features per layer		256	
Sparsity loss scale β^{sparse}	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 $\boldsymbol{A}_t$	3	5	5
Use terminations d_t for segmentation	yes	no	no
Loss for training $F_\theta^{\hat{\boldsymbol{a}}}\big(\hat{\boldsymbol{a}}_{\tau(t)-1} \,\vert\, \boldsymbol{A}_t, \boldsymbol{c}_t, \boldsymbol{z}_t\big)$	CCE	CCE	NLL
Action prediction loss scale $\alpha^{\mathrm{a}_{\tau(\mathrm{t})-1}}$	1	1	0.1
State prediction loss scale $\alpha^{\mathrm{z}_{\tau(\mathrm{t})-1}}$		1	
Time prediction loss scale $\alpha^{\Delta\tau(\mathrm{t})}$	1	1	0.1
Reward prediction loss scale $\alpha^{\mathrm{r}^{\gamma}_{\mathrm{t}:\tau(\mathrm{t})}}$		1	
KL balancing α^{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	$4\times$ 400	$4\times$ 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	

Table 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 Mulitworld (MW).

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 [16] – except for minor variations. Specifically, we decreased the model size by setting the feature size of the RSSM and the dimensionality of $\boldsymbol{h}_t$ to 256. 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 [35], 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 [15]. 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 [14]. In some tasks, PlaNet repeats the same action multiple times in order to tackle long horizon planning. To give PlaNet a fair chance for the long-horizon tasks `Door` and `PickUp`, we use an action repeat $R = 2$ in those tasks for all models. For high-level planning with MCTS, we use MuZero's [34] implementation, with the same hyperparameters if not specified otherwise.

Differences between environments The main difference between MiniHack and the other environments is that in MiniHack episodes can terminate based on task success or 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 the termination of episodes in the replay buffer. Importantly, we do not treat episode terminations as context changes. For action prediction, we use 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`; 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. F.4). For simplicity and to demonstrate robustness, we used the same values for each suite.

How to tune When tuning THICK world models for a new task, we recommend mainly searching over the sparsity loss scale $\beta^{\text{sparse}} \in \{0.1, 1, 5, 10, 25, 50\}$. Typically, one random seed is sufficient to determine which β^{sparse} leads to few, but not too few, context changes. Depending on horizon length and reward sparsity of the task, searching for $\psi \in \{0.8, 0.9\}$ and $\kappa \in \{0.025, 0.1, 0.5\}$ can also boost performance.

C Extended Related Work

Sparsity in RNNs: Developing hierarchical RNNs based on sparse activity was already proposed in the 90s by Jürgen Schmidhuber [7]. The Neural History Compressor [7] uses a hierarchical stack of RNNs, that autoregressively predict the next inputs. The higher levels in the hierarchy remain inactive until the lower level fails to predict the next input. Recently, there has been increasing interest in regularizing RNNs towards sparse latent updates. Alternative approaches to the L_0-regularization of GateL0RD [27] are using sparse attention masks [41, 42], competition among submodules [42], regularizing update gates towards a variational prior [43], or time-dependent updates [44, 45].

Temporal abstractions from regularized latent dynamics: Previously, sparse changes in the latent states of a low-level model have been used to model temporal abstractions [46, 50]. In contrast to our work, the temporal abstractions in Gumbsch et al. [46] were learned in much simpler settings with highly structured observations, instead of the high-dimensional, pixel-based observations examined in this work. Additionally, these temporal abstractions were only used to model goal-anticipatory gaze behavior of infants and have not been applied for MPC or MBRL. Separately, Saxena et al. [50] 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 apt at learning slow-moving content at higher levels of the temporal hierarchy, unlike C-RSSM and THICK, it requires the temporal abstraction factor to be defined explicitly.

Temporal abstractions from predictability: Adaptive Skip Intervals (ASI) [47] 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 an extended sequences into the future. As a result, the model learns to skip a number of inputs towards predictable transitions. Along similar lines, Temporal-Agnostic Predictions (TAP) [48] identifies 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. [49] provide a learning-free mechanism for detecting 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. This is unlike THICK, where context changes are determined using a learning-based sparsity regularization. An opposing approach is using unexpected prediction errors of a forward model for self-supervised time series segmentation [51]. Here, the idea is that in certain states the dynamics of the agent-environment interactions change, e.g. changing the terrain during locomotion, which lead to a temporary increase in the prediction error.

Temporal abstractions from learning boundary detectors: Besides using indirect measure to segment a sequence, a straight-forward approach is to train a boundary detector that signals the boundary of subsequences [52, 53]. Kim et al. [52] train a boundary detectors that is regularized by specifying the maximum number of subsequences allowed and their maximal 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. Conversely, Zakharov et al. [53] 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.

D THICK World Models: Implementation Details

D.1 GateL0RD

We want the context code $\boldsymbol{c}_t$ to only change sparsely in time. Thus, we implement the discrete context dynamics g_ϕ as a GateL0RD cell [27]. GateL0RD is an RNN designed to maintain sparsely changing latent states $\boldsymbol{c}_t$. In order to realize this inductive bias, GateL0RD uses two subnetworks g^p_ϕ and g^g_ϕ that control $\boldsymbol{c}_t$-updates via an internal update gate $\boldsymbol{\Lambda}_t$. GateL0RD can be summarized as follows:

$$\begin{aligned}
\text{Candidate proposal:} \quad & \hat{\boldsymbol{c}}_t = g^p_\phi(\boldsymbol{a}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{z}_{t-1}) && (25)\\
\text{Update gate:} \quad & \boldsymbol{\Lambda}_t = g^g_\phi(\boldsymbol{a}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{z}_{t-1}) && (26)\\
\text{Context Update:} \quad & \boldsymbol{c}_t = \boldsymbol{\Lambda}_t \circ \hat{\boldsymbol{c}}_t + (1 - \boldsymbol{\Lambda}_t) \circ \boldsymbol{c}_{t-1} && (27)
\end{aligned}$$

with $\circ$ denoting the Hadamard product. We use the action $\boldsymbol{a}_{t-1}$ and the last stochastic state $\boldsymbol{z}_{t-1}$ as the cell inputs. Based on this cell input and the last context $\boldsymbol{c}_{t-1}$, GateL0RD proposes a new context $\hat{\boldsymbol{c}}_t$ via its proposal subnetwork g^p_ϕ (Eq. 25). Whether the context is updated depends on an update gate $\boldsymbol{\Lambda}_t \in [0,1]^m$ (Eq. 27). This update gate $\boldsymbol{\Lambda}_t$ is the output of the gating subnetwork g^g_ϕ (Eq. 26) which uses a rectified $\tanh$ activation function (ReTanh), with $\mathrm{ReTanh}(x) := \max(0, \tanh(x))$. This ensures that the gate activations are $\in [0,1]^m$. Note that to compute $\boldsymbol{\Lambda}_t$, the subnetwork g^g_ϕ internally samples from a Gaussian distribution before applying the ReTanh function. This was shown to improve robustness [27]. Thus, the context updates are a stochastic process.

Originally [27], GateL0RD used a subnetwork to compute the cell output using multiplicative gating. We omit this here and instead feed to output to the GRU cell f_ϕ as shown in Fig. 2 (right).

The centralized gate $\boldsymbol{\Lambda}_t$ of GateL0RD makes it easy to determine context changes, i.e. $\boldsymbol{c}_t \neq \boldsymbol{c}_{t-1}$. Since all context updates depend on $\boldsymbol{\Lambda}_t$, we know that the context changed if $\boldsymbol{\Lambda}_t > \boldsymbol{0}$. 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.

D.2 C-RSSM Loss

The loss of the C-RSSM (Eq. 9) is composed of three parts: the prediction loss $\mathcal{L}^{\text{pred}}$, the KL loss $\mathcal{L}^{\text{KL}}$, and the sparsity loss $\mathcal{L}^{\text{sparse}}$. Expect 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

$$\mathcal{L}^{\text{pred}}(\phi) = \sum_{t=1}^{T} \Big[\sum_{y \in \{\boldsymbol{i}_t, \boldsymbol{r}_t, \gamma_t\}} - \log o_\phi(y \mid \boldsymbol{s}_t) - \log o_\phi^c(y \mid \boldsymbol{c}_t, \boldsymbol{z}_t) \Big]. \tag{28}$$

Equations in red are exclusive to the C-RSSM. Thus, the network is trained to minimize the negative log likelihood for predicting the images $\boldsymbol{i}_t$, rewards r_t and future discounts γ_t. Here we account for both the precise predictions over the output heads o_ϕ (Eq. 7), as well as for the coarse predictions over the output heads o_ϕ^c (Eq. 8). Following the codebase of DreamerV2 [16], 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 $\hat{\boldsymbol{z}}_t$: fine predictions using the full state (Eq. 6) and coarse predictions based only on the context, last action and stochastic state (Eq. 5). We need to account for both types of prediction in the KL loss $\mathcal{L}^{\text{KL}}$ with

$$\mathcal{L}^{\text{KL}}(\phi) = \sum_{t=1}^{T} \text{KL}\Big[q_\phi(\boldsymbol{z}_t \mid \boldsymbol{h}_t, \boldsymbol{i}_t) || p_\phi^h(\hat{\boldsymbol{z}}_t \mid \boldsymbol{h}_t)\Big] + \text{KL}\Big[q_\phi(\boldsymbol{z}_t \mid \boldsymbol{h}_t, \boldsymbol{i}_t) || p_\phi^c(\hat{\boldsymbol{z}}_t^{\boldsymbol{c}} \mid \boldsymbol{a}_{t-1}, \boldsymbol{c}_t, \boldsymbol{z}_{t-1})\Big]. \tag{29}$$

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 [16], we use KL-balancing, which scales the prior p_ϕ of each KL divergence by a factor $\beta^{\text{bal}} = 0.8$, and 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 GateL0RD [27] which is an L_0-regularization of the context changes $\boldsymbol{\Delta c}_t$. This is implemented as

$$\mathcal{L}^{\text{sparse}}(\phi) = \sum_{t=1}^{T} ||\boldsymbol{\Delta c}_t||_0 = \sum_{t=1}^{T} \Theta(\boldsymbol{\Lambda}_t) \tag{30}$$

where $\Theta(\cdot)$ denotes the Heaviside step function. That is, an L_0-regularization of the context changes is implemented as the binarization of the update gates $\boldsymbol{\Lambda}_t$ (Eq. 27). We estimate the gradient of the Heaviside step function using the straight-through estimator [64]. The advantage of GateL0RD's L_0-regularization to other regularization towards sparsity, such as using low variational prior [43, 52], is that in fully-observable and highly predictable situations, GateL0RD will shut its gates and not change the context almost regardless of the order of magnitude of β^{sparse} to avoid a punishment.

D.3 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 KL loss $\mathfrak{L}^{\text{KL}}$ between the high-level action distributions. We define the prediction loss $\mathfrak{L}^{\text{pred}}$ as the summed negative log likelihood (NLL) of the to be predicted action $\boldsymbol{a}_{\tau(t)-1}$, stochastic state $\boldsymbol{z}_{\tau(t)-1}$, passed time $\Delta\tau(t)$, and rewards $r^{\gamma}_{t:\tau(t)}$, i.e.

$$\mathfrak{L}^{\text{pred}}(\theta) = \sum_{t=1}^{T} \Big[\sum_{Y \in \{\boldsymbol{a}_{\tau(t)-1}, \boldsymbol{z}_{\tau(t)-1}, \Delta\tau(t), r^{\gamma}_{t:\tau(t)}\}} \alpha^Y \; - \log F_\theta(Y \mid \boldsymbol{A}_t, \boldsymbol{c}_t, \boldsymbol{z}_t) \Big]. \tag{31}$$

The hyperparameters $\alpha^Y \in \{\alpha^{\mathrm{a}_{\tau(t)-1}}, \alpha^{\mathrm{z}_{\tau(t)-1}}, \alpha^{\Delta\tau(t)}, \alpha^{\mathrm{r}^{\gamma}_{t:\tau(t)}}\}$ can be used to scale the individual prediction losses. As default, we set $\alpha^Y = 1$ for all loss terms. When training the network on task-free exploration, i.e. during zero-shot MPC as described in Sec. 3.3, we found that predicting the actions $\boldsymbol{a}_{\tau(t)-1}$ at context changes and elapsed time $\Delta\tau(t)$ was challenging. To mitigate this, during task-free exploration we set $\alpha^{\mathrm{a}_{\tau(t)-1}} = 0.1$ and $\alpha^{\Delta\tau(\mathrm{t})} = 0.1$. For predicting continuous actions we sample from a Gaussian distribution of predicted actions and compute the NLL as the loss for action prediction. For discrete actions we predict a Categorical distribution from which we sample the actions, and compute the Categorical Cross Entropy (CCE) loss.

The KL loss $\mathfrak{L}^{\mathrm{KL}}$ drives the system to minimize the divergence between the posterior high-level action distribution $Q_\theta(\boldsymbol{A}_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t, \boldsymbol{c}_{\tau(t)}, \boldsymbol{z}_{\tau(t)})$, and the prior distribution $P_\theta(\hat{\boldsymbol{A}}_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t)$ with

$$\mathfrak{L}^{\mathrm{KL}} = \sum_{t=1}^{T} \mathrm{KL}\Big[Q_\theta(\boldsymbol{A}_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t, \boldsymbol{c}_{\tau(t)}, \boldsymbol{z}_{\tau(t)}) \quad || \quad P_\theta(\hat{\boldsymbol{A}}_t \mid \boldsymbol{c}_t, \boldsymbol{z}_t)\Big]. \tag{32}$$

Like the KL loss $\mathcal{L}^{\mathrm{KL}}$ on the low level (see Suppl. D.2), we use KL balancing [16] to scale the prior part by $\alpha^{\mathrm{bal}} = 0.8$ and the posterior part by $1 - \alpha^{\mathrm{bal}}$.

D.4 THICK Dreamer: Details

THICK Dreamer estimates the overall value $V(\boldsymbol{s}_t)$ of a state $\boldsymbol{s}_t$ as a mixture of short- and long-horizon estimates (Eq. 20). The short-horizon value estimate is computed as the general λ-target as in DreamerV2 with

$$V^\lambda(\boldsymbol{s}_t) = \hat{r}_t + \hat{\gamma}_t \begin{cases} (1-\lambda)v_\xi(\hat{\boldsymbol{s}}_{t+1}) + \lambda V^\lambda(\hat{\boldsymbol{s}}_{t+1}) & \text{for } t < H, \\ v_\xi(\hat{\boldsymbol{s}}_{t+1}) & \text{for } t = H \end{cases} \tag{33}$$

where $\hat{r}_t$ and $\hat{\gamma}_t$ are sampled from the output heads o_ϕ given $\boldsymbol{s}_t$ (Eq. 7) and λ is a hyperparameter.

The functions V^λ and V^{long} compute value targets using the critics v_ξ and v_χ, respectively. Like DreamerV2, we stabilize critic training by using a copy of the critics during values estimation (in Eq. 33 and Eq. 19). The copy is updated after every 100 updates.

D.5 THICK PlaNet: Details

For planning on the high level we use a MCTS implementation based on MuZero. However, intuitively we wouldn't expect multiple predictions to reach a goal, we decrease the number of simulations to $S = 100$.

We only replan on the high level if the context changes, i.e. $\boldsymbol{c}_t \neq \boldsymbol{c}_{t-1}$. Since all the subgoals $\boldsymbol{z}_t^{\mathrm{goal}}$ 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. Too avoid high computational load from replanning and to enable 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 Environment Details

E.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 agents 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 two easiest tasks (`WaterCrossing-Ring`, `KeyRoom-Fixed-S5`) 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 each level.

`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. 9a). When a ring is picked up in our tasks, it is automatically worn[6]. The level is inspired by `LavaCross-Levitate-Ring-PickUp`

[6]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.

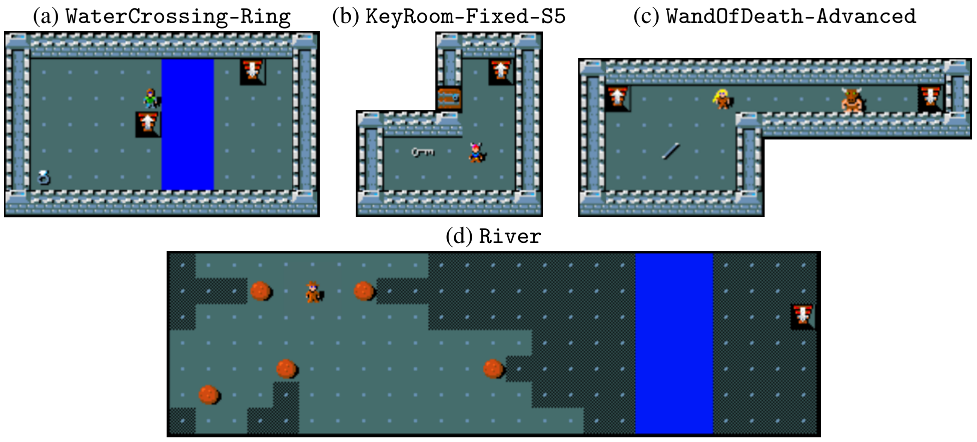


Figure 9: **MiniHack environments**. Staircases with an upwards facing arrows mark the starting point of the agents. Staircases with downward facing arrows are the exits that need to be reached. In (a), (b), and (d) start points and exits are randomized.

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. 9b). 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.

`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 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. 9c) is a self-created level layout, designed to be more challenging that `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. 9d). 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. 13a). Via `EscapeRoom` we test the ability to learn to execute a complex event sequence. Nonetheless, 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 get the key, the agent needs to first pick up a ring of levitation and float over a small patch of water into a corridor. In the corridor the agent can exchange the ring of levitation for a key. In order to get back to the door in the first room, the gent needs to push a boulder into water. While levitating, the agent is too light to push the boulder. In `EscapeRoom`, the agent can only carry one new item and picking up a second item results in dropping the first one.

E.2 Multiworld

All tasks in Multiworld use different action spaces and camera viewpoints for their pixel-based observation, shown in Fig. 10. In `Pusher` the 2-dimensional actions control the $x-$ and $y-$movement of the endeffector, whereas the gripper is fixed. In `Door` the robot has a hook instead of a gripper at its endeffector 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

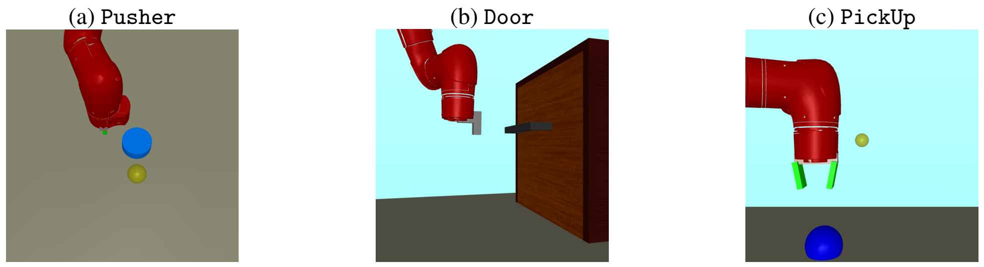


Figure 10: **Multiworld environments**. Goal positions for the objects are shown in yellow.

Pusher and PickUp they are visible in the video frames. In Door the goal is to open the door fully. For Pusher-Dense and PickUp we compute the reward r_t for every time step t as

$$r_t = 1 - \frac{\delta_t}{\delta_1}, \tag{34}$$

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 euclidean 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.

F Extended Results and Experiment Details

F.1 Analysis of Contexts and Predictions

In this section we provide further examples of high- and low-level predictions and context codes $\boldsymbol{c}_t$. Figure 11 visualizes the low-level predictions for two example sequences. The low-level world model predicts the immediate next state and the reconstructions are more accurate than the abstract high-level predictions (cf. Fig. 5). Figure 12 displays four example sequences with the corresponding contexts $\boldsymbol{c}_t$ and high-level predictions.

We analyze the high-level actions $\boldsymbol{A}_t$ in more detail for the EscapeRoom problem. EscapeRoom is a challenging MiniHack level, designed to contain diverse agent-environment interactions, shown in Fig. 13a and described in detail in Suppl. E.1. To illustrate the emerging high-level action representations of THICK Dreamer, we show inputs $\boldsymbol{i}_t$ and image reconstructions of high-level predictions for all high-level actions $\boldsymbol{A}_t$ in Fig. 13b for one exemplary sequence.

At specific time steps the three possible high-level actions $\boldsymbol{A}_t$ encode particular agent-environment interactions: $\boldsymbol{A}^1_t$ encodes picking up the ring of levitation ($t = 3$) or exiting the level ($t \in \{22, 26\}$). $\boldsymbol{A}^2_t$ encodes crossing the water after obtaining the ability to levitate ($t \in \{6, 10, 16\}$). $\boldsymbol{A}^3_t$ encodes pushing the boulder into water ($t \in \{6, 10, 16\}$) or opening the door ($t = 22$). For all other time steps, the high-level actions produce either the identity predictions (e.g. A^1_6) or predictions that seem to encode average scene settings (cf. A^2_3 or A^1_{10}). These predictions account for unexpected context shifts, which can always occur with a small chance due to the stochasticity from sampling $\boldsymbol{z}_t$ and the stochastic update gates of GateL0RD (see Suppl. D.1). The prior Q_ϕ (red frames in Fig. 13b) mostly

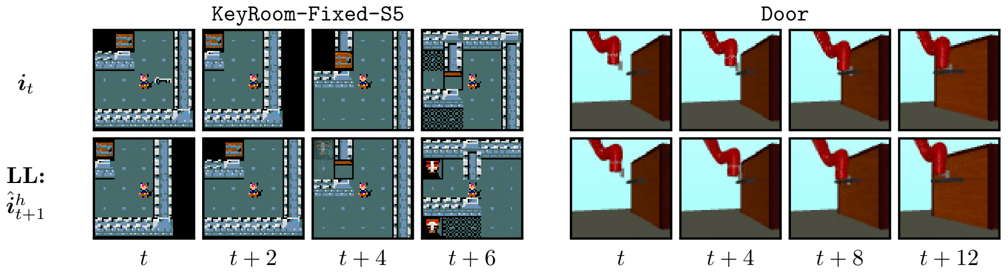


Figure 11: **Low-level predictions**. The low-level predictions accurately predict the next frames (cf. Fig. 5 for high-level predictions and contexts $\boldsymbol{c}_t$ of the same sequences).

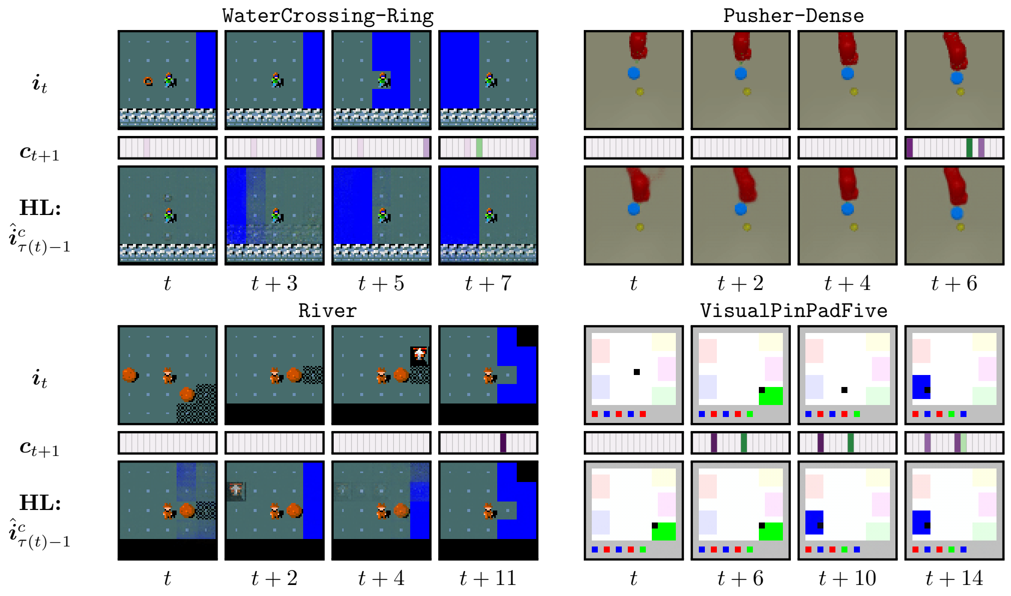


Figure 12: **Context changes and high-level predictions**. We show the input images $\boldsymbol{i}_t$, 16-dim. contexts $\boldsymbol{c}_{t+1}$ and reconstructions $\hat{\boldsymbol{i}}^c_{\tau(t)-1}$ of the high-level predictions. For `WaterCrossing-Ring` the context changes when stepping on the ring, 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.

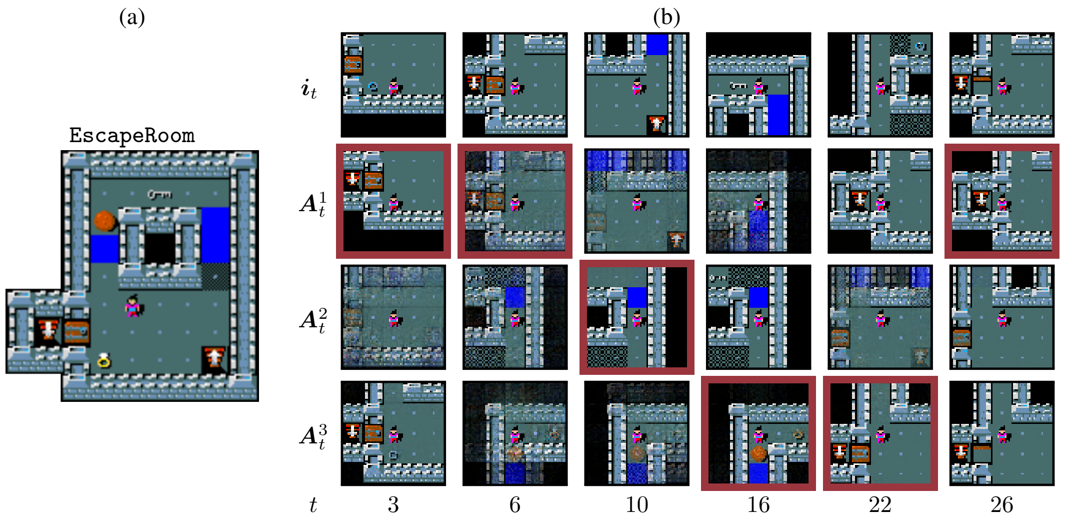


Figure 13: **High-level action predictions.** (a) In the `EscapeRoom` problem an agent needs to pick up a ring of levitation, hover over a patch of water to get to a key, exchange the ring for key, push a boulder into the water, and use the key to unlock a door. (b) Visualization of the high-level actions for one exemplary sequence. The top row shows the input image $\boldsymbol{i}_t$. Image reconstructions $\hat{i}^c_{\tau(t)-1}$ are shown for the three high-level actions $\boldsymbol{A}_t$. Red outlines depict which action $\hat{\boldsymbol{A}}_t$ was sampled.

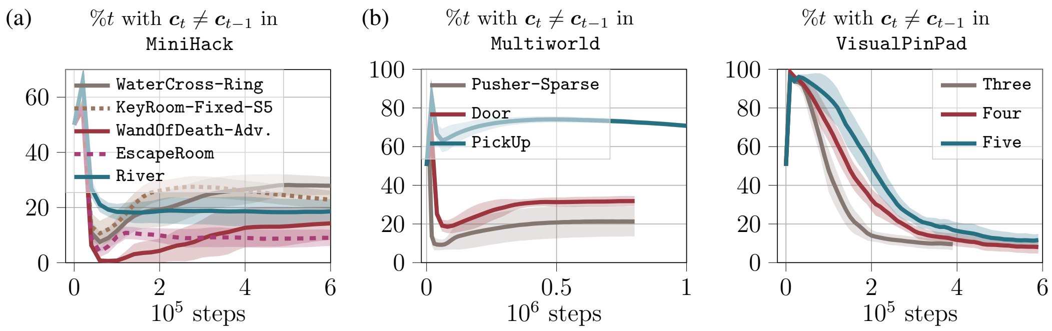


Figure 14: **Context changes over training**. Each graphic plots the mean percentage of time steps per training batch for which the context $\boldsymbol{c}_t$ changes in MiniHack (a) , VisualPinPad (b), and Multiworld (c). Shaded areas depict the standard deviation.

samples reasonable high-level actions. However, occasionally the prior samples an action $\hat{\boldsymbol{A}}_t$ leading to an identity or average prediction (e.g. $t = 6$) due to the randomness of the process.

To quantify the context changes, we plot the mean percentage of time steps when context changes occur (i.e., $\boldsymbol{c}_t \neq \boldsymbol{c}_{t-1}$) over the course of training in Fig. 14. Importantly, the context changes are somewhat consistent within the same task but, as expected, can vary across tasks of 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. 15 which illustrates that different tasks also show different distributions of context durations.

Lastly, we analyze whether context changes occur at task-relevant situations for some MiniHack problems. For this we generate rollouts using 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 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.

	`WaterCross-Ring`	`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)

Table 2: **Task-relevance of context changes**. We list the mean percentage of context changes $\boldsymbol{c}_{t^*} \neq \boldsymbol{c}_{t^*-1}$ for fully trained policies (7 seeds) at crucial task relevant points in time t^* in 10 sequences. See text for criterion of t^*. The standard deviation is denoted by $\pm$.

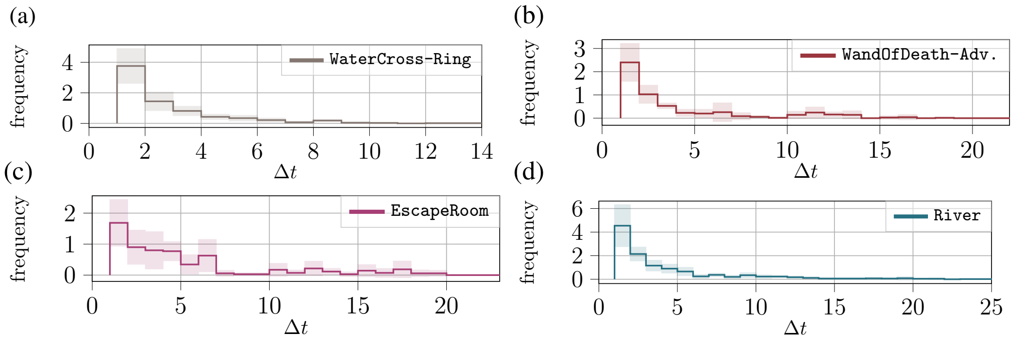


Figure 15: **Context duration during an episode**. Each graphic plots a histogram of the mean number of time steps Δt between two consecutive context changes during an episode (over 10 episodes, max 50 steps) for different MiniHack tasks (7 seeds). Shaded areas depict the standard deviation.

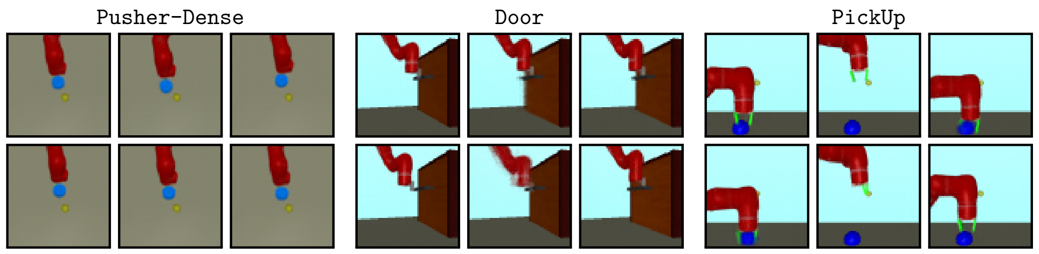


Figure 16: **Subgoals proposed in the first time step**. We reconstruct images based on the subgoals z_1^{goal} that THICK PlaNet set at the first time step. The subgoals are typically pushing the puck (`Pusher`), moving to the door handle (`Door`), or grasping the ball (`PickUp`). For `PickUp` the system sometimes fails to find a reasonable subgoal (center).

F.2 MBRL: Experiment details

For the Visual Pin Pad suite we generated offline training data to sidestep the challenge of discovering the very sparse rewards. For the data collection, we used Plan2Explore [24] with the default settings of the DreamerV2 [16] codebase. We trained two randomly initialized models of Plan2Explore for 1M environment steps in each task of the Visual Pin Pad suite. For each task, we determined the model that achieved the highest overall returns during training. We used the datasets collected by the explorative agents to initialize the replay buffers of all new models.

Originally, Visual Pin Pad has four levels of difficulty. However, in `VisualPinPadSix` Plan2Explore did not receive any reward during 1M steps of exploration. Besides that, the results in Hafner et al. [35] suggest that Dreamer is also not able to discover the very sparse rewards of `VisualPinPadSix` on its own. Thus, we omitted `VisualPinPadSix`.

F.3 MPC: Experiment details

To study zero-shot planning, we generated offline datasets for every task. For data collection, we use Plan2Explore in the same way as described in Suppl. F.2. After determining one dataset for every task, we train the models purely on this data.

Besides boosting performance for long-horizon tasks, THICK PlaNet 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 16 shows exemplary goals selected by the high-level planner in the first time step of an episode. Thus, the behavior of THICK PlaNet is much more explainable than simply performing MPC in a flat world model.

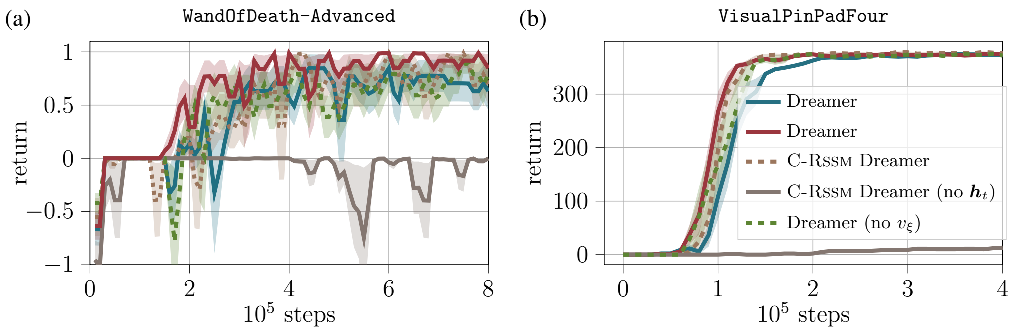


Figure 17: **C-RSSM and THICK Dreamer ablations**. Each graphic plots the mean returns over training steps. We compare Dreamer and THICK Dreamer (7 random seeds) against various ablations (5 seeds each): Dreamer using the C-RSSM (C-RSSM Dreamer), Dreamer using only the coarse processing pathway of the C-RSSM (C-RSSM Dreamer no $\boldsymbol{h}_t$), and THICK Dreamer using only one critic (THICK Dreamer no v_ξ). Shaded areas depict the standard error.

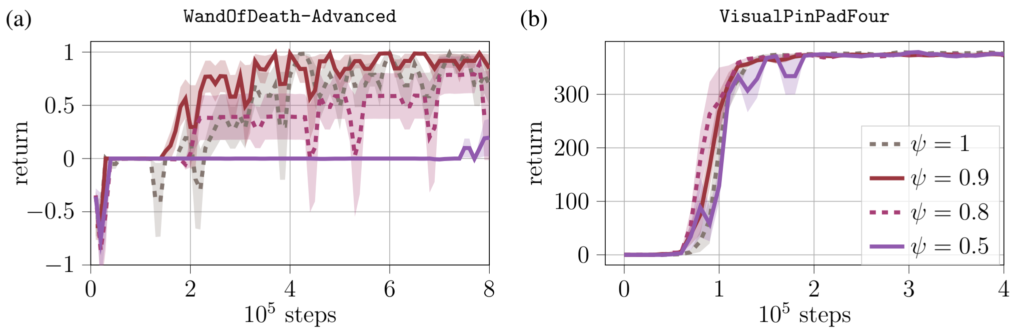


Figure 18: **Effect of hyperparameter ψ.** Each graphic plots the mean returns for THICK Dreamer over training steps for different values of the hyperparameter ψ (5 random seeds). Shaded areas depict the standard error.

F.4 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-Advances` and `VisualPinPadFour`. Figure 17 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 $\boldsymbol{h}_t$ and the precise processing pathway from the C-RSSM (i.e. C-RSSM Dreamer without $\boldsymbol{h}$) impedes the system from learning the tasks.[7] Omitting v_ξ, and only using one critic v_χ for both the short- and long-horizon returns (Eq. 20), slightly degrades the performance of THICK Dreamer.

Hyperparameter ψ THICK Dreamer introduces a new hyperparameter ψ which balances the influence of the short-horizon value estimates V^λ and long-horizon value estimates V^{long} on the overall value V (Eq. 20). Figure 18 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-Advances` 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. 23). We analyze the effect of κ on THICK PlaNet's performance in `Multiworld-Door`, shown in Fig. 19. 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.

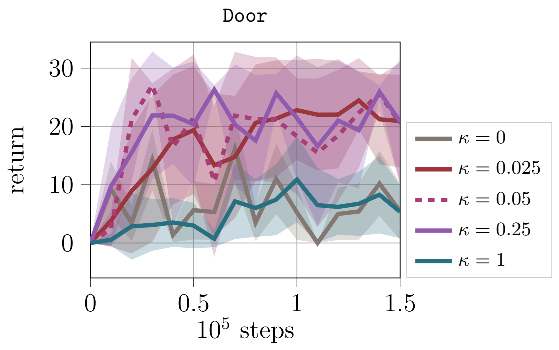


Figure 19: **Effect of hyperparameter κ.** We plot the mean returns of THICK PlaNet during zero-shot planning for different values of the hyperparameter κ (5 random seeds). Shaded areas depict the standard deviation.

Hyperparameter β^{sparse} Lastly, we compare the effect of sparsity loss scale β^{sparse} on THICK Dreamer in `VisualPinPadFour` and on THICK PlaNet in `Multiworld-Door`.

[7]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.

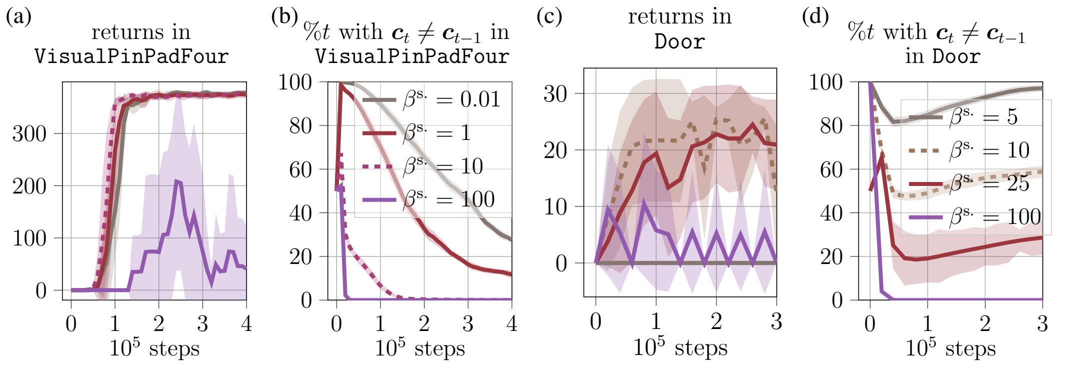


Figure 20: **Effect of sparsity**. We plot the the mean return for THICK Dreamer in `VisualPinPadFour` (a) and the mean zero-shot planning returns of THICK PlaNet in `Door` (c) for different values of the hyperparameter β^{sparse} (5 random seeds). Additionally, we plot the percentage of time steps with 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.

Figure 20a plots the mean returns of THICK Dreamer or different values for β^{sparse} in `VisualPinPadFour`. Figure 20b shows the percentage of time steps with context 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. 1). Stronger regularization boost 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 learning the task (e.g. $\beta^{\text{sparse}} = 100$). In this case, the high-level predictions are essentially average state predictions, which simply contributes noisy values for learning the critic. THICK Dreamer is very robust to the choice of β^{sparse} in `VisualPinPadFour`.

Figure 20c plots zero-shot planning performance of THICK PlaNet for different values for β^{sparse}, with the percentage of context changes shown in Fig. 20d. 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.

G Computation and Code

All experiments were run on an internal computing cluster. Each experiment used one GPU. Experiments using DreamerV2 took roughly 15-20 hours, whereas THICK Dreamer experiments took around 35-45 hours of wall clock time, depending on the overall number of environment steps. Zero-shot MPC experiments with PlaNet took at round 30-35 hours for 10^6 training steps, whereas THICK PlaNet took roughly 50-60 hours for the same number of training steps. The higher wall clock time for training THICK world models stems mainly from the larger number of trainable parameters and more detailed logging. Besides that, THICK PlaNet takes longer to evaluate, due to the additional computational cost of running MCTS on the high level during planning. The code to run all experiments will be open-sourced over our website.