AMORTIZED LOW-RANK ADAPTATION FOR MODEL-BASED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

World models let agents plan by predicting the consequences of their actions, but changes in the environment can make them inaccurate. We study the problem of adapting a world model to an unknown test-time environment, drawn from a known environment family, using only a few episodes of interaction. Existing approaches trade off computational cost against expressivity, i.e., the range of models a method can produce. For example, in-context learning is computationally cheap but limited in expressivity, and gradient-based adaptation is expressive but computationally expensive. We present CLAW (**C**ontext-conditioned **L**ow-rank **A**daptation of **W**orld models), which addresses this tradeoff by using a hypernetwork to generate low-rank (LoRA) adapters at test time. During pretraining, we simulate adaptation to a variety of environments and jointly train the hypernetwork and base world model. At test time, we freeze the base model and use a forward pass of the hypernetwork to generate adapters from a small batch of test-time transitions. We evaluate CLAW in locomotion and manipulation environment families that vary in dynamics, embodiment, and reward. We show that, using only seconds of test-time data, CLAW outperforms gradient-based adaptation and in-context learning during online adaptation. We also show that CLAW avoids overfitting in data-scarce regimes, that its advantage comes from the expressive adapters rather than context conditioning, and that pretraining the hypernetwork jointly with the base model outperforms training it post hoc.

1 INTRODUCTION

In reinforcement learning (RL), learned world models allow agents to plan by predicting the consequences of their actions (Ha & Schmidhuber, 2018; Hafner et al., 2023; Hansen et al., 2024). However, changes in the environment can make a world model inaccurate, and it is often desirable to adapt it without collecting large test-time datasets. In this paper, we study the problem of adapting a world model to an unknown test-time environment drawn from a known environment family (e.g., a set of locomotion environments with varying dynamics, embodiments, or rewards).

We cast this problem as a meta-reinforcement learning problem where we amortize adaptation during pretraining, enabling online adaptation with little data. Existing approaches face a tradeoff: they are either computationally cheap but limited in expressivity (e.g., in-context learning), or expressive but computationally expensive (e.g., gradient-based weight adaptation). We present CLAW (**C**ontext-conditioned **L**ow-rank **A**daptation of **W**orld models), an adaptation method which addresses this tradeoff. CLAW uses hypernetworks (Ha et al., 2017), neural networks trained to produce weights for other neural networks, to produce low-rank (LoRA) adapters (Hu et al., 2022) for the world model using a small batch of test-time transitions. This enables adaptation with the computational efficiency of in-context learning while retaining the expressivity of weight-space adaptation.

We validate CLAW with experiments in a variety of locomotion and manipulation environment families, where we vary dynamics, embodiments and rewards. We show that CLAW outperforms gradient-based and in-context learning baselines during online adaptation, that it avoids overfitting in data-scarce regimes, that its advantage comes from the expressivity of the generated adapters (rather than context conditioning), and that pretraining the hypernetwork jointly with the base model outperforms training it post hoc.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2 Related Work

2.1 Adaptation in Model-Based RL

There are three main approaches to adapt a world model to test-time environments. The first approach changes the model's weights directly by, e.g., gradient-based re-training (Levy et al., 2025; 2026; Xu et al., 2025; Pfrommer et al., 2020; Lanier et al., 2025; Kim et al., 2026; Wang et al., 2025); the same approach, applied to policies rather than world models, also appears in model-free RL (Ball et al., 2023; Kostrikov et al., 2022; Yin et al., 2025; Physical Intelligence et al., 2025). This approach is highly expressive (i.e., can produce a large range of models) at the cost of increased test-time compute and high sample complexity.

The second approach infers a learned embedding of the test-time environment (e.g., a latent task variable, a belief, or context embedding) and conditions a fixed model on it (Rakelly et al., 2019; Zintgraf et al., 2020; 2019; Duan et al., 2016; Mishra et al., 2018; James et al., 2018; Kumar et al., 2021; Yu et al., 2017; Zhang et al., 2025; Chen et al., 2021; Reed et al., 2022; Jiang et al., 2022; Laskin et al., 2023; Liu et al., 2025). This approach is computationally inexpensive (embedding inference is often a single forward pass of a neural network) but limited in expressivity since the weights are fixed at test time.

A third approach skips test-time adaptation entirely. Instead, it trains one model to be robust across an entire family of environments (Tobin et al., 2017; Peng et al., 2018; Andrychowicz et al., 2020; Lee et al., 2020). This approach does not require test-time adaptation mechanisms, but sacrifices accuracy on the test-time environment for robustness across the whole family.

Our approach, CLAW, is a combination of the first and second approaches. We use a test-time context embedding to generate low-rank adapters (Hu et al., 2022) that modify the model weights. CLAW takes the best of both approaches: model adaptation is nearly instantaneous (a single forward pass of the hypernetwork), and has the expressivity of full-weight finetuning, depending on the adapter rank.

2.2 Selecting What to Adapt

During weight-based model adaptation, it is not always clear what weights within the model should be adapted. Some approaches adapt all the weights, e.g., TD-MPC2 adapts the entire model during online learning (Hansen et al., 2024). This has the benefit of maximum adaptation expressivity at the cost of increased test-time computation and potential model collapse (Smith et al., 2022a;b; Yin et al., 2025). Other approaches adapt specific world model modules, e.g., the dynamics module (Levy et al., 2026), or specific layers, e.g., the last layers (Levy et al., 2025; Harrison et al., 2018; Davydov et al., 2025). These approaches are computationally cheaper and avoid model collapse at the expense of manual module/layer selection and reduced adaptation expressivity. A third approach does not modify weights and instead modulates the model's intermediate activations, e.g., FiLM's feature-wise affine transforms (Perez et al., 2018). This also avoids manual selection, but limits adaptation to modulation of existing activations rather than a full weight-space update, which is more expressive (Jayakumar et al., 2020).

Our approach is different because it inserts LoRA adapters (Hu et al., 2022) into all layers of the world model. Therefore, we maintain most of the adaptation expressivity of full-model adaptation at a fraction of test-time compute cost. Moreover, the pretraining procedure (Algorithm 1) implicitly teaches the hypernetwork how strongly to adapt each layer, obviating the need for manual module/layer selection.

2.3 Hypernetworks in RL

Hypernetworks have been used in meta-RL before (Finn et al., 2017; Nagabandi et al., 2019; Raghu et al., 2020) to replace test-time gradient descent with a forward pass of a neural network (Huang et al., 2021; Xian et al., 2021; Przewięźlikowski et al., 2024; Beck et al., 2023; Rezaei-Shoshtari et al., 2023; Beukman et al., 2023). These methods often produce full weight tensors (instead of LoRA adapters).

In the domain of large language models (LLMs), existing work generates LoRA adapters from text prompts (Ye & Ren, 2021; Mahabadi et al., 2021; Ortiz-Barajas et al., 2024; Charakorn et al., 2025; Chen et al., 2025; Jukić et al., 2025; Liu et al., 2026). A recent paper (Bianchi et al., 2026) proposes generating LoRA adapters for a robot policy from a language instruction and a demo video. These are all forms of amortized optimization (Amos, 2023) in which a hypernetwork is trained to amortize test-time low-rank adaptation.

Existing hypernetwork approaches either generate full model weights (which is expensive and/or intractable for large world models), or generate adapters from explicit task descriptions for a fixed pretrained model. Neither approach addresses efficient world model adaptation from test-time transitions. CLAW fills this gap by jointly pretraining the world model and hypernetwork to infer adapters directly from test-time interactions, enabling repeated, low-cost adaptation during deployment.

3 PRELIMINARIES

Each environment e is a Markov Decision Process (MDP), which we assume is drawn at test time from a uniform distribution $p(\mathcal{E})$. For example, $p(\mathcal{E})$ could assign probabilities to bipedal locomotion environments that differ in actuator strength. At each step t, the agent receives state s_t, selects an action, and receives reward r_t together with the next state s_{t+1}.

We define two functions that produce the agent's actions: policies and planners. Policies produce actions directly from states, i.e., $a_t = \pi(s_t)$, and are often instantiated as neural networks. In contrast, planners select an action by evaluating candidate actions using a learned model of the MDP. That is, given world model parameters θ, a planner call is given by $a_t = \pi(s_t, \theta)$, where we overload π for notational convenience in the next paragraph. Planners are often instantiated as sampling-based algorithms, e.g., MPPI (Williams et al., 2015).

We refer to the tuple $\zeta_t = (s_t, a_t, r_t, s_{t+1})$ as a transition. A trajectory $\tau = (\zeta_0, \zeta_1, \ldots, \zeta_{M-1})$ collects a sequence of M transitions. $p(\tau \mid e, \pi)$ denotes the distribution over trajectories induced by acting in environment e according to the policy or planner π. A batch of trajectories with size N is the tuple $b = (\tau_1, \ldots, \tau_N)$, where each trajectory is drawn independently and identically distributed from $p(\tau \mid e, \pi)$. At test time, the agent interacts with environments from $p(\mathcal{E})$ which it may or may not have encountered during pretraining.

3.1 WORLD MODEL

World models are learned approximations of MDPs that agents can use to plan actions. They often consist of an encoder, a dynamics function, a value function, and a reward function (Hafner et al., 2020; 2023; Hansen et al., 2024; Levy et al., 2026). These modules are neural networks jointly trained using data from high-fidelity simulation or the real world. We refer to the parameters for all modules as θ_w, the *base* parameters. In this paper, the agent uses θ_w with a planner π_{upd} to select actions, i.e., $a_t = \pi_{\text{upd}}(s_t, \theta_w)$. Figure 1 shows the control flow for action selection given a set of candidate actions.

First, the encoder produces a latent state z_t from an environment state s_t. Then, the dynamics module predicts the next latent state z_{t+1} given z_t and a_t. Next, the reward and value modules use z_{t+1} to compute the reward r_{t+1} and value v_{t+1}. Finally, the planner ranks a_t using the estimated return $r_{t+1} + \gamma\, v_{t+1}$, where $\gamma \in [0, 1)$ is the discount factor.

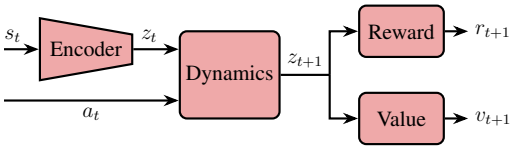


Figure 1: A single forward pass of a world model used to plan by maximizing predicted return. Blocks in red have trainable parameters, which constitute θ_w.

In this paper, we apply our method, CLAW, to the TD-MPC2 world model architecture (Hansen et al., 2024) because it consistently achieves strong performance across a variety of continuous control tasks using a single set of hyperparameters.[1] We emphasize that CLAW is agnostic to world model architecture since our only requirement is that we can sample environments during pretraining and insert LoRA adapters into world model modules.

[1] TD-MPC2 pretrains a model-free policy to seed a sampling-based planner. To avoid confusion, we include the parameters for this policy within θ_w.

Therefore, CLAW can be applied to, e.g., action-conditioned JEPA world models (Maes et al., 2026; Assran et al., 2025) or Dreamer-style architectures (Hafner et al., 2020; 2023).

3.2 Low-Rank Adaptation

At test time, the base parameters θ_w stay frozen and we insert LoRA adapters (Hu et al., 2022) into the world model's layers. Concretely, for an adapted layer with base weight matrix $W_0 \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, where d_{out} and d_{in} are the layer's output and input dimensions, the effective weight is

$$W = W_0 + AB, \qquad A \in \mathbb{R}^{d_{\text{out}} \times d_r}, \qquad B \in \mathbb{R}^{d_r \times d_{\text{in}}}, \qquad d_r \ll \min(d_{\text{out}}, d_{\text{in}}). \tag{1}$$

The factors A and B constitute the adapter, and d_r is the low-rank bottleneck dimension. We define ϕ_w as the collection of all A, B factors across the adapted layers. For simplicity, we adapt all world model layers and keep rank d_r constant.

3.3 Hypernetworks

A hypernetwork is a neural network H_{θ_h} that outputs the weights of another network (Ha et al., 2017). Given an input x, the target network parameters ϕ are produced as

$$\phi = H_{\theta_h}(x), \tag{2}$$

where θ_h are the weights for the hypernetwork. In the next section, we present more details on the architecture and training procedure for our LoRA-generating hypernetwork.

4 Approach

We now present CLAW, a model-based RL adaptation algorithm that uses a hypernetwork to generate low-rank adapters at test time. This section is structured as follows: First, we detail the architecture of our hypernetwork (Section 4.1). Then, we describe a pretraining procedure to jointly train the base model and hypernetwork (Section 4.2).

4.1 Hypernet-Based Adapter Generation

We use a hypernetwork to generate LoRA adapters for each environment, online at test time. Using a context batch b_{con} of N_{con} trajectories of M transitions each, it generates LoRA adapters ϕ_w for the world model's base weights θ_w. Formally:

$$\phi_w = H_{\theta_h}(b_{\text{con}}). \tag{3}$$

We now describe the architecture of H and how b_{con} is used to generate ϕ_w.

First, we generate an embedding u_{emb} for each transition $\zeta \in b_{\text{con}}$ using a small multi-layer perceptron (MLP) h_{emb}. Then, we average the embeddings and feed them into a second MLP h_{con} to produce a context embedding u_{con} that identifies the environment. Note that embedding each transition independently, then averaging, discards information about transition order.

Once we have identified the environment with u_{con}, we feed it into h_{gen}, an MLP, to generate $\Delta\phi_w$, which is added to a learned template ϕ_0 to produce ϕ_w, i.e.,

$$\phi_w = \phi_0 + \Delta\phi_w. \tag{4}$$

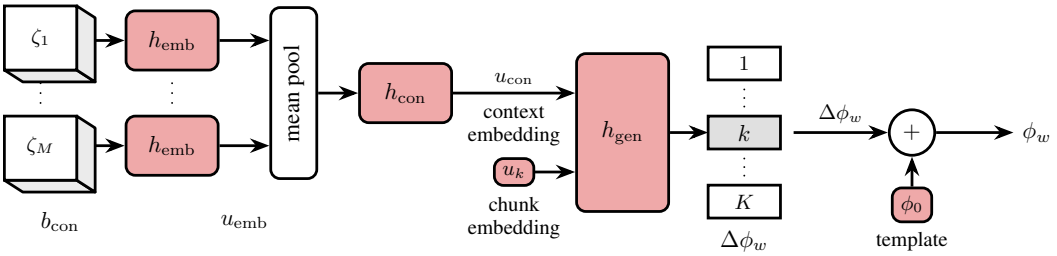


Figure 2: Forward pass of the hypernetwork H_{θ_h} when generating chunk k of $\Delta\phi_w$ using a batch b_{con} of N_{con} trajectories, each with M transitions. In the b_{con} block, depth represents the batch dimension. Blocks in red have trainable parameters which constitute θ_h.

Templates have been shown to improve learning performance in hypernetworks (Beck et al., 2023).

Generating $\Delta\phi_w$ from a single forward pass would require an impractically large generator network h_{gen}. Instead, h_{gen} is a small network we reuse to generate $\Delta\phi_w$ in chunks. Concretely, following Von Oswald et al. (2020), we divide $\Delta\phi_w$ into K chunks and generate chunk k by feeding the chunk embedding u_k and the context embedding u_{con} into h_{gen}. Figure 2 illustrates the full procedure.

4.2 Amortizing Low-Rank Adaptation

We now describe the procedure to jointly pretrain the world model and hypernetwork, and thereby amortize test-time low-rank adaptation. At a high level, CLAW wraps a standard world model pretraining algorithm with a loop that iterates between environments sampled from an environment family. This allows us to train the hypernetwork to generate adapters for each environment. In the following subsections we explain: **1)** a single-environment training step for the base model and hypernetwork parameters (i.e., θ_w, θ_h), **2)** a training loss that averages over environments, thus meta-training the base and hypernetwork for online adaptation, and **3)** the full pretraining algorithm, instantiated for the TD-MPC2 architecture (Hansen et al., 2024).

Single-Environment Training Step. Given environment j, we start by filling two buffers: $\mathcal{B}^j_{\text{con}}$, a context buffer containing trajectories collected using a context-collection policy π_{con}, and $\mathcal{B}^j_{\text{upd}}$, a buffer containing trajectories collected using the agent's planner, π_{upd}. Next, we sample a batch b_{con} of N_{con} trajectories from $\mathcal{B}^j_{\text{con}}$ and produce environment-specific low-rank adapters using the hypernetwork, i.e., $\phi_w = H_{\theta_h}(b_{\text{con}})$. Then, we combine θ_w and ϕ_w per (1), and use a batch b_{upd} of N_{upd} trajectories sampled from $\mathcal{B}^j_{\text{upd}}$ to compute the loss $\mathcal{L}$ (detailed in the next section). Finally, we differentiate $\mathcal{L}$ with respect to θ_w and θ_h, and update θ_w and θ_h with an Adam optimizer (Kingma & Ba, 2015).

The result is a hypernetwork trained to produce low-rank adapters ϕ_w for θ_w using b_{con}, and a base model trained to be adapted with ϕ_w. A visualization of this procedure can be seen in Fig. 3.

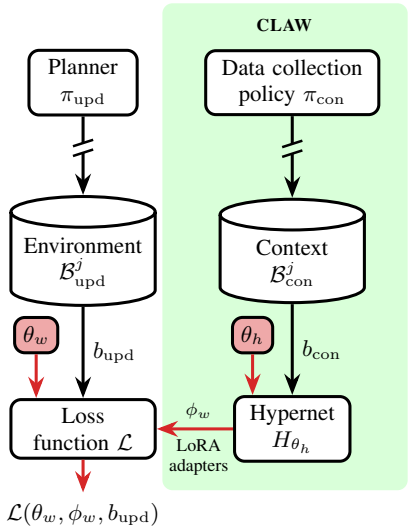


Figure 3: Computing the loss $\mathcal{L}$ for environment j during CLAW pretraining. Green marks what CLAW adds to standard world model pretraining loss computation. Red marks trainable parameters and their gradient path, and ticks mark a stopped gradient. CLAW can be applied to any world model training algorithm with the same high-level computation graph.

Our method collects b_{con} using the context-collection policy π_{con} instead of the agent's planner. There are two reasons for this approach:

1. It ensures the distribution of context trajectories is the same during training and testing, facilitating environment identification for the hypernetwork H.
2. In future work, we can directly optimize the policy π_{con} to produce trajectories b_{con} which are better at identifying the environment and/or lead to better test-time performance.

In our experiments, π_{con} is always a random-action policy. This fixes context data quality across adaptation baselines so that performance differences reflect the adaptation method rather than the collected context data. However, practitioners may choose to use a different policy, e.g., one that is safe for real-life deployment (but not necessarily high-performing). The use of a non-random π_{con} is fully supported by CLAW as long as π_{con} can be queried during pretraining.

Multi-Environment Training Loss. The loss in Fig. 3 decomposes as

$$\mathcal{L}(\theta_w, \phi_w, b_{\text{upd}}) \;=\; \mathcal{L}_{\text{TDMPC2}}(\Theta_w, b_{\text{upd}}) \;+\; \lambda\|\phi_w\|_2^2, \tag{5}$$

where Θ_w are the merged base and adapter weights per (1), $\lambda \in \mathbb{R}_{\geq 0}$ weighs the adapter's L_2 penalty, and $\mathcal{L}_{\text{TDMPC2}}$ is the training loss for a TDMPC2 world model described by Hansen et al.

(2024). If a different world model architecture is used, $\mathcal{L}_{\text{TDMPC2}}$ may be replaced with the corresponding loss. Figure 3 computes a single-environment, single-batch approximation of the objective in the following minimization problem:

$$\min_{\theta_w,\,\theta_h} \; \mathbb{E}_e \left[\mathbb{E}_{b_{\text{con}},\,b_{\text{upd}}} \left[\mathcal{L} \right] \right], \tag{6}$$

where $e \sim p(\mathcal{E})$, and b_{con} and b_{upd} are batches of N_{con} and N_{upd} trajectories, respectively. Each trajectory is drawn i.i.d. from $p(\tau \mid e, \pi_{\text{con}})$ and $p(\tau \mid e, \pi_{\text{upd}})$, respectively.

Problem (6) illustrates how our method can be applied to any world model architecture. If we remove b_{con} and the $\lambda\|\phi_w\|_2^2$ term, the inner expectation of (6) is the objective of a standard world model training problem. Our approach, CLAW, defines the hypernetwork machinery (green box in Fig. 3), and uses it to amortize low-rank adaptation by wrapping the standard objective with an expectation over the environment family $p(\mathcal{E})$.

Applying CLAW to the TD-MPC2 Architecture. Algorithm 1 shows the TD-MPC2 (Hansen et al., 2024) pretraining algorithm modified to use CLAW. It includes the following additions: **1)** logic to sample a rollout environment from $p(\mathcal{E})$ every episode, thereby approximating the outer expectation in (6); **2)** logic to sample a *training* environment from $p(\mathcal{E})$ every step, thereby stabilizing multi-environment training; **3)** a buffer $\mathcal{B}^j_{\text{con}}$ per environment j with corresponding samples b_{con}, and **4)** a forward pass of H_{θ_h} to produce adapters ϕ_w given b_{con}.

We denote additions to the original TD-MPC2 algorithm with green highlights in Algorithm 1. Our algorithm uses a modified version of the TD-MPC2 weight update, TDMPC2UPDATE, which we include as Algorithm 2 in Appendix A. The result of Algorithm 1 is a base TD-MPC2 model and hypernet that can generate test-time LoRA adapters for any layer of the base model.

Algorithm 1: CLAW + TD-MPC2 pretraining

Input: Environment family $\{E^1, \ldots, E^n\}$, context-collection policy π_{con}, initial θ_w, θ_h, total training steps T

// Context data collection
for $j = 1, \ldots, n$ **do**
  Init. and fill $\mathcal{B}^j_{\text{con}}$ by acting in E^j with π_{con}

// Update data prefill
for $j = 1, \ldots, n$ **do**
  Init. and fill $\mathcal{B}^j_{\text{upd}}$ by acting in E^j with π_{upd}

// Online Interaction and Training Loop
$i \sim \text{Uniform}(\{1, \ldots, n\})$ ▷ Sample rollout env.
$s \leftarrow E^i.\text{reset}()$
for $t = 1, \ldots, T$ **do**
  $k \sim \text{Uniform}(\{1, \ldots, n\})$ ▷ Sample training env.

  // Update base and hypernet parameters
  $b_{\text{upd}} \leftarrow \text{Sample}(\mathcal{B}^k_{\text{upd}})$
  $b_{\text{con}} \leftarrow \text{Sample}(\mathcal{B}^k_{\text{con}})$
  $\phi_w \leftarrow H_{\theta_h}(b_{\text{con}})$ ▷ Training env. adapters
  $\Theta_w \leftarrow (\theta_w, \phi_w)$ ▷ Merge adapters
  $(\theta_w, \theta_h) \leftarrow \text{TDMPC2UPDATE}(b_{\text{upd}}, b_{\text{con}}, \Theta_w, \theta_h)$

  // Plan and act
  $b^i_{\text{con}} \leftarrow \text{Sample}(\mathcal{B}^i_{\text{con}})$
  $\phi_w \leftarrow H_{\theta_h}(b^i_{\text{con}})$ ▷ Rollout env. adapters
  $\Theta_w \leftarrow (\theta_w, \phi_w)$
  $a \leftarrow \pi_{\text{upd}}(s, \Theta_w)$ ▷ Plan with latest parameters
  $s', r, \text{done} \leftarrow E^i.\text{step}(a)$

  // Update environment and buffers
  $\mathcal{B}^i_{\text{upd}} \leftarrow \mathcal{B}^i_{\text{upd}} \cup \{(s, a, r, s')\}$
  if done **then**
    $i \sim \text{Uniform}(\{1, \ldots, n\})$ ▷ Sample rollout env.
    $s \leftarrow E^i.\text{reset}()$
  else
    $s \leftarrow s'$

Output: θ_w, θ_h

We emphasize that CLAW can be applied regardless of the chosen world model architecture or training algorithm. Specifically, an adapter-generating hypernetwork can be trained as long as one can sample environments, maintain a buffer for each of them, and maintain an unbroken gradient chain between the loss and hypernetwork (as shown by the red lines in Fig. 3).

5 EXPERIMENTS

We test CLAW on the environment families shown in Table 1, where each family consists of a set of environments that differ in one property. We use 5 seeds per experiment and report 95% confidence intervals for the mean across seeds, computed with Student's t distribution. Appendix B lists the architecture, optimization, and per-experiment hyperparameters needed to reproduce every experiment below. We will include a public codebase in the final version of this paper.

5.1 EXPERIMENT 1: ONLINE ADAPTATION

Question. How does CLAW's performance compare to online, gradient-based adaptation?

Family	Robot	Objective	Environment property	# Environments
Walker	Walker2d	Walk forward	Actuator strength	20
Reach	Meta-World arm	Reach target position	Goal position	6
Push	Meta-World arm	Place puck at goal	Puck goal position	6
Go1	Unitree Go1	Run	Running direction (forward or backward)	2
Cheetah/Walker	Half-Cheetah / Walker2d	Run forward	Robot embodiment	2

Table 1: Environment families used in our experiments, adapted from DeepMind Control Suite (Tassa et al., 2018) and Meta-World (Yu et al., 2019). The environment property column lists the property that varies between environments in a family.

Setup. For each environment in a family, we begin by collecting 3 episodes of context data with π_{con} (7.5–75s of data depending on the environment). Then, we roll out a pretrained model, adapting it at every step using the context data. Gradient-based baselines take one gradient step per environment step, adapting a set of LoRA adapters. On the other hand, CLAW uses its hypernetwork to generate new LoRA adapters at every environment step. We report episode return (or success rate) averaged across environments and seeds.

Baselines. We compare CLAW against adaptation on models trained on a single task (ST) or with Domain Randomization (DR; Tobin et al., 2017). DR pretrains a base model on all environments in the family, and ST pretrains a base model on a single environment in the family. We also compare against a non-adapting oracle baseline, which appends the environment's ground-truth parameter directly to the state, e.g., actuator strength, or running direction (specified with a one-hot vector). Since every other method must infer this parameter from context data, the oracle's performance serves as a ceiling on the performance achievable with perfect environment identification. World model hyperparameters are fixed across baselines.

Results. Results are shown in Fig. 4. In all families, CLAW closely tracks the oracle's performance. In the manipulation families (Push and Reach), DR and ST climb toward that ceiling, but they take many environment steps to get there, and performance is very noisy. In Walker, DR and ST get worse instead of catching up. In Go1, neither DR nor ST catches up. DR's low performance is likely due to Go1's two running directions forming a bimodal distribution of behavior, which DR training resolves into a single "average" model that is not good at either behavior. ST pretrains on the forward direction alone, so it starts far from the backward environment and never closes the gap.

5.2 Experiment 2: Adaptation when Data Is Limited

Question. Given that test-time data is often expensive and/or scarce, how does CLAW's performance compare to the return achieved by applying many steps of gradient-based adaptation on a fixed test-time dataset?

Setup. For each family and environment, we collect 3 episodes of context data using π_{con} (between 7.5 and 75 seconds depending on the environment). Then, we use this data to adapt all methods offline, freeze the models, and evaluate them. For the gradient-based adaptation baselines, we sweep a range of offline gradient steps, thereby testing how performance changes as a function of offline adaptation. We report the same metrics as Experiment 1.

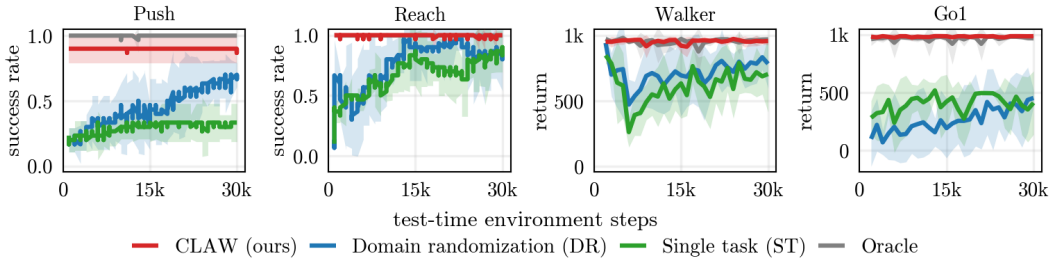


Figure 4: Experiment 1. Episode return (or success rate) vs. test-time environment steps, averaged over all environments in a family.

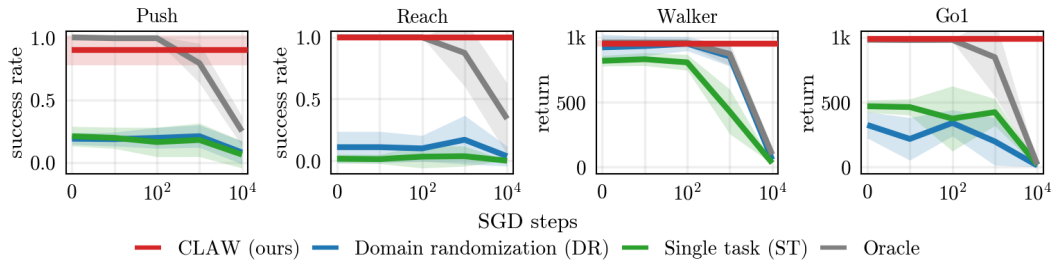


Figure 5: Experiment 2. CLAW's single hypernetwork pass (red) vs. gradient-based baselines (DR, ST, oracle) as a function of SGD steps.

Baselines. As in Experiment 1, we compare to DR, ST, and an oracle, here also swept over adaptation gradient steps.

Results. Results are shown in Fig. 5. CLAW mostly tracks oracle performance, and outperforms other baselines. Moreover, as the number of SGD steps increases, performance collapses for all gradient-based methods as they overfit to the data. This is supported by the collapse of the oracle: since it already knows the environment, the only thing left for it to fit is noise in the small dataset.

5.3 Experiment 3: Comparison to In-Context Learning

Question. Does CLAW's performance depend on generating expressive world-model adapters, or merely conditioning on the context embedding? How does this relationship change with model size?

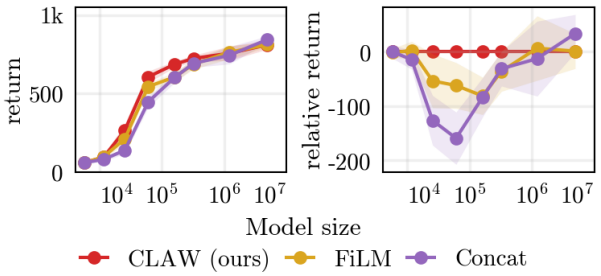


Figure 6: Experiment 3. **Left:** return vs. model size. **Right:** reward relative to CLAW.

Setup. For each environment in the Cheetah/Walker family in Table 1, we roll out one episode adapting at every timestep (as in previous experiments). We choose this family because we need a harder family than the others to tell the in-context baselines apart. Cheetah/Walker provides this since state and action spaces differ completely across environments. We also sweep over model size and report the same evaluation metrics as Experiment 1.

Baselines. We compare against two in-context learning baselines that condition a fixed model on context, rather than adapting its weights, to isolate the source of CLAW's performance. Concatenation (Concat) feeds the hypernetwork's context embedding directly into the world model's state (instead of using it to generate adapters); this is akin to the approaches of Kumar et al. (2021); Rakelly et al. (2019); Liu et al. (2025). We also compare to a hypernetwork trained to generate FiLM parameters (Perez et al., 2018), which scale and shift features.

Results. Results are shown in Fig. 6. At the smallest sizes, no method learns enough to separate performance. At the largest sizes, the base model has enough capacity to represent every environment without conditioning, so performance does not separate either. CLAW's advantage is clearest at intermediate sizes where it outperforms both baselines. These results align with those of Jayakumar et al. (2020), who predict that hypernetwork-generated weights are more expressive than the conditioning mechanisms of FiLM and Concat.

5.4 Experiment 4: LoRA Rank Sensitivity

Question. For a fixed model size, how does CLAW's performance change with LoRA rank?

Setup. We fix the model size and pretrain a hypernetwork with CLAW for a range of LoRA ranks. We use the Cheetah/Walker family and select an intermediate size where in-context baseline performance is separable. As LoRA rank increases, we increase the number of chunk embeddings K in Fig. 2 and keep all other hyperparameters fixed. This controls for hypernetwork capacity, so any

performance change reflects the rank increase, not a larger generator network. Then, we evaluate online adaptation performance as in Experiment 1.

Baselines. We compare against the same in-context learning baselines as Experiment 3.

Results. Results are shown in Fig. 7. We report performance vs. adapter rank and adapter size as a percentage of the base parameter count. CLAW's performance rises with rank and percentage. On the parameter axis, FiLM outperforms CLAW at low budgets, and CLAW needs around four times FiLM's parameter budget to match its return. Past that budget, CLAW surpasses FiLM, whose performance stays constant since it has no comparable knob to spend a larger budget on.

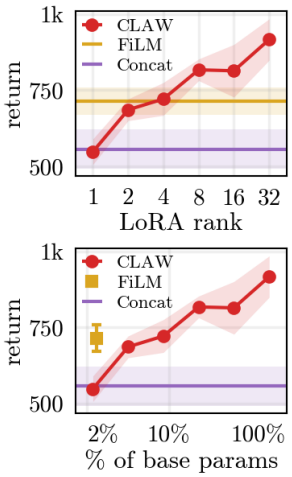


Figure 7: Experiment 4. **Top:** return vs. LoRA rank. **Bottom:** return vs. adapter size as a percentage of the base parameter count.

5.5 Experiment 5: Post-Hoc Hypernet Comparison

Question. How does training the base model and hypernetwork jointly (CLAW) compare to training a hypernetwork on top of a frozen pretrained base model (cf. Bianchi et al., 2026)?

Setup. We evaluate performance on Cheetah using the same metrics as previous experiments. The post-hoc hypernet's total number of training steps and training data match what CLAW would see for this family (500k steps total, split between both environments).

Baseline. First, we pretrain a plain TD-MPC2 base on Walker only for 250k steps. Then, we freeze it and train a hypernetwork for 250k steps to generate adapters for the Cheetah environment. We refer to this as the "post-hoc hypernet." This matches the approach proposed by Bianchi et al. (2026), who train a hypernetwork to generate LoRA adapters using a frozen robotics foundation model.

Results. Results are shown in Fig. 8. CLAW outperforms a post-hoc hypernet, showing that jointly training the base model and hypernetwork leads to better performance.

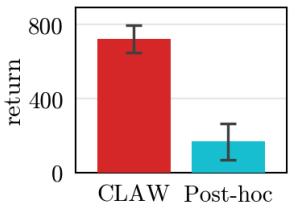


Figure 8: Experiment 5. Cheetah return for CLAW vs. post-hoc baseline.

6 Conclusion, Limitations, and Future Work

We introduce CLAW, a method for world model adaptation that uses a hypernetwork to generate LoRA adapters online, replacing gradient-based test-time adaptation. Across all tested environment families, CLAW outperforms online gradient-based and in-context learning baselines. Moreover, under limited data, CLAW avoids the overfitting that degrades gradient-based approaches. We also present ablations showing that CLAW's performance comes from generating world model adapters rather than merely conditioning on a context embedding, that performance improves with LoRA rank, and that training the hypernetwork jointly with the base model outperforms training it post hoc.

CLAW is limited by the identifiability of the environment when using the data-collection policy π_{con}. Additionally, CLAW learns to generate adapters only for the environments seen during pretraining. Future work includes pretraining and/or architectural improvements for better generalization, applying CLAW to sim-to-real transfer, optimizing the data-collection policy for downstream performance, and algorithmically selecting which environments to train on (Dennis et al., 2020).

AI USE STATEMENT

We used generative AI to implement methods and for general coding assistance, to edit and format this paper, and to summarize existing literature and surface relevant work we had not considered. We did not use generative AI for any other task. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We are not aware of any ethical concerns raised by this work.

REPRODUCIBILITY STATEMENT

Appendix B reports every hyperparameter used to reproduce our experiments. We will also include a code base with instructions on how to run every experiment in this paper.

REFERENCES

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Mido Assran, Adrien Bardes, David Fan, Quentin Garrido, Russell Howes, Mojtaba Komeili, Matthew Muckley, Ammar Rizvi, Claire Roberts, Koustuv Sinha, Artem Zholus, et al. V-JEPA 2: Self-supervised video models enable understanding, prediction and planning. arXiv preprint arXiv:2506.09985, 2025.

Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. In *International Conference on Machine Learning*, pp. 1577–1594. PMLR, 2023.

Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on robot learning*, pp. 1478–1487. PMLR, 2023.

Michael Beukman, Devon Jarvis, Richard Klein, Steven James, and Benjamin Rosman. Dynamics generalisation in reinforcement learning via adaptive context-aware policies. In *Neural Information Processing Systems*, 2023.

Christian Bianchi, Siamak Yousefi, Alessio Sampieri, Andrea Roberti, Luca Rigazio, Fabio Galasso, and Luca Franco. Robotic policy adaptation via weight-space meta-learning. arXiv preprint arXiv:2606.07217, 2026.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-lora: Instant transformer adaption. In *International Conference on Machine Learning*, 2025.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Neural Information Processing Systems*, 2021.

Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass. In *International Conference on Learning Representations*, 2025.

Alexander Davydov, Franck Djeumou, Marcus Greiff, Makoto Suminaka, Michael Thompson, John Subosits, and Thomas Lew. First, learn what you don't know: Active information gathering for driving at the limits of handling. *IEEE Robotics and Automation Letters*, 2025.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. arXiv preprint arXiv:1611.02779, 2016.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

David Ha and Jürgen Schmidhuber. World models. arXiv preprint arXiv:1803.10122, 2018.

David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. arXiv preprint arXiv:2301.04104, 2023.

Nicklas Hansen, Hao Su, and Xiaolong Wang. Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*, 2024.

James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. In *International Workshop on the Algorithmic Foundations of Robotics*, pp. 318–337. Springer, 2018.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *IEEE International Conference on Robotics and Automation*, 2021.

Stephen James, Michael Bloesch, and Andrew J. Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on Robot Learning*, 2018.

Siddhant M. Jayakumar, Wojciech M. Czarnecki, Jacob Menick, Jonathan Schwarz, Jack Rae, Simon Osindero, Yee Whye Teh, Tim Harley, and Razvan Pascanu. Multiplicative interactions and where to find them. In *International Conference on Learning Representations*, 2020.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. arXiv preprint arXiv:2210.03094, 2022.

Josip Jukić, Martin Tutek, and Jan Šnajder. Context parametrization with compositional adapters. arXiv preprint arXiv:2509.22158, 2025.

Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. Cosmos policy: Fine-tuning video models for visuomotor control and planning. arXiv preprint arXiv:2601.16163, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. In *International Conference on Learning Representations*, 2022.

Ashish Kumar, Zipeng Fu, Deepak Pathak, and Jitendra Malik. Rma: Rapid motor adaptation for legged robots. In *Robotics: Science and Systems*, 2021.

JB Lanier, Kyungmin Kim, Armin Karamzade, Yifei Liu, Ankita Sinha, Kat He, Davide Corsi, and Roy Fox. Adapting world models with latent-state dynamics residuals. arXiv preprint arXiv:2504.02252, 2025.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation. In *International Conference on Learning Representations*, 2023.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 2020.

Jacob Levy, Jason Gibson, Bogdan Vlahov, Erica Tevere, Evangelos Theodorou, David Fridovich-Keil, and Patrick Spieler. Meta-learning online dynamics model adaptation in off-road autonomous driving. In *Robotics: Science and Systems*, 2025. doi: 10.15607/RSS.2025.XXI.139.

Jacob Levy, Tyler Westenbroek, Kevin Huang, Fernando Palafox, Patrick Yin, Shayegan Omidshafiei, Dong-Ki Kim, Abhishek Gupta, and David Fridovich-Keil. Simulation distillation: Pretraining world models in simulation for rapid real-world adaptation. In *Robotics: Science and Systems*, 2026. doi: 10.15607/RSS.2026.XXII.017.

Min Liu, Deepak Pathak, and Ananye Agarwal. Locoformer: Generalist locomotion via long-context adaptation. arXiv preprint arXiv:2509.23745, 2025.

Yewei Liu, Xiyuan Wang, Yansheng Mao, Yoav Gelbery, Haggai Maron, and Muhan Zhang. Shine: A scalable in-context hypernetwork for mapping context to lora in a single pass. arXiv preprint arXiv:2602.06358, 2026.

Lucas Maes, Quentin Le Lidec, Damien Scieur, Yann LeCun, and Randall Balestriero. LeWorld-Model: Stable end-to-end joint-embedding predictive architecture from pixels. arXiv preprint arXiv:2603.19312, 2026.

Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *International Conference on Learning Representations*, 2018.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations*, 2019.

Jesus-German Ortiz-Barajas, Helena Gomez-Adorno, and Thamar Solorio. Hyperloader: Integrating hypernetwork-based lora and adapter layers into multi-task transformers for sequence labelling. arXiv preprint arXiv:2407.01411, 2024.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation*, 2018.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

Samuel Pfrommer, Mathew Halm, and Michael Posa. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Conference on Robot Learning*, 2020.

Physical Intelligence, Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, Michael Equi, Adnan Esmail, Yunhao Fang, Chelsea Finn, Catherine Glossop, Thomas Godden, Ivan Goryachev, Lachy Groom, Hunter Hancock, Karol Hausman, Gashon Hussein, Brian Ichter, Szymon Jakubczak, Rowan Jen, Tim Jones, Ben Katz, Liyiming Ke, Chandra Kuchi, Marinda Lamb, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Yao Lu, Vishnu Mano, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Charvi Sharma, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, Will Stoeckle, Alex Swerdlow, James Tanner, Marcel Torne, Quan Vuong, Anna Walling, Haohuan Wang, Blake Williams, Sukwon Yoo, Lili Yu, Ury Zhilinsky, and Zhiyuan Zhou. $\pi^*_{0.6}$: a VLA that learns from experience. arXiv preprint arXiv:2511.14759, 2025. Physical Intelligence Technical Report.

M. Przewięźlikowski, P. Przybysz, J. Tabor, M. Zięba, and P. Spurek. Hypermaml: Few-shot adaptation of deep models with hypernetworks. *Neurocomputing*, 2024.

Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, 2019.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent. *Transactions on Machine Learning Research*, 2022.

Sahand Rezaei-Shoshtari, Charlotte Morissette, Francois Robert Hogan, Gregory Dudek, and David Meger. Hypernetworks for zero-shot transfer in reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2023.

Laura Smith, J. Chase Kew, Xue Bin Peng, Sehoon Ha, Jie Tan, and Sergey Levine. Legged robots that keep on learning: Fine-tuning locomotion policies in the real world. In *2022 International Conference on Robotics and Automation (ICRA)*, pp. 1593–1599. IEEE, 2022a.

Laura Smith, Ilya Kostrikov, and Sergey Levine. A walk in the park: Learning to walk in 20 minutes with model-free reinforcement learning. arXiv preprint arXiv:2208.07860, 2022b.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. arXiv preprint arXiv:1801.00690, 2018.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. In *8th International Conference on Learning Representations (ICLR 2020)(virtual)*. International Conference on Learning Representations, 2020.

Yiqi Wang, Mrinal Verghese, and Jeff Schneider. Latent policy steering with embodiment-agnostic pretrained world models. arXiv preprint arXiv:2507.13340, 2025.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. arXiv preprint arXiv:1509.01149, 2015.

Zhou Xian, Shamit Lal, Hsiao-Yu Tung, Emmanouil Antonios Platanios, and Katerina Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. In *International Conference on Learning Representations*, 2021.

Jie Xu, Eric Heiden, Iretiayo Akinola, Dieter Fox, Miles Macklin, and Yashraj Narang. Neural robot dynamics. arXiv preprint arXiv:2508.15755, 2025.

Qinyuan Ye and Xiang Ren. Learning to generate task-specific adapters from task description. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

Patrick Yin, Tyler Westenbroek, Simran Bagaria, Kevin Huang, Ching-An Cheng, Andrey Kolobov, and Abhishek Gupta. Rapidly adapting policies to the real world via simulation-guided fine-tuning. In *International Conference on Learning Representations*, 2025.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.

Wenhao Yu, Jie Tan, C. Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *Robotics: Science and Systems*, 2017.

Xilun Zhang, Shiqi Liu, Peide Huang, William Jongwon Han, Yiqi Lyu, Mengdi Xu, and Ding Zhao. Dynamics as prompts: In-context learning for sim-to-real system identifications. *IEEE Robotics and Automation Letters*, 2025.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *International Conference on Learning Representations*, 2020.

Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, 2019.

A TDMPC2UPDATE: WORLD MODEL AND POLICY UPDATE

Algorithm 2 is the TD-MPC2 weight update (Hansen et al., 2024) modified to train the hypernetwork alongside the base model. It runs in two phases. The first phase computes the world model loss and updates the world model base slice θ_w^{wm} jointly with the hypernetwork parameters θ_h. The second phase computes the policy loss and updates the policy base slice θ_w^{π}, again jointly with θ_h. Between the two phases we regenerate the policy adapter ϕ_w^{π} with the updated hypernetwork, so the policy loss uses the adapter the current hypernetwork produces. The update ends by refreshing the target critic and the Q-scale. The target critic EMA tracks base weights only, since its adapter is regenerated on every call.

Algorithm 2 TDMPC2UPDATE — world model, policy, and hypernetwork update

Input: b_{upd}, b_{con}, $\Theta_w = (\theta_w, \phi_w)$, where $\phi_w = (\phi_w^{\mathrm{wm}}, \phi_w^{\pi})$ are the world-model and policy adapter slices, θ_h

$\mathbf{z} \leftarrow \mathrm{encode}(b_{\mathrm{upd}}, \Theta_w)$

$\mathcal{L}_{\mathrm{TDMPC2,\,wm}} \leftarrow \mathrm{TDMPC2WorldModelLoss}(b_{\mathrm{upd}}, \mathbf{z}, \Theta_w)$

$\theta_w^{\mathrm{wm}}, \theta_w^{\pi} \leftarrow \theta_w$ ▷ break out world-model and policy slices

$\left(\theta_w^{\mathrm{wm}\prime}, \theta_h'\right) \leftarrow \mathrm{Optimizer}\left((\theta_w^{\mathrm{wm}}, \theta_h), \nabla_{(\theta_w^{\mathrm{wm}}, \theta_h)} \mathcal{L}_{\mathrm{TDMPC2,\,wm}}\right)$

$\theta_w' \leftarrow (\theta_w^{\mathrm{wm}\prime}, \theta_w^{\pi})$

$\phi_w^{\pi\prime} \leftarrow H_{\theta_h'}(b_{\mathrm{con}})$ ▷ Refresh policy adapter

$\Theta_w' \leftarrow (\theta_w', (\phi_w^{\mathrm{wm}}, \phi_w^{\pi\prime}))$

$\mathcal{L}_{\mathrm{TDMPC2},\,\pi} \leftarrow \mathrm{TDMPC2PolicyLoss}(\mathsf{sg}(\mathbf{z}),\ Q(\mathsf{sg}(\Theta_w'))\,,\ \Theta_w')$

$\left(\theta_w^{\pi\prime}, \theta_h''\right) \leftarrow \mathrm{Optimizer}\Big((\theta_w^{\pi}, \theta_h'), \nabla_{(\theta_w^{\pi}, \theta_h')} \mathcal{L}_{\mathrm{TDMPC2},\,\pi}\Big)$

$\theta_w'' \leftarrow (\theta_w^{\mathrm{wm}\prime}, \theta_w^{\pi\prime}),\quad \Theta_w'' \leftarrow (\theta_w'', (\phi_w^{\mathrm{wm}}, \phi_w^{\pi\prime}))$

Update target critic: EMA the critic base into a persistent variable; adapter regenerated fresh, not EMA'd

Update Q-scale

Output: θ_w'', θ_h''

B Hyperparameters and Experimental Details

This appendix reports every hyperparameter needed to reproduce our experiments. Tables 2 and 3 give the architecture, adapter, optimization, and planning hyperparameters shared by all methods and experiments unless a table below states otherwise. Tables 4 to 6 cover Experiments 1 and 2, which share pretrained checkpoints across the Walker, Reach, Push, and Go1 families. Tables 7 to 9 cover Experiments 3, 4, and 5, which all use the Cheetah/Walker family.

B.1 Shared Architecture and Adapter Hyperparameters

Component	Value
Encoder	
Hidden features	256, 512
SimNorm groups / temperature	8 / 1.0
Dynamics	
Hidden features	512, 512, 512
SimNorm groups / temperature	8 / 1.0
Critic	
Hidden features	512, 512
Ensemble size / subsampled per target	5 / 2
Return bins / range	101 / $[-10, 10]$
Dropout	0.01
Policy	
Hidden features	512, 512
Log-std range	$[-10, 2]$
Reward	
Hidden features	512, 512
Bins / range	101 / $[-10, 10]$
Hypernetwork H_{θ_h} (Section 4.1)	
Context dimension C	64
Transition-embedding MLP h_{emb} hidden width	256
Generator MLP h_{gen} hidden width	128
Chunk size c (parameters per generated chunk)	64
Chunk-embedding dimension	32

Table 2: Network architecture, shared by CLAW and all baselines (DR, ST, oracle), and by every experiment and family unless overridden in Tables 7 and 8. The number of hypernetwork chunks is $K = \lceil N_{\mathrm{adapt}}/c \rceil$, where N_{adapt} is the number of parameters in ϕ_w; it grows automatically with LoRA rank (Section 5.4).

Parameter	Value
TD-MPC2 loss and optimizer (Algorithm 2)	
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate (world model / encoder / policy)	3e-4 / 9e-5 / 3e-4
Hypernetwork learning rate (CLAW only)	3e-4
Gradient clip norm	20
Planning horizon	3
Discount factor γ	0.99
Temporal decay	0.5
Target critic EMA decay	0.99
Consistency / reward / value / entropy coefficients	20.0 / 0.1 / 0.1 / 1e-4
Adapter regularization weight λ ((5))	0.01
Batch size	256
Initial SGD burst before rollout begins	1,000 steps
Planner (MPPI (Williams et al., 2015))	
Horizon	3
Iterations	6
Population size	512
Policy-seeded trajectories	24
Temperature	0.5
Action noise std range	$[0.05, 2.0]$
Elites	64

Table 3: Optimization and planning hyperparameters, shared across CLAW and all baselines.

B.2 Experiments 1 and 2: Environment Families and Pretraining

Family	$\dim(s)$	$\dim(a)$	Episode length	Envs. (train)	Pretraining steps
Walker	24	6	1000	20	600,000
Reach	39	4	100	6	50,000
Push	39	4	100	6	300,000
Go1	39	12	1000	2	1,000,000

Table 4: Per-family pretraining setup for Experiments 1 and 2, which adapt the same checkpoints. All methods (CLAW, DR, ST, oracle) pretrain for the same number of steps per family with 5 seeds each. The oracle baseline appends the ground-truth environment parameter to the state, increasing $\dim(s)$ by 1 (Walker actuator scale, Go1 running direction) or 3 (Reach/Push goal position).

Method	**Adapts weights?**	**Mechanism**
CLAW	no (base frozen after pretraining)	One hypernet forward pass per environment step. Generates rank-16 LoRA adapters.
DR, ST	yes	one gradient step per environment step, on fresh rank-16 adapters
Oracle	no	ground-truth environment parameter appended to the state

Table 5: Experiment 1 online adaptation protocol. Every method rolls out for 30,000 environment steps per environment, with 5 seeds; Optimizer settings for DR/ST's gradient step are in Table 3.

Parameter	**Value**
Context data	3 episodes, collected with π_{con}
Offline gradient steps swept (DR, ST, oracle)	$\{0, 1, 100, 1000, 10000\}$
CLAW adaptation	1 hypernetwork forward pass (no gradient steps)
Evaluation episodes per environment	10
Seeds	5

Table 6: Experiment 2 offline adaptation protocol. All methods adapt on the same fixed context dataset, then freeze and evaluate. Pretraining is shared with Experiment 1 (Table 4).

B.3 Experiments 3–5: Cheetah/Walker Model-Capacity Studies

Experiments 3, 4, and 5 all use the Cheetah/Walker family ($\dim(s)$=24, padded for Cheetah; $\dim(a)$=6; episode length 1000, 2 environments) and share an evaluation protocol: each method re-adapts before every planner call, evaluated over 10 episodes per environment with 5 seeds.

Size	Encoder features	Dynamics features	Critic/Policy/Reward features	Ensemble	LoRA rank	Params. $\lvert\theta_w\rvert$
A	256, 512	512, 512, 512	512, 512	5	16	4,960,618
B	256, 128	384, 384, 128	384, 384	2	16	1,214,651
D	192, 48	192, 192, 48	192, 192	2	16	325,467
E	128, 32	128, 128, 32	128, 128	2	16	161,787
F	64, 32	64, 64, 32	64, 64	2	8	60,667
G	32, 32	32, 32, 32	32, 32	2	4	25,467
H	16, 32	16, 16, 32	16, 16	2	2	11,707
I	8, 32	8, 8, 32	8, 8	2	1	5,787

Table 7: Experiment 3 model-size sweep. Size label C is unused. Params. column is the base world model parameter count $|\theta_w|$ (excludes hypernetwork parameters θ_h), matching the x-axis of Fig. 6. All other hyperparameters follow Tables 2 and 3. Sizes F–I additionally shrink LoRA rank, tied across all five adapted modules; FiLM and Concat baselines use the same feature widths with no rank.

Method	LoRA rank	Adapter params	% of world model	Return
CLAW	1	1,200	2.25%	547.8 ± 41.9
CLAW	2	2,400	4.50%	685.8 ± 35.7
CLAW	4	4,800	9.00%	721.5 ± 54.0
CLAW	8	9,600	17.99%	817.3 ± 35.6
CLAW	16	19,200	35.98%	814.2 ± 86.6
CLAW	32	38,400	71.97%	917.0 ± 67.9
FiLM	—	1,280	2.40%	714.3 ± 44.3
Concat	—	—	—	556.2 ± 64.2

Table 8: Experiment 4 rank sweep. Backbone size is fixed at F (Table 7); rank is swept jointly across all world model parameters (5 seeds each) while all other hyperparameters stay fixed. Return is mean $\pm$ 95% CI across 5 seeds, matching Fig. 7.

Method	Task(s)	Steps	Trainable
CLAW (joint)	Walker + Cheetah	500,000 (250,000/task)	base and hypernetwork
Post-hoc, pass 1	Walker only	250,000	base only (plain TD-MPC2, no hypernetwork)
Post-hoc, pass 2	Cheetah only	250,000	hypernetwork only (base frozen)

Table 9: Experiment 5 post-hoc comparison, at size F (Table 7), rank 8, 5 seeds. CLAW's checkpoints are reused from Experiment 3, evaluated on the Cheetah task only to match the post-hoc baseline's single-task evaluation. Both methods see 250,000 steps of Cheetah data and 500,000 total training steps; the only remaining difference is joint vs. sequential training of the base model and hypernetwork.