Disentangled Multi-Context Meta-Learning: Unlocking Robust and Generalized Task Learning

Seonsoo Kim* **Jun-Gill Kang*** **Taehong Kim** **Seongil Hong**
Agency for Defense Development
{sunsoo3165,jungillkang,kimtaehong07,science4729}@gmail.com
seonsoo-p1.github.io/DMCM

Abstract: In meta-learning and its downstream tasks, many methods rely on implicit adaptation to task variations, where multiple factors are mixed together in a single entangled representation. This makes it difficult to interpret which factors drive performance and can hinder generalization. In this work, we introduce a disentangled multi-context meta-learning framework that explicitly assigns each task factor to a distinct context vector. By decoupling these variations, our approach improves robustness through deeper task understanding and enhances generalization by enabling context vector sharing across tasks with shared factors. We evaluate our approach in two domains. First, on a sinusoidal regression task, our model outperforms baselines on out-of-distribution tasks and generalizes to unseen sine functions by sharing context vectors associated with shared amplitudes or phase shifts. Second, in a quadruped robot locomotion task, we disentangle the robot-specific properties and the characteristics of the terrain in the robot dynamics model. By transferring disentangled context vectors acquired from the dynamics model into reinforcement learning, the resulting policy achieves improved robustness under out-of-distribution conditions, surpassing the baselines that rely on a single unified context. Furthermore, by effectively sharing context, our model enables successful sim-to-real policy transfer to challenging terrains with out-of-distribution robot-specific properties, using just 20 seconds of real data from flat terrain, a result not achievable with single-task adaptation.

Keywords: Meta-Learning, Factors of Variation, Quadruped Robot Locomotion

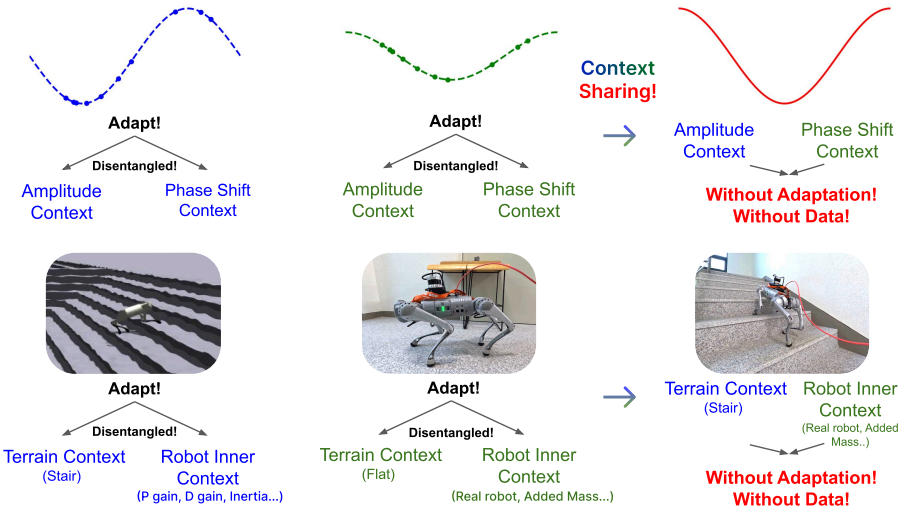


Figure 1: Basic Concept of Disentangled Multi-Context Meta-Learning. Adaptation is illustrated for both the sine regression and robot dynamics tasks. The model adapts by disentangling task-specific factors into separate context vectors. These learned contexts can be reused across tasks with overlapping factors, enabling generalization.

*Equal contribution.

9th Conference on Robot Learning (CoRL 2025), Seoul, Korea.

1 Introduction

Meta-learning, or "learning to learn," was introduced as a general framework for task adaptation in machine learning [1]. In robotics, where variations in physical conditions, environments, and dynamics can significantly affect task execution, meta-learning has emerged as a powerful approach for improving generalization and robustness [2–5].

Conventional Gradient-Based Meta-Learning (GBML) methods [6–10] including MAML [11] typically learn shared parameters for adapting to distinct task features. While effective, these approaches often assume all tasks are equally distinct, which leads to learning entangled representations of different variations. This entanglement can hinder the model's transferability to novel situations [7, 12, 13], and may lead to conflicting gradients during learning [6]. Therefore, for general and efficient adaptation, disentangling task variations is beneficial [14, 15]. Such disentanglement has been explored not only in meta-learning, but also as a broader goal across diverse areas [16–22], where building interpretable, disentangled, and general models is important.

To this end, we propose *Disentangled Multi-Context Meta-Learning* (DMCM), a framework that learns multiple context vectors — each corresponding to a distinct factor of variation. For example, in sine regression, DMCM separates amplitude and phase contexts; in quadruped robot locomotion, it separates contexts for terrain characteristics and robot-specific properties. We build our method on CAVIA [7], but unlike CAVIA, DMCM updates only the disentangled context vector relevant to the changed information.

This disentangled structure offers two key benefits: (1) **improved robustness**, by identifying which type of variation the model is encountering, and (2) **enhanced generalization**, by sharing context vectors across tasks that share factors, without adaptation. Fig. 1 illustrates the main concept of our approach. DMCM is designed to capture distinct context vectors corresponding to different feature subsets within tasks. These context vectors can then be recombined and shared for adapting to novel tasks, where each partial context may not have been previously learned together.

We validate DMCM on sine regression and quadruped robot locomotion. In sine regression, DMCM outperforms MAML, CAVIA, and ANIL [7, 8, 11], showing stable performance under out-of-distribution (OOD) conditions and enables zero-shot prediction via context sharing. We further analyze DMCM on the three-factor sine task with different numbers of contexts, comparing robustness and computational cost (Appendix B). In quadrupedal robot locomotion, policies trained with DMCM-derived contexts outperform both CAVIA-based and vanilla policies under OOD conditions. Notably, DMCM achieves real-world stair climbing with OOD low K_p gain through context sharing, even though the only real-world data used for adaptation was 20 seconds of flat-terrain walking.

Our contributions are two-fold. **(1)** A meta-learning method that learns distinct factors of variation through regulated data sequencing. **(2)** Demonstrating that disentangled contexts enable effective OOD adaptation and zero-shot transfer in sine regression and quadrupedal robot locomotion, including real-world deployment.

2 Preliminaries

2.1 Gradient-Based Meta Learning

Many GBML methods, such as MAML, follow a bi-level optimization scheme with an *inner loop* for task-specific adaptation and an *outer loop* for meta-optimization. Each task $\mathcal{T}_i$ has a dataset split into $\mathcal{D}_i^{\text{train}}$ and $\mathcal{D}_i^{\text{test}}$, used for inner-loop adaptation and outer-loop meta-update, respectively.

Let θ denote the shared parameters and ϕ the task-specific parameters. In CAVIA [7], ϕ is a context parameter concatenated to the model, and only ϕ is updated during the inner loop, where ϕ_i is initialized to zero and α is the inner-loop learning rate:

$$\phi_i \leftarrow \phi_i - \alpha \nabla_\phi \frac{1}{|\mathcal{D}_i^{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_i^{\text{train}}} \mathcal{L}_{\mathcal{T}_i}(f_{\phi_i,\theta}(x), y) \tag{1}$$

The outer loop then updates θ using gradients from the test loss, where β is the meta-learning rate and N is the number of tasks per batch:

$$\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i} \frac{1}{|\mathcal{D}_i^{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_i^{\text{test}}} \mathcal{L}_{\mathcal{T}_i}(f_{\phi_i,\theta}(x), y) \tag{2}$$

Our approach extends CAVIA by introducing K disentangled context vectors $\phi^1, \ldots, \phi^K$, each representing a distinct task factor (e.g., amplitude, phase). These vectors are concatenated at the layers and initialized to zero. During learning, only the vector corresponding to the changed factor is updated, enabling selective, interpretable, and robust task adaptation.

2.2 Reinforcement Learning for Quadrupedal Robot Locomotion

We formulate the quadrupedal robot locomotion problem as a Partially Observable Markov Decision Process (POMDP) defined by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, r, \gamma)$, where $\mathcal{S}$, $\mathcal{O}$, and $\mathcal{A}$ denote the state, observation, and action spaces, respectively. $\mathcal{T}(s_{t+1} \mid s_t, a_t)$ defines the transition dynamics, $r(s_t, a_t)$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor [23]. Due to partial observability, the agent receives observations $o_t \in \mathcal{O}$ rather than full states $s_t \in \mathcal{S}$.

The objective is to learn a policy $\pi^* : \mathcal{O} \to \mathcal{A}$ that maximizes expected return:

$$\pi^* = \arg\max_\pi \mathbb{E}_{s_{t+1}\sim\mathcal{T},\, a_t\sim\pi(o_t)} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right]. \tag{3}$$

To compensate for partial observability, many locomotion agents incorporate latent variables that capture task-relevant factors such as terrain properties and robot-specific properties [24, 25]. These latent representations are often learned via teacher-student frameworks or supervised loss signals.

While some prior works attempt to separate information using multiple encoders or decoders [26, 27], our method explicitly disentangles multiple context vectors based on distinct task variations. This leads to more robust and interpretable adaptation in complex locomotion tasks.

3 Disentangled Multi-Context Meta-Learning (DMCM)

In this section, we provide a detailed description of DMCM, a framework that not only separates context vectors from model parameters but also disentangles the context vectors from each other. Each context vector is explicitly assigned to represent a distinct task factor, guided by the sequence of data provided during training. Pseudocode is shown in Alg. 1.

3.1 Context-Based Factor Separation

DMCM requires an additional task-labeling step to associate each dataset with its underlying context factors. A task $\mathcal{T}_i$ is defined by K context vectors, collectively represented as the task context C_i:

$$\mathcal{T}_i = \mathcal{T}_i(C_i), \quad \text{where } C_i = (c_i^1, \ldots, c_i^K). \tag{4}$$

For instance, in a sine regression task with varying amplitude and phase shift, $K = 2$; c^1 represents amplitude and c^2 represents phase shift.

By explicitly separating these factors, DMCM learns specialized context vectors for each, enabling targeted adaptation and better generalization across combinations of factors.

3.2 Inner Loop Training

The inner loop update builds on CAVIA, but adapts only the relevant context vector. Let s be the selected factor to be trained ($s \in \{1, 2, \ldots, K\}$). At each step, only the context vector ϕ_i^s corresponding to the changing factor c^s is updated, while the others remain unchanged. Specifically, task C_i is chosen to differ from C_{i-1} only in c^s, with the other contexts identical.

3.3 Basic Outer Loop (Meta-gradient step)

The basic outer loop updates the shared model parameters θ after completing B warm-up tasks with their respective inner loops. This allows the context vectors $\{\phi^1, \ldots, \phi^K\}$ to accumulate meaningful information before affecting θ.

Algorithm 1: Disentangled Multi-Context Meta-Learning (DMCM)

Require : Task distribution $p(\mathcal{T})$, conditional distribution $p(\mathcal{T}_i \mid \mathcal{T}_{i-1})$
Require : Inner learning rate α, outer learning rate β, model $f_{\phi_0^1,\ldots,\phi_0^K,\theta}$
Require : Warm-up task number B, Recombination enabled or not
Indexing: For any n, $(s \pm n)$ is taken mod K over $\{1,\ldots,K\}$;
while not done **do**
  $\mathcal{L}_{\text{meta-basic}}$, $\mathcal{L}_{\text{meta-recombination}} \leftarrow 0$;
  for each task $i = 0$ to $N-1$ **do**
    if $i = 0$ **then**
      Sample $\mathcal{T}_0 \sim p(\mathcal{T})$;
    else
      Sample $\mathcal{T}_i \sim p(\mathcal{T}_i \mid \mathcal{T}_{i-1})$;
    Split into $\mathcal{D}_i^{\text{train}}, \mathcal{D}_i^{\text{test}}$;
    Initialize $\phi_i^s \leftarrow 0$;
    for inner-loop step **do**
      $\phi_i^s \leftarrow \phi_i^s - \alpha \nabla_{\phi^s} \frac{1}{|\mathcal{D}_i^{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_i^{\text{train}}} \mathcal{L}_{\mathcal{T}_i}\big(f_{\phi_i^1,\ldots,\phi_i^K,\theta}(x), y\big)$;
    if $i \geq B$ **then**
      $\mathcal{L}_{\text{meta-basic}} \mathrel{+}= \frac{1}{N-B} \frac{1}{|\mathcal{D}_i^{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_i^{\text{test}}} \mathcal{L}_{\mathcal{T}_i}\big(f_{\phi_i^1,\ldots,\phi_i^K,\theta}(x), y\big)$;
    if $i \geq B$ **and** Recombination enabled **then**
      Temporarily load the saved $\bar{\phi} = \left\{\phi_{i-(K+1)}^{s-1}, \phi_{i-2(K+1)}^{s-2}, \ldots, \phi_{i-(K-1)(K+1)}^{s-K+1}\right\}$;
      Sample $\mathcal{D}_j^{\text{test}}$ corresponding to $\bar{\phi}$ and ϕ_i^s;
      $\mathcal{L}_{\text{meta-recombination}} \mathrel{+}= \frac{1}{N-B} \frac{1}{|\mathcal{D}_j^{\text{test}}|} \sum_{(x,y)\in\mathcal{D}_j^{\text{test}}} \mathcal{L}_{\mathcal{T}_j}\big(f_{\phi_i^s,\bar{\phi},\theta}(x), y\big)$;
      Save ϕ_i^{s+1} (i.e., ϕ_{i-K+1}^{s+1}) for the later recombination loop;
    $s \mathrel{+}= 1$
  $\theta \leftarrow \theta - \beta \nabla_\theta\big(\mathcal{L}_{\text{meta-basic}} + \mathcal{L}_{\text{meta-recombination}}\big)$;

3.4 Recombination Loop (Optional)

To support zero-shot generalization when context vectors are shared, DMCM can optionally employ a *recombination loop*. In this procedure, the model parameters θ are updated using context vectors that were not adapted together during the inner loop. This encourages the model to work effectively with independently adapted context vectors. The detailed procedure and its effects are provided in Appendix B.2.

3.5 Adaptation Procedure

At test time, sequential adaptation is done from context c^1 to c^K over S_{adapt} iterations. This hyperparameter S_{adapt} specifies the adaptation depth required for the task. If S_{adapt} is too low, the model may fail to fully adapt to the target task, even if it has the capacity to do so.

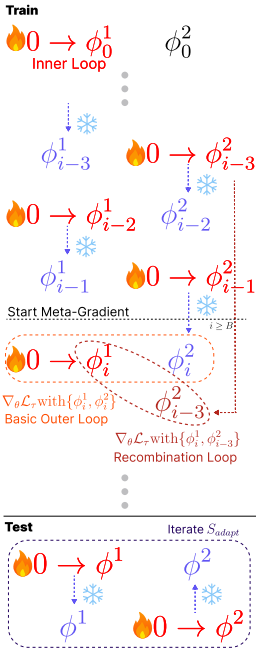


Figure 2: Simple diagram of DMCM for K=2 case

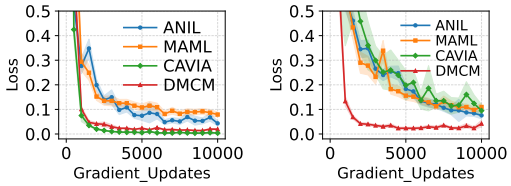
4 Experimental Results

4.1 Sine Task

We evaluate DMCM's effectiveness and the value of disentangled contexts using the standard sine regression benchmark [11]. Full experimental settings are provided in Appendix A.1.

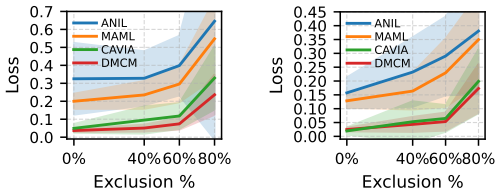
4.1.1 Robustness under OOD Conditions

To assess robustness, we compare DMCM with baselines—MAML, CAVIA, and ANIL [7, 8, 11]—which are basic meta-learning algorithms known to work well at sine tasks.



(a) Full range (100%) **(b)** 40% Data range (60% excluded)

Figure 3: 10-shot sine regression with full data range and with 40% data range used for training. Evaluation is performed on the full distribution. The shaded area shows a 95% confidence interval. (a) 100% of the range. (b) One case of 40% range.



(a) 2000 Meta-gradients **(b)** 4000 Meta-gradients

Figure 4: Average loss comparisons with range exclusion. (a) At 2000 meta gradients (near convergence for CAVIA/DMCM on full range), (b) At 4000 meta gradients (CAVIA/DMCM fully converged on full range). The shaded region shows standard deviation. 30 randomly selected ranges were used for evaluation.

We partition the amplitude and phase shift into five intervals, creating 25 range combinations. We then simulate out-of-distribution (OOD) conditions by excluding 10 combinations (40% missing), 15 combinations (60% missing), and 20 combinations (80% missing) during training.

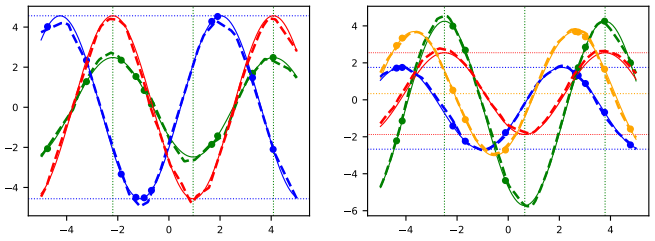
Each method is evaluated under identical conditions (10 inner steps, same learning rate) for both 5-shot and 10-shot settings. 5-shot results are attached in Appendix A.2.

As shown in Fig. 3a, CAVIA performs the best with full data, but its performance generally becomes unstable under exclusion of data as in Fig. 3b. In contrast, DMCM achieves consistently lower loss and smaller variance, especially in high-exclusion settings. In Fig. 4, DMCM shows robustness for data exclusion at both 2000 and 4000 meta-gradient steps. This robustness stems from disentangling task factors, reducing sensitivity to missing combinations.

4.1.2 Zero-Shot Generalization via Recombination

We assess DMCM's effectiveness in assigning each factor of variation to the correct context vector and achieving zero-shot generalization to unseen tasks by sharing previously learned context vectors without further adaptation.

Fig. 5 visualizes predictions of unseen sine functions (red) using context vectors reused from other adaptations that share the same factors. This demonstrates DMCM's ability to recombine contexts **without inner-loop adaptation**. Such capability not only enables zero-shot transfer but also allows selectively adapting or extracting specific context information, broadly applicable in diverse scenarios. Experimental settings are provided in Appendix A.3 (two context) and B.2 (three context).



(a) Two contexts: Amplitude (blue), Phase-Shift (green) **(b)** Three contexts: Amplitude (blue), Phase-Shift (green), Y-Shift (orange)

Figure 5: Zero-shot prediction with disentangled context vectors. Dotted lines (green, blue, orange) show predictions after adaptation to the displayed points, while the solid line denotes the ground truth. Red dotted lines indicate predictions using only the shared context vectors **without adaptation**.

4.1.3 Number of Contexts for DMCM

Tasks often involve multiple factors of variation, making the choice of context count critical for DMCM. Our experiments show that aligning the number of context vectors with the true factors of variation improves robustness, while using too many contexts slows adaptation. Thus, balancing robustness and efficiency is essential when deciding how many contexts to disentangle. Detailed results about the three-factor sine task, are provided in Appendix B.

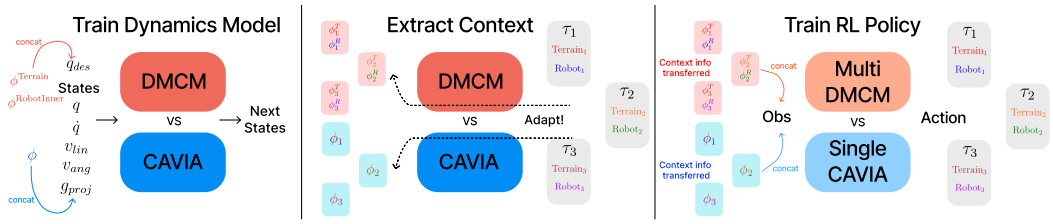


Figure 6: Learning procedure at Quadrupedal Robot Locomotion Task

4.2 Quadrupedal Robot Locomotion Task

In this section, we first train the robot dynamics model using both CAVIA and DMCM, then extract their latent features—context parameters (CAVIA) and context vectors (DMCM)—across diverse simulation conditions. These extracted contexts are subsequently transferred as prior knowledge in RL, illustrated in Fig. 6. We compare three RL policies under identical settings, differing only in the use of context information: a vanilla policy without context, a single-CAVIA policy using unified context parameters, and a multi-DMCM policy using disentangled context vectors (We set K=2).

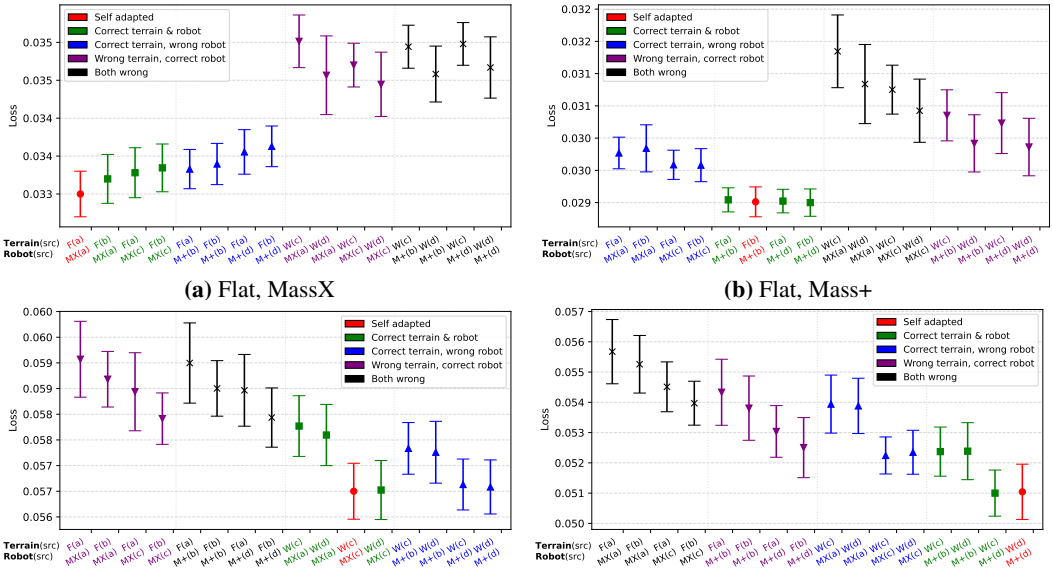
4.2.1 Go1 Dynamics Model

We collect training data in simulation using a pretrained naive walking policy, whose performance is substantially worse than that of the baseline vanilla policy (Appendix D.2). This ensures that context extraction is not biased by high-performing behaviors. A dynamics model is then trained to predict joint positions (q), velocities ($\dot{q}$), projected gravity, linear velocities, and angular velocities from the robot state and desired joint positions (q_{des}) 0.02 seconds earlier.

Tasks are defined along two axes: *terrain type* (44 variations, including flat, wavy, sloped, stair, and random terrains) and *robot-specific properties* (K_p, K_d, payload, center of mass, delay, torque limits; full ranges in Appendix C.1). This yields 44×350 training tasks from Isaac Gym simulator [28].

We train dynamics model using DMCM and CAVIA with matched hyperparameters and total context parameter counts, resulting in comparable prediction performance (see Appendix C for details).

We evaluate the disentanglement capabilities of DMCM, trained only on simulation data, across four real-world datasets: (a) Flat terrain, (b) Flat terrain with 1.5 kg payload, (c) Wavy terrain, and



(a) Flat, MassX **(b)** Flat, Mass+

(c) Wavy, MassX **(d)** Wavy, Mass+

Figure 7: Each subplot shows the average loss for each dataset using 16 context vector sets derived from four real-world datasets. Each set includes a terrain (Flat (F) or Wavy (W)) and a robot property context (Additional Mass (M+) or No Additional Mass (MX)). Results are averaged over 10 trials, with adaptation and test sets randomly sampled from 20 seconds within the 40 seconds of data. Error bars indicate the standard deviation.

(d) Wavy terrain with 1.5 kg payload. Adaptation to these datasets yields terrain and robot-specific context vectors, allowing 16 possible context vector combinations for evaluation.

As shown in Fig. 7, models evaluated with context vectors aligned to the test data factor (red and green) generally achieve better performance than those using unrelated contexts. Notably, in some cases the shared context vector (green) even outperforms the self-adapted context vector (red). These results demonstrate that the disentanglement learned in simulation is transferred meaningfully to real-world dynamics.

4.2.2 Go1 Reinforcement Learning with Contexts

We train RL policies with the extracted context parameters (CAVIA) and context vectors (DMCM), collected from all terrains and 500 robot-specific property settings, and concatenated them with observations during training. We evaluate the trained multi-DMCM, single-CAVIA and, vanilla policies in two complementary simulation experiments and real-world experiment.

General OOD Evaluation For evaluation, three environments are conducted: (1) The in-distribution terrain of steepest stair climbing, (2) OOD terrains with unseen height variations, (3) OOD robot-specific properties at the same terrain as (1).

For OOD robot property conditions, we apply a high payload (5–6 kg) and significantly reduced K_p gains (ΔK_p between –5 and –4 N $\cdot$ m), beyond the training ranges ($\leq$ 4 kg payload and $\Delta K_p \geq -2.5\,\mathrm{N}\cdot\mathrm{m}$). Full experimental details are provided in Appendix D.3 and D.4

Metric	Multi-DMCM	Single-CAVIA	Vanilla
Inner Distribution			
Success Counter	1892/2000	**1905/2000**	1580/2000
RMSE Linear Vel (m/s)	0.0816	**0.0798**	0.1552
Reward (w/o Termination Reward)	21.9682	**22.1789**	17.4848
Lifespan	0.9851	**0.9852**	0.9190
AvgDist (Success)(m)	8.4581	**8.5203**	7.3806
OOD Terrain			
Success Counter	**281/2000**	149/2000	205/2000
RMSE Linear Vel (m/s)	**0.2486**	0.2844	0.2555
Reward (w/o Termination Reward)	**10.0295**	7.5442	8.5807
Lifespan	**0.5476**	0.4530	0.4737
OOD Robot-Specific Properties			
Success Counter	**602/2000**	500/2000	48/2000
RMSE Linear Vel (m/s)	**0.0944**	0.1287	~~0.0826~~
Reward (w/o Termination Reward)	**19.2283**	17.8395	~~19.2308~~
Lifespan	**0.6009**	0.5844	0.3050
AvgDist (Success)(m)	**7.9205**	6.0237	6.0990

Table 1: Comparison of three policies across inner distribution, OOD terrain, and OOD robot properties (based on 2000 trials) at simulation (Isaac Gym). Average distance is omitted for the OOD terrain due to the wide and variable range of linear velocity commands. For the OOD robot-specific properties case, the RMSE of linear velocity and reward values for the vanilla policy are not meaningful, as the robot mostly failed before climbing the stairs and only moved across flat regions.

As shown in Table 1, both the single-CAVIA and multi-DMCM policies outperform the vanilla baseline across all evaluation metrics under in-distribution conditions, showing that the context learned from the dynamics model is effective. Under OOD settings, the multi-DMCM model consistently achieves the best performance, thereby demonstrating robustness under unseen conditions.

Gradual Payload Variation To assess robustness more systematically, we evaluate all policies in Isaac Gym on a 5 cm stair-climbing task under increasing payloads. As shown in Table 2, all policies perform well under in-distribution payloads (0–3 kg). However, at OOD payloads (6 kg and 9 kg), the vanilla and single-CAVIA policies experience a sharp performance drop, while the multi-DMCM policy maintains strong performance, confirming its robustness to unseen physical variations.

The single-CAVIA policy occasionally underperforms the vanilla policy in the two simulation experiments. This observation suggests that reliance on a single, unified context can induce overfitting and hinder robust application to unseen conditions, whereas disentangled multi-DMCM policy retains robustness. Additional simulation results of verifying context usage are attached at Appendix D.6.

Metric	Method	0 kg	3 kg	6 kg (OOD)	9 kg (OOD)
Lifespan	Vanilla	**0.998**	0.996	0.986	0.897
	Single-CAVIA	0.996	0.988	0.942	0.745
	Multi-DMCM	0.998	**0.996**	**0.987**	**0.953**
Reward	Vanilla	25.38	24.98	23.56	21.33
	Single-CAVIA	24.60	24.32	23.00	19.61
	Multi-DMCM	**25.51**	**25.09**	**24.27**	**22.62**

Table 2: Performance comparison across gradual payload variations

Real-world Deployment. Finally, we evaluate the RL policies on the Go1 robot in real-world conditions. All models perform reasonably when provided with appropriate context parameters or vectors. To further assess robustness, we deploy the three models under out-of-distribution (OOD) conditions by reducing the robot's K_p gains from 20 N · m to 16 N · m and adding a 1.5 kg payload while climbing 17 cm stairs, settings that were not encountered during training.

For evaluation, two pre-collected datasets are used as context: (i) simulation stair climbing data with 16 N · m K_p, and (ii) real flat-terrain data with 16 N · m K_p and an additional payload. In the multi-DMCM policy, the terrain context was derived from (i) and the robot-specific context from (ii). In contrast, the single-CAVIA policy was evaluated separately with each dataset because CAVIA cannot disentangle the two sources of variation.

Metric	Multi-DMCM	Single (Sim Stair)	Single (Real Flat)	Vanilla
Success Rate (%)	**80**	0	0	40
Avg. Time to Success (s)	**11.15**	NaN	NaN	18.61

Table 3: Success rate and average time to success in real-world OOD setting (5 trials, 17 cm stairs, payload = 1.5 kg, $\Delta K_p = -4\,\mathrm{N \cdot m}$).

Using these contexts, the multi-DMCM policy achieves the best performance in climbing steep stairs at OOD conditions, leveraging its inherent robustness, whereas the single-CAVIA policy fails completely. Moreover, by sharing contexts, DMCM enables successful adaptation to stair conditions without any real-world stair data, relying only on 20 seconds of flat-terrain walking. Additional deployment results for multi-DMCM policy, along with detailed experimental settings, are provided in Appendix D.7.

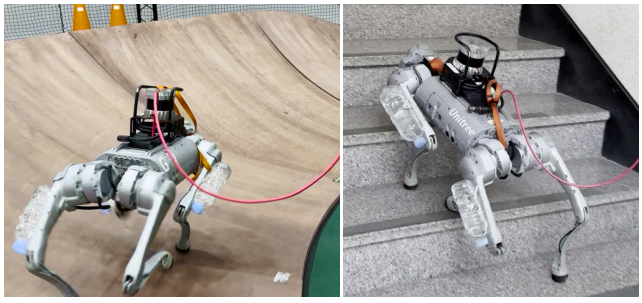


Figure 8: Additional deployment results showing the multi-DMCM policy on wavy terrain (left) and stair climbing (right) under asymmetric payloads with water bottles and a lidar sensor.

5 Conclusion

We present Disentangled Multi-Context Meta-Learning (DMCM), a framework that learns multiple context vectors, each corresponding to a distinct factor of variation within tasks. Unlike prior approaches that rely on entangled or unified task representations, DMCM supports selective adaptation by updating only the relevant context vector. This design improves robustness by identifying which aspect of a task has changed, and enhances generalization by enabling context vector sharing across tasks with partially overlapping features.

Through experiments on sinusoidal regression and quadruped robot locomotion, we demonstrate DMCM's robustness and generalization, including zero-shot adaptation to novel situations using prior experience.

Our work highlights the importance of disentangled context representations, offering a step toward interpretable, robust, and generalizable task adaptation in robotics and beyond.

6 Limitations

While DMCM improves robustness and generalization, several limitations remain:

- **Context labeling**: The current training process allows selecting the contexts to be learned, but it relies on manually defined, meaningful context labels with sufficient variation. Automating context discovery while preserving robustness and generalization is an important direction for future work.
- **Task generality**: DMCM model itself has so far been validated only on regression tasks. Its extension to classification, reinforcement learning, and other domains remains to be explored.
- **Real-time adaptation**: Although DMCM improves adaptability in quadruped robot locomotion tasks, our experiments relied on pre-collected data for adaptation. Incorporating meta-learning's fast adaptation capability with selective context updates could enable robust real-time deployment in future settings.

Future work will therefore focus on automatic context discovery, broadening applicability across domains, and integrating selective real-time adaptation to further enhance DMCM's utility in real-world robotics and beyond areas.

7 Acknowledgments

This work was supported by the Agency for Defense Development grant funded by the Korean Government in 2025.

References

[1] S. Thrun and L. Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pages 3–17. Springer, 1998. 2

[2] C. Finn, T. Yu, T. Zhang, P. Abbeel, and S. Levine. One-shot visual imitation learning via meta-learning. In *Conference on robot learning*, pages 357–368. PMLR, 2017. 2

[3] R. Kaushik, T. Anne, and J.-B. Mouret. Fast online adaptation in robotics through meta-learning embeddings of simulated priors. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5269–5276. IEEE, 2020.

[4] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[5] M. O'Connell, G. Shi, X. Shi, K. Azizzadenesheli, A. Anandkumar, Y. Yue, and S.-J. Chung. Neural-fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7(66): eabm6597, 2022. 2

[6] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018. 2

[7] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *International conference on machine learning*, pages 7693–7702. PMLR, 2019. 2, 4, 12

[8] A. Raghu, M. Raghu, S. Bengio, and O. Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019. 2, 4

[9] Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[10] A. Nichol and J. Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2(3):4, 2018. 2

[11] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017. 2, 4

[12] F. Alet, T. Lozano-Pérez, and L. P. Kaelbling. Modular meta-learning. In *Conference on robot learning*, pages 856–868. PMLR, 2018. 2

[13] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019. 2

[14] H. Yao, Y. Wei, J. Huang, and Z. Li. Hierarchically structured meta-learning. In *International conference on machine learning*, pages 7045–7054. PMLR, 2019. 2

[15] R. Vuorio, S.-H. Sun, H. Hu, and J. J. Lim. Multimodal model-agnostic meta-learning via task-aware modulation. *Advances in neural information processing systems*, 32, 2019. 2

[16] M. W. Gondal, M. Wuthrich, D. Miladinovic, F. Locatello, M. Breidt, V. Volchkov, J. Akpo, O. Bachem, B. Schölkopf, and S. Bauer. On the transfer of inductive bias from simulation to the real world: a new disentanglement dataset. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[17] K. Hsu, J. I. Hamid, K. Burns, C. Finn, and J. Wu. Tripod: Three complementary inductive biases for disentangled representation learning. *arXiv preprint arXiv:2404.10282*, 2024.

[18] M. F. Mathieu, J. J. Zhao, J. Zhao, A. Ramesh, P. Sprechmann, and Y. LeCun. Disentangling factors of variation in deep representation using adversarial training. *Advances in neural information processing systems*, 29, 2016.

[19] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

[20] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. *Advances in neural information processing systems*, 26, 2013.

[21] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*, 2017.

[22] I. Higgins, A. Pal, A. Rusu, L. Matthey, C. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International conference on machine learning*, pages 1480–1490. PMLR, 2017. 2

[23] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998. 3

[24] A. Kumar, Z. Fu, D. Pathak, and J. Malik. Rma: Rapid motor adaptation for legged robots. *arXiv preprint arXiv:2107.04034*, 2021. 3

[25] I. M. A. Nahrendra, B. Yu, and H. Myung. Dreamwaq: Learning robust quadrupedal locomotion with implicit terrain imagination via deep reinforcement learning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5078–5084. IEEE, 2023. 3

[26] P. Wu, W. Xie, J. Cao, H. Lai, and W. Zhang. Loopsr: Looping sim-and-real for lifelong policy adaptation of legged robots. *arXiv preprint arXiv:2409.17992*, 2024. 3

[27] Z. Xiao, X. Zhang, X. Zhou, and Q. Zhang. Pa-loco: Learning perturbation-adaptive locomotion for quadruped robots. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9110–9115. IEEE, 2024. 3

[28] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021. 6, 16, 18

[29] J. Hwangbo, J. Lee, and M. Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018. 18

[30] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 18

[31] G. Ji, J. Mun, H. Kim, and J. Hwangbo. Concurrent training of a control policy and a state estimator for dynamic and robust legged locomotion. *IEEE Robotics and Automation Letters*, 7(2):4630–4637, 2022. 19

[32] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. *Conference on Robot Learning (PMLR)*, pages 91–100, 2022. 19

[33] G. B. Margolis and P. Agrawal. Walk these ways: Tuning robot control for generalization with multiplicity of behavior. *Conference on Robot Learning (PMLR)*, pages 22–31, 2023. 19

Appendix

A Sine Task

A.1 Hyperparameter and Experiment Settings

To ensure a fair comparison across MAML, ANIL, CAVIA, and DMCM, we align the hyperparameters as closely as possible. All models use a meta learning rate of 0.001. The inner-loop learning rate is 0.1 for ANIL, CAVIA, and DMCM, but reduced to 0.001 for MAML, as higher values cause instability. Each algorithm applies 10 inner-loop updates.

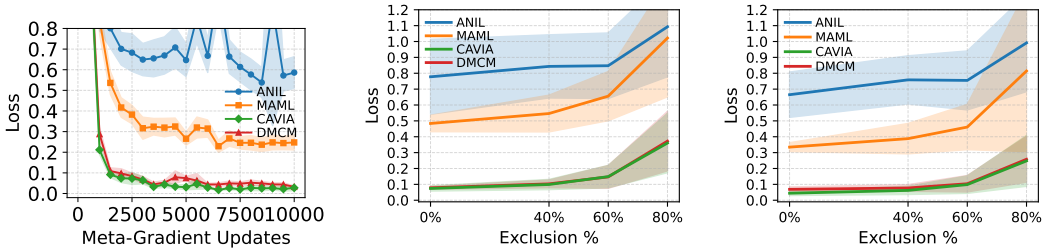
Following common practice in sine regression tasks, the amplitude is sampled from $[0.1, 5.0]$ and the phase shift from $[0, \pi]$ [7]. For the 10-shot case, 10 data points are provided for inner-loop training, meta-training, and evaluation, while in the 5-shot case, 5 data points are provided for each. For out-of-distribution evaluation, both the amplitude and phase shift ranges are uniformly divided into five intervals, resulting in a total of 25 range combinations for systematic exclusion.

All models employ a neural network with two hidden layers of 40 units each. Each meta-update uses a batch of 25 tasks. Evaluation is conducted on the same set of 500 sampled tasks, using mean squared error (MSE) computed over 100 test points.

For context dimensions, CAVIA uses 6 context parameters, while DMCM uses 3 parameters for each disentangled context vector. In DMCM, 10 warm-up iterations are performed before starting the outer loop ($B = 10$), and 10 inner-loop iterations are applied for adaptation during evaluation ($S_{\text{adapt}} = 10$). Although all four algorithms use the same number of meta-gradient updates, DMCM applies additional inner-loop sequences before the warm-up stage. The recombination loop is not applied in DMCM.

A.2 5shot Result

Same as 10 shot cases, we compare MAML, ANIL, CAVIA, and DMCM for 5shot cases. We sampled 30 different range selections same as 10 shot cases.



(a) Full data at 5shot. Shaded ranges are 95%confidence range

(b) Average loss comparisons with range exclusions at 2000 meta-gradient steps. Shaded ranges are standard deviation.

(c) Average loss comparisons with range exclusions at 4000 meta-gradient steps. Shaded ranges are standard deviation.

Figure 9: Loss curves for (a) No exclusion, (b) average loss comparison with range exclusions at 2000 meta-gradients, and (c) 4000 meta-gradients

Unlike the results in the 10-shot cases shown in Fig. 3 and Fig. 4, DMCM and CAVIA exhibit similar performance under the same exclusion setting, as shown in Fig.9. We hypothesize that clear disentanglement requires sufficient data to capture the underlying task factors. Nevertheless, both CAVIA and DMCM outperform the other methods.

A.3 Analysis of Zero Shot Prediction

As illustrated in Fig. 5, the trained DMCM model successfully predicts sine functions without direct access to task data, relying solely on shared context vectors. The model uses 3 parameters for each

context vector, an inner-loop learning rate of 0.08 with a decay factor of 0.92 applied at each step, and an outer-loop learning rate of 0.001. It performs 30 inner-loop steps, begins meta-gradient updates after 10 tasks ($B = 10$), and employs a neural network with two hidden layers of 40 units each. Each meta-update aggregates over 25 tasks, and evaluation is conducted using a 3-step iterative adaptation ($S_{\text{adapt}} = 3$). Two context vectors are concatenated with the input. In addition, the recombination loop is applied.

In our experiments, recombination loss, reflecting zero-shot prediction using shared contexts, is generally higher than self-adaptation loss, which involves direct adaptation using task specific data. This trend holds in sine regression as well, shown at Fig. 10. However, we find that the important contributor to the higher recombination loss is the difficulty of extracting meaningful phase context from low-amplitude sine waves.

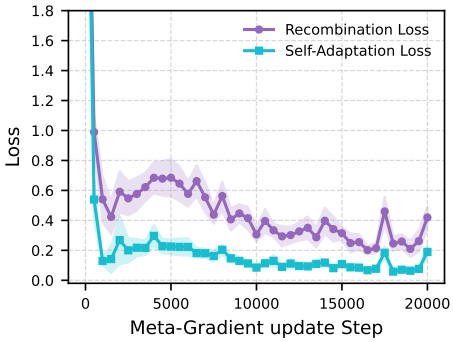


Figure 10: Learning of Sine Task with DMCM with recombination loop.

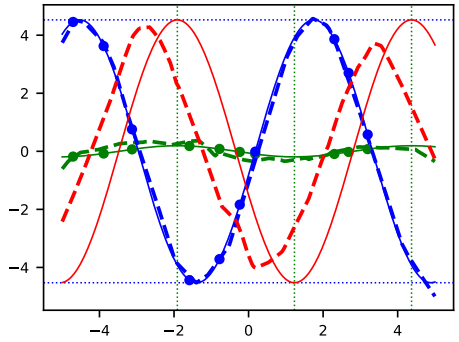


Figure 11: Example of an incorrect zero-shot prediction when the phase-shift context is derived from a low-amplitude case (green). The zero-shot prediction (red dot) deviates significantly from the ground truth (red solid line)

As shown in Fig. 11, the DMCM model struggles to infer the phase shift context when the amplitude is very low. To examine this effect, we compare three cases: (i) self-adaptation loss, (ii) recombination (zero-shot) loss over the full amplitude range, and (iii) recombination loss restricted to the amplitude range $[1.5, 5.0]$.

Method	**Mean Loss**	**95% Confidence**
Self Adapted	0.0578	$\pm$ 0.0051
Recombination (Zero-Shot, Full Range)	0.1876	$\pm$ 0.0416
Recombination (Zero-Shot, Amp [1.5, 5.0])	0.0893	$\pm$ 0.0090

Table 3: Mean loss and 95% confidence intervals. 500 tasks are randomly sampled for evaluation.

As shown in Table 3, the recombination loss is significantly reduced—from 0.1876 to 0.0893—when restricting the amplitude range to $[1.5, 5.0]$. This 52.4% reduction suggests that high recombination loss in the full-range case is primarily due to the shared context vector being derived from inputs with insufficient information (i.e., very low amplitudes). These results support the hypothesis that zero-shot prediction becomes unreliable when the context vector is formed from data that lacks enough information to represent the underlying task.

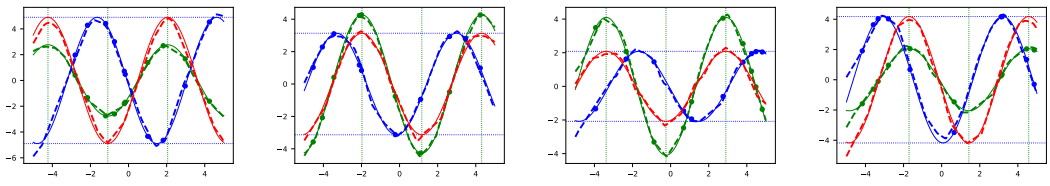


Figure 12: Additional zero-shot results from DMCM with two contexts. Zero-shot predictions (red dots) are obtained through shared context vectors. Tasks used for adapting amplitude and phase-shift contexts are shown in blue and green, respectively.

A.4 Mislabeled Contexts

We evaluate the effect of mislabeled task data on the sine regression task. Hyperparameters and training settings follow Appendix A.1, and evaluation is conducted after 6000 meta-gradient steps. As shown in Table 4, both DMCM and CAVIA show similar degradation as the proportion of mislabeled data increases. Here, "10% mislabeling" means that during task sampling, each task has a 10% probability of being assigned an incorrect task label.

Model	Clean	10% mislabeled	20% mislabeled	Increase (%)
DMCM	0.0225	0.0476	0.0631	+111% / +180%
CAVIA	0.0135	0.0267	0.0467	+98% / +246%

Table 4: Effect of mislabeled context data on sine regression tasks. Percentage increase is relative to the clean baseline.

B Additional Analysis of DMCM: N Context Sine

To explore DMCM's scalability to multiple task factors and further analysis, we extend our evaluation to a setting with three factors of variation: (i) amplitude, (ii) phase shift, and (iii) y-shift(vertical offset).

Each sine function is modeled as:

$$y = A \cdot \sin(x - \phi) + b, \tag{5}$$

where Amplitude $A \in [0.1,\ 5.0]$, Phase shift $\phi \in [0,\ \pi]$, Y-shift $b \in [-2.0,\ 2.0]$.

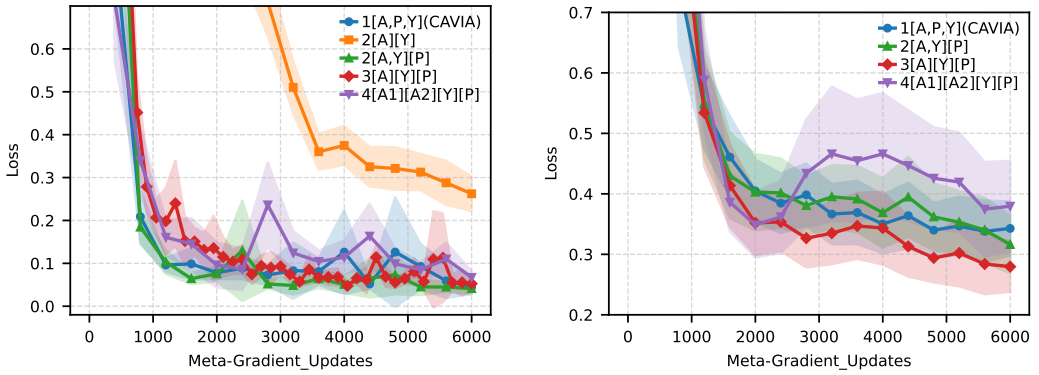
B.1 Robustness with Varying Numbers of Contexts

We compare models with 1 to 4 context vectors on sine regression tasks involving amplitude, phase shift, and y-shift variations. For the 2-context case, we test (i) a variant with amplitude and y-shift only (phase missing), and (ii) a variant combining amplitude and y-shift in one context and phase shift in another. For the 4-context setting, we introduce redundancy by assigning two separate context vectors to the amplitude term, resulting in $(A_1 \cdot \sin(x - \phi) + A_2 \cdot \sin(x - \phi) + b)$.

Hyperparameters and Experimental Settings. Models are evaluated under full-range and exclusion-range training. All experiments use an inner-loop learning rate of 0.05 with a decay factor of 0.92 per step, and an outer-loop (meta) learning rate of 0.00033. Each model is trained for 20 inner-loop steps using a fully connected network with four hidden layers of 40 units each. We sample 45 tasks per meta-update. For DMCM, meta-gradient updates begin after $B = 20$ warm-up tasks. During evaluation, we use $S_{\text{adapt}} = 5$ iterative loops for the full-range setting and $S_{\text{adapt}} = 10$ loops for the OOD setting. Recombination loop is not applied.

Context Configurations. We evaluate the following context configurations:

- **1[A,P,Y] (CAVIA)**: A single context encodes amplitude, phase, and y-shift jointly (9 parameters), concatenated with input.
- **2[A][Y] (Phase shift missing)**: Two contexts (4 parameters each), encoding amplitude and y-shift separately, without phase-shift context. Both concatenated with input.
- **2[A,Y][P]**: Two contexts (4 parameters each), where one encodes amplitude and y-shift jointly, and the other encodes phase shift. Both concatenated with input.
- **3[A][Y][P]**: Three contexts (3 parameters each), encoding amplitude, y-shift, and phase shift separately. Amplitude context is concatenated at the first layer, y-shift at the second layer, and phase shift with input.
- **4[A1][A2][Y][P]**: Four contexts (2 parameters each), where amplitude is split into two contexts (A_1, A_2) with ranges $[0.05, 2.5]$, plus y-shift and phase-shift contexts. A_1 and A_2 are concatenated at the first layer, y-shift at the second layer, and phase shift with input.



(a) Full-range training and evaluation.

(b) OOD evaluation over 40 training range samples.

Figure 13: Training curves of N-context models under (a) full-range and (b) OOD evaluation. Shaded areas represent confidence intervals: 95% for (a) and 0.25 standard deviation for (b). The two-context case without the phase-shift factor is excluded in (b) due to extremely poor performance.

In the full-range setting, all models perform well unless a critical variation is excluded from the contexts (the two-context model without phase shift). For OOD evaluation, each factor range is divided into four intervals, with only two intervals used for training. This corresponds to 12.5% of the total range being covered during training, except in the four-context case (where each amplitude subrange $[0.05, 2.5]$ is divided and sampled separately). Under these OOD conditions, DMCM with the exact number of variation-specific contexts (3) achieves the best performance, consistently yielding low loss. By contrast, adding redundant context makes learning less stable and increases variability during training, as seen in the 4-context model.

B.2 N-Context Recombination

Although Fig. 2 illustrates only the $K = 2$ case, our method naturally extends to N contexts. The core idea of the recombination loop is to update the model using context vectors that were not jointly adapted in the inner loop. This is achieved by saving $K-1$ context vectors for each of the K contexts and reusing them during training.

For example, consider an adaptation sequence of {y-shift, amplitude, phase shift}. When updating the y-shift context at iteration i, we require:

- the phase-shift context vector adapted at $i - (K + 1) = i - 4$, and
- the amplitude context vector adapted at $i - 2(K + 1) = i - 8$,

ensuring that none of these were adapted together in the same inner loop.

This mechanism requires $(K - 1) \times K$ additional context vectors in memory; for $K = 3$, this corresponds to six extra context vectors. Without the recombination loop, zero-shot prediction fails; in our experiments, the recombination loss increased from 0.069 (with loop, restricted amplitude range) to 3.66 (without loop, restricted amplitude range). While DMCM still provides robustness without the recombination loop, zero-shot recombination is only possible when the loop is applied. However, introducing it slightly reduces self-adaptation performance. Learning settings follow Appendix B.1, with recombination loop and amplitude restrictions as in Appendix A.3.

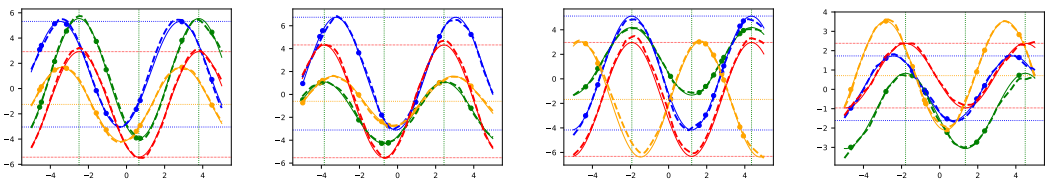


Figure 14: Additional zero-shot results from DMCM with three contexts. Zero-shot predictions (red dots) are obtained through shared context vectors. Tasks used for adapting amplitude, phase-shift, and y-shift contexts are shown in blue, green, and orange, respectively.

B.3 Computation of N Context Cases

In Table 5, we compare the training time of MAML, CAVIA, and DMCM on the sine task with varying numbers of context vectors listed at Appendix B.1. For DMCM, training time per meta-gradient decreases as the number of context vectors increases, due to smaller update sizes from fewer parameters per vector. However, adaptation time grows with more contexts, since DMCM adapts each context separately and requires S_{adapt} iterations for all contexts to interact sufficiently. In this work, we set $S_{\text{adapt}} = 5$, though this value may vary depending on the task.

	MAML	CAVIA (1)	DMCM (2)	DMCM (3)	DMCM (4)
Train Time (400 meta-grad)	679	428	350	331	**317**
Adapt Time (300 sine)	15.30	**4.01**	12.04	18.38	25.20

Table 5: Training and adaptation time (seconds) at RTX 3080Ti GPU with 11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz CPU.

B.4 Number of Parameter Variation at Context Vector

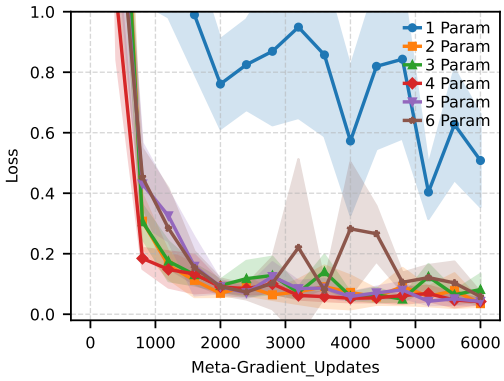


Figure 15: Training curves for the 2-context model (one entangled amplitude–phase context and one y-shift context) with different numbers of parameters for one context vector. Shaded regions indicate the 95% confidence interval.

We evaluate the effect of varying the number of parameters in each context vector using a 2-context model (one entangled amplitude–phase context and one y-shift context). As shown in Fig. 15, a context vector with only a single parameter fails to capture the task variations, as expected, since the entangled context must simultaneously encode amplitude and phase shift, requiring at least two parameters. On the other hand, the relatively unstable learning and large deviation observed in the 6-parameter case suggest that overly large context vectors may reduce training stability and hinder generalization. Overall, these results highlight the importance of selecting an appropriate parameter dimension to balance expressiveness and stability.

C Go1 Dynamics Model

C.1 Hyperparameter and Experiment Settings

For the Go1 dynamics modeling task, we train DMCM and CAVIA under closely aligned hyperparameters to ensure a fair comparison. Both models use an inner-loop learning rate of 0.1, a meta-learning rate of 0.001, and a fully connected network with four hidden layers of 512, 256, 128, 64 units, each with ReLU activations. The meta-batch size is set to 25 tasks per update, and each model employs a total of 40 context parameters (20 per context vector). We apply a warm-up of $B = 10$ iterations before meta-updates and perform $S_{\text{adapt}} = 10$ inner-loop updates for adaptation. The recombination loop is also applied.

Data Collection and Prediction Target. We collected training data using the Isaac Gym simulator [28] and a naive walking policy explained at section D.2. The model learns to predict 33-

dimensional outputs, including joint positions (q), joint velocities ($\dot{q}$), projected gravity, linear velocity, and angular velocity, from 45-dimensional input. The input consists of the robot's state and desired joint position (q_{des}) from the timestep 0.02 seconds earlier.

Terrain Variation. We train across 44 types of terrains, including flat, wavy, sloped, and stair (both ascending and descending) terrains, as well as randomly structured terrains. Terrain diversity is further increased by randomizing friction coefficients and difficulty levels.

Robot Property Variation. Robot-specific variations are introduced through domain randomization, summarized in Table 6. The randomized parameters include mass, center of mass (COM), action delay, control gains, and torque limits. Each property is perturbed independently across training tasks to enhance robustness.

Property	Min	Max
Mass offset (kg)	-1.0	$+4.0$
Center of Mass shift (m)	-0.05	$+0.05$
Action delay (s)	0.0	0.02
K_p gain offset (N · m)	-4.5	$+4.5$
K_d gain offset (N·m·s/rad)	-0.2	$+0.2$
Torque limit offset (N · m)	-4.0	$+4.0$

Table 6: Randomized robot-specific properties and their value ranges used during training.

Default K_p gain is set 20N·m, default K_d gain is set 0.5N·m·s/rad, default torque limit is set 30N·m.

Task Composition. We construct a dataset of $44 \times 350 = 15{,}400$ tasks, each corresponding to a unique combination of terrain and robot-specific properties. Each task contains 2,000 samples, equivalent to 40 seconds of walking. For inner-loop training, outer-loop updates, and evaluation, we randomly sample 700 data points per task.

C.2 Dynamics Model Training

We train both CAVIA and DMCM dynamics models using the same dataset and aligned hyperparameters. All inputs and outputs are normalized by subtracting the mean and dividing by the standard deviation computed from the total training set. As shown in Fig. 16, both models achieve comparable performance. Additionally, DMCM demonstrates reasonable performance under zero-shot conditions (recombination loss), using shared context vectors without access to task-specific data.

As shown in Fig. 16, the DMCM model exhibits a steadily decreasing loss, indicating improved understanding of dynamics over the course of training. However, to ensure that the model truly benefits from task-specific adaptation, we must verify that it is not simply relying on shared parameters

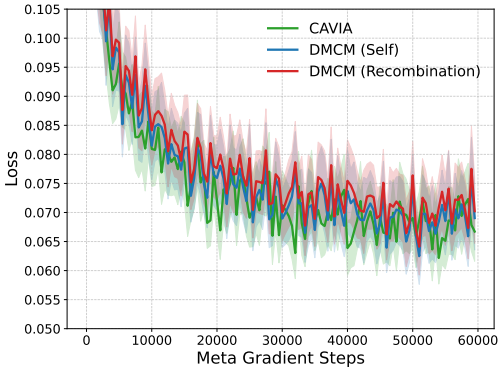


Figure 16: Dynamics model loss for CAVIA, DMCM with self-adaptation, and DMCM with recombination (zero-shot with shared context vectors). Shaded region indicates 95% confidence interval. Evaluation is performed on 500 tasks.

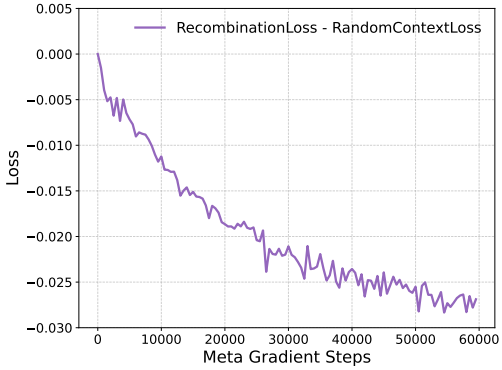


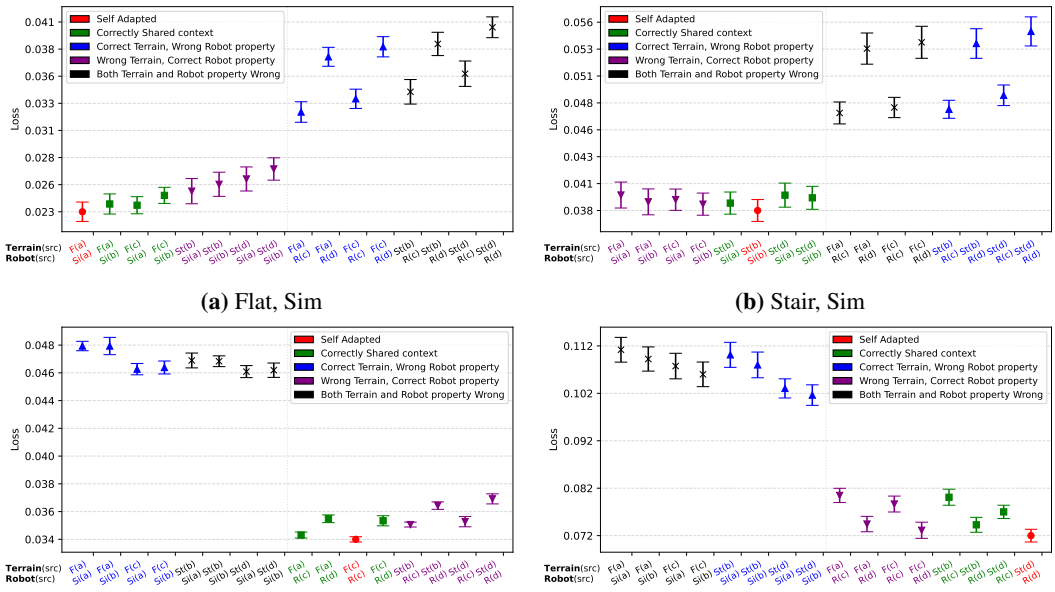
Figure 17: Loss gap between proper context sharing and randomly assigned context at DMCM. Negative values imply meaningful adaptation.

(θ) that happen to generalize across all tasks. To assess this, we evaluate the model using randomly assigned context vectors that are not aligned with the target task. Fig. 17 shows the gap between recombination loss and random-context loss increases during DMCM training, indicating that the model is learning to encode useful, task-specific information.

C.3 Additional Result:Sim Real Difference

To evaluate DMCM's ability to capture sim-to-real discrepancies, we conduct an additional experiment comparing simulated and real-world dynamics data. Real-world trajectories are collected on both flat and stair terrains using the Go1 robot, while simulation data are generated on corresponding flat and stair terrains using RaiSim [29].

For a controlled comparison, we do not apply domain randomization, and robot-specific properties are matched as closely as possible between simulation and real-world conditions. This setup allows us to directly assess the sensitivity of the learned dynamics models to the domain gap that arises from sim-to-real transfer.



(a) Flat, Sim

(b) Stair, Sim

(c) Flat, Real

(d) Stair, Real

Figure 18: Each subplot shows the average loss for each dataset using 16 context vector sets derived from four datasets. Each set includes a terrain context (Flat (F) or Stair (St)) and a robot property context (Sim (Si) or Real (R)). Results are averaged over 10 trials, with adaptation and test sets randomly sampled from 20 seconds within the 40 seconds of data. Error bars indicate the standard deviation.

As shown in Fig. 18, the DMCM model exhibits a noticeably higher loss when robot-specific properties differ between simulation and reality, compared to when only the terrain differs. This is observed even though the Raisim stair and the real stair differ in height. While this analysis does not fully explain the entire sim-to-real gap, the model's predictions clearly reveal a measurable discrepancy in dynamics between the two domains, underscoring DMCM's ability to capture variations that arise from differences in robot embodiment.

D Go1 Reinforcement Learning Policy

D.1 Training Hyperparameter of the Reinforcement Learning Policy

To train the Reinforcement Learning(RL) policy of quadrupedal robot locomotion for rough terrain, we employ PPO [30] as our policy gradient algorithm within the Isaac Gym simulator [28].

We use a concurrent training architecture to estimate proprioceptive value while learning the policy [31].

D.1.1 Naive Policy

For the training curriculum we use only pyramid stair and slope terrain from [32] with 4096 parallel environments. We largely follow the motor gains, reward functions and randomization settings from [33] to enable robust sim-to-real transfer. Detailed input and hyperparameters are in Table 7, 8.

Observation Type	Input	Dim.
Proprioception	Linear body velocity estimation	3
	Angular body velocity	3
	Body height estimation	1
	Foot height estimation	4
	Contact probability estimation	4
	Command	3
	Projected gravity vector	3
	Action	12
	Joint position	12
	Joint velocity	12
	Action (2 time steps ago)	12
	Joint position (2 time steps ago)	12
	Joint velocity history (2 steps ago)	12
	Action (4 time steps ago)	12
	Joint position (4 time steps ago)	12
	Joint velocity history (4 steps ago)	12

Table 7: Observation Types and Dimensions

Parameter	Value
horizon length (dt: 0.02)	25
learning rate	3.0E-4
kl threshold	0.008
discount factor	0.99
entropy coef	0.001
clip ratio	0.2
batch size	102400
mini batch size	20480

Table 8: Hyperparameters for PPO, naive policy

For the linear body velocity, body height, foot height and contact probability, we use the concurrent estimation strategy using ground truth data from simulation. We also note that we give true value to the critic network during training the policy.

D.1.2 Remaining Policies

Beyond the naive policy environments, we expand the training curriculum by incorporating wavy and randomly distributed terrains and by increasing the difficulty levels of the pyramid stair and slope terrains. These enhanced environments are trained using 8,816 parallel simulation instances.

The input observations remain consistent with those listed in Table 7, while context parameters are concatenated at single-CAVIA policy and context vectors are concatenated at multi-DMCM policy. Training is performed using the hyperparameters detailed in Table 9.

Parameter	Value
horizon length (dt : 0.02)	25
learning rate	3.0 E-4
kl threshold	0.008
discount factor	0.99
entropy coef	0.001
clip ratio	0.2
batch size	220400
mini batch size	44080

Table 9: Hyperparameters for PPO, rest of the policies

D.2 Data Acquisition with Naive Policy

For the dynamics modeling and context collecting for reinforcement learning experiments, a naive policy is employed for data acquisition. This policy performs significantly worse than any of the trained policies, including the vanilla policy, as shown in Table 10. The evaluation is conducted under the same experimental conditions described in Appendix D.4. The naive policy is selected to ensure that context learning is not biased by high-performing behavior and that performance improvements in the trained policies are not attributable to favorable data collection.

Metric	Naive	Vanilla
Inner Distribution		
Success Counter	404/2000	**1580/2000**
RMSE Linear Vel (m/s)	0.2276	**0.1552**
Reward (w/o Termination Reward)	12.0252	**17.4848**
Lifespan	0.5886	**0.9190**
AvgDist (Success) (m)	4.8803	**7.3806**
OOD Terrain		
Success Counter	98/2000	**205/2000**
RMSE Linear Vel (m/s)	0.2912	**0.2555**
Reward (w/o Termination Reward)	5.9131	**8.5807**
Lifespan	0.3522	**0.4737**
OOD Robot-Specific Properties		
Success Counter	0/2000	**48/2000**
Lifespan	0.2586	**0.3050**

Table 10: Comparison between Naive and Vanilla policies across inner distribution, OOD terrain, and OOD robot-specific properties.

Context data for both the multi-DMCM and single-CAVIA policies are collected using this naive policy. The dataset comprises trajectories from 58 different terrains and 500 randomly sampled robot-specific property configurations (Table 11), yielding $58 \times 500 = 29{,}000$ tasks, each consisting of 2000 timesteps (40 seconds) of walking data.

Property	Min	Max
Mass offset (kg)	-1.0	$+4.0$
Center of Mass shift (m)	-0.05	$+0.05$
Action delay (s)	0.0	0.02
K_p gain offset (N · m)	-2.5	$+2.5$
K_d gain offset (N·m·s/rad)	-0.2	$+0.2$

Table 11: Randomized robot-specific properties and their value ranges used during RL training.

Compared to Table 6, torque limit variation is removed and the K_p gain range is slightly reduced. For each task, we randomly sample 700 points three times to generate context parameters and context vectors. This produces three context parameter sets per task for the single-CAVIA model, and three terrain context vectors along with three robot-specific property context vectors per task for the multi-DMCM model.

Real-World Data Collection: The same naive policy is used for real-world data collection. The following seven conditions are used to represent diverse robot-terrain settings:

1. Flat terrain
2. Flat terrain with 1.5kg payload
3. Climbing stairs (17cm depth)
4. Wavy terrain
5. Wavy terrain with 1.5kg payload
6. Flat terrain with 1.5kg payload and water bottle on legs
7. Flat terrain with 16N · m K_p (low gain) and 1.5kg payload

Data collection on stairs with a 1.5 kg payload was not possible, as the naive policy was unable to successfully complete the task. These datasets are used for adapting the CAVIA and DMCM dynamics models. The adapted contexts are then applied in the single-CAVIA and multi-DMCM policies for real-world deployment.

D.3 Training and Experiment Settings

During training, the context parameters or vectors are concatenated with the corresponding observations. For the single-CAVIA policy, one of the three generated context parameter sets is randomly

selected at the beginning of each episode. In contrast, the disentangled structure of DMCM allows the multi-DMCM policy to reuse context vectors derived from the same factor of variation across different tasks. During training, at multi-DMCM policy, randomly assigned context vectors that share the same underlying context are concatenated with observations. Additional analysis is provided in Appendix D.5.

Terrain Setup: We divide the environment terrains into the following 8 types:

1. Low stairs up
2. Low stairs down
3. Wavy terrain
4. Sloped terrain
5. Random terrain
6. High stairs down (with velocity limits)
7. High stairs up (with velocity limits)
8. High stairs up (with velocity limits and low friction)

Velocity Limits: For high stairs, the linear command x directional velocity ranges from $0.4\,\mathrm{m/s}$ to $0.6\,\mathrm{m/s}$, y directional velocity as 0, and yaw rate from $-0.6\,\mathrm{rad/s}$ to $0.6\,\mathrm{rad/s}$. Except for high stairs, the linear command x directional velocity ranges from $-1.0\,\mathrm{m/s}$ to $1.0\,\mathrm{m/s}$, y directional velocity from $-1.0\,\mathrm{m/s}$ to $1.0\,\mathrm{m/s}$, and yaw rate from $-\pi\,\mathrm{rad/s}$ to $\pi\,\mathrm{rad/s}$.

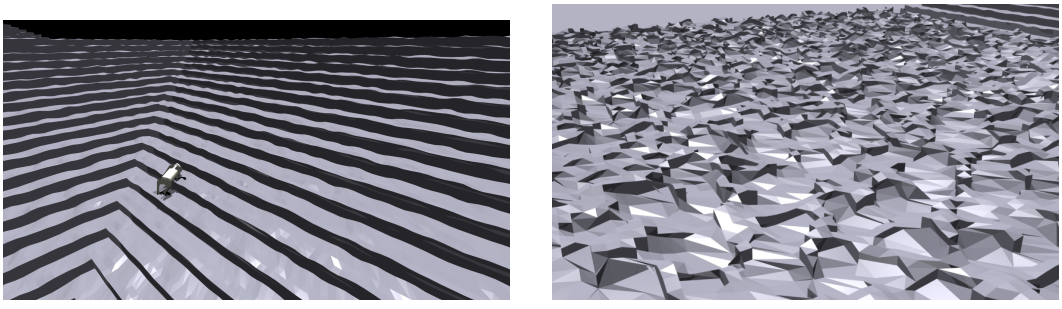
Robot-Specific Properties: Each environment is assigned one configuration sampled from Table 11.

Policy Selection: For all policies (vanilla, single-CAVIA, multi-DMCM), trained policy is selected with the highest reward after training over 33,000 episodes.

In this experiment, we focus on developing a policy that can climb stairs effectively. Therefore, 40% of environments are assigned to stair climbing task.

D.4 Evaluation Settings

Three environments are primarily used for evaluation in this study. The first is a high-stair environment, one of the most challenging terrains in the training settings, with a stair height of 0.27 m and a width of 0.31 m. The second is a highly randomized out-of-distribution (OOD) terrain, illustrated in Fig. 19b. For both environments, the same robot-specific property distributions as in training are applied. The third evaluation case involves OOD robot-specific properties, as listed in Table 12, while using the same terrain as the first environment, the steep stair.



(a) Highest level stair terrain

(b) Difficult Randomly distributed terrain used for OOD terrain

Figure 19: Terrains used for evaluation

Property	Min	Max
Mass offset (kg)	5.0	6.0
Center of Mass shift (m)	-0.05	$+0.05$
Action delay (s)	0.0	0.02
K_p gain offset (N · m)	-5.0	-4.0
K_d gain offset (N·m·s/rad)	-0.2	$+0.2$

Table 12: Randomized robot-specific properties and their value ranges used in the OOD robot-specific evaluation.

D.5 Robustness via Context Sharing at Multi-DMCM Policy

In the single-CAVIA setup, each task is associated with three distinct context parameter sets, which are restricted to that specific task and cannot be reused elsewhere. In contrast, DMCM's disentangled structure allows the multi-DMCM policy to share partial contexts across different tasks. For instance, a robot-specific context learned from one task can be combined with a terrain context learned from another. We further investigate whether evaluation performance improves when probabilistically assigning context vectors that share the same underlying factor of variation, or deterministically using the context vectors derived under the exact same conditions.

Metric	Multi-DMCM (Probabilistic)	Multi-DMCM (Deterministic)
Inner Distribution		
Success Counter	**1892/2000**	1688/2000
RMSE Linear Vel (m/s)	**0.0816**	0.0992
Reward (w/o Termination Reward)	**21.9682**	19.4695
Lifespan	**0.9851**	0.9433
AvgDist (Success) (m)	**8.4581**	8.0676
OOD Terrain		
Success Counter	281/2000	**431/2000**
RMSE Linear Vel (m/s)	**0.2486**	0.2607
Reward (w/o Termination Reward)	**10.0295**	9.1101
Lifespan	0.5476	**0.6104**
OOD Robot-Specific Properties		
Success Counter	**602/2000**	151/2000
RMSE Linear Vel (m/s)	**0.0944**	0.1083
Reward (w/o Termination Reward)	**19.2283**	17.7090
Lifespan	**0.6009**	0.3813
AvgDist (Success) (m)	**7.9205**	7.2410

Table 13: Performance comparison of multi-DMCM (Probabilistic) and multi-DMCM (Deterministic) across inner distribution, OOD terrain, and OOD robot-specific properties. Best values in each row are bolded.

The results in Table 13 demonstrate that the multi-DMCM policy generally performs better when using context vectors probabilistically selected from other tasks sharing the same factor, rather than those derived from the current task. This property suggests potential benefits for real-world deployment, as it reflects the policy's robustness in interpreting context vectors. For the multi-DMCM policy, a probabilistically selected context configuration is used during all the other evaluations.

D.6 Verifying Policy's Context Utilization

D.6.1 Policy Performance with Incorrect Context

We evaluate the performance of both single-CAVIA and multi-DMCM policies. However, their performance may not degrade significantly even when incorrect context vectors or parameters are used, raising questions about whether they truly utilize context information. Therefore, we check the performance with context parameters and vectors derived from the wrong terrain context and randomly selected robot-specific contexts on the steep stair evaluation (Fig.19a). Specifically, we assign the terrain context of stair down instead of stair up.

Metric	Multi-DMCM	Multi (Wrong)	Single-CAVIA	Single (Wrong)	Vanilla
Success	1892/2000	1323/2000	1905/2000	1339/2000	1580/2000
RMSE Lin Vel (m/s)	0.0816	0.0931	0.0798	0.1232	0.1552
Reward	21.97	20.32	22.18	18.57	17.48
Lifespan	0.9851	0.8652	0.9852	0.8598	0.9190
AvgDist (Success) (m)	8.4581	8.3407	8.5203	7.3069	7.3806

Table 14: Comparison across different methods under inner distribution conditions.

As Table 14 shows, performance drops across all metrics for both policies when incorrect context information is used, with success rate and lifespan falling below those of the vanilla policy. These results confirm that both single-CAVIA and multi-DMCM policies actively rely on context information during deployment.

D.6.2 Effect of Contexts on Multi-DMCM Policy

To examine whether terrain and robot-specific property contexts meaningfully influence behavior in the multi-DMCM policy, we design controlled experiments that isolate the effect of each context type. Two types of variations are considered: (i) **Robot-specific context variation**: payload differences. (ii) **Terrain context variation**: changes in stair level and type (ascending and descending).

The robot is evaluated on flat terrain without payload as the baseline. Then, incorrect context vectors are deliberately assigned from variation cases (i) and (ii). For the robot-specific context variation, assigning a high-payload context to the no-payload setting increased the robot's body height (Table 15), suggesting that the policy compensates for weight that is not actually present. For the terrain context variation, assigning a steep-terrain context on flat ground increased the standard deviation of the front foot's height (Table 16), indicating that the blind policy adapts by probing more aggressively with its front legs to detect terrain changes, prioritizing safety at the cost of stability.

Mass for Context	**-1.5kg**	**0kg**	**3kg**	**6kg**	**9kg**
Mean Body Height (m)	0.327	0.329	0.341	0.374	0.387

Table 15: Mean body height under different mass contexts.

Terrain	High Down	Low Down	Flat	Low Up	High Up
Std (L/R)	0.0283 / 0.0351	0.0236 / 0.0329	0.0215 / 0.0316	0.0240 / 0.0333	0.0271 / 0.0352

Table 16: Standard deviation of front leg heights (Left/Right) under different terrain contexts.

D.7 Additional Real-World Experiments

We further evaluate the performance of the multi-DMCM, single-CAVIA, and vanilla policies on a 17 cm stair-climbing task, followed by more challenging scenarios designed for the multi-DMCM model.

D.7.1 Stair with 1.5kg Payload

All three policies successfully complete the 17 cm stair-climbing task with a 1.5 kg payload. For the multi-DMCM policy, the terrain context is taken from Isaac Gym simulation data of an 18 cm stair climb, while the robot-specific context comes from real-world flat-terrain data with a 1.5 kg payload. The single-CAVIA policy uses Isaac Gym data of an 18 cm stair climb with a 1.5 kg payload, with other robot-specific properties without domain randomization. Despite the added mass, all models complete the task without failure.

D.7.2 Multi-DMCM Policy in Complex Scenarios

To further test robustness under more complex conditions, asymmetrical payloads are added to the robot's legs: a 500 ml water bottle on the front left leg, and 300 ml bottles on the front right and rear left legs (Fig. 20).

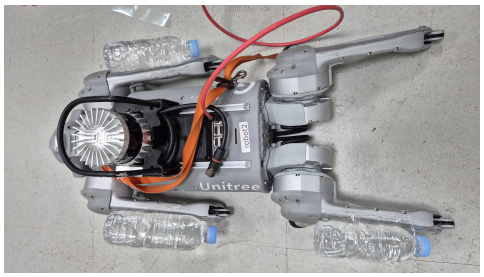


Figure 20: Go1 robot with water bottles attached at three legs

Using context vectors obtained from previously collected data, the multi-DMCM policy successfully navigates both stair and wavy terrains. For the stair task, the terrain context is taken from simulation data of an 18cm stair climb, and the robot-specific context is taken from real-world flat-terrain data with the water-bottle payload. For the wavy terrain, the terrain context comes from real-world wavy-terrain data without payload, while the flat-terrain water bottle setup context is reused for robot-specific context. In both scenarios, the multi-DMCM policy achieves stable locomotion with no failures.