Learning Domain-Adaptive Manipulation Sequences with Environment Models and Decision Transformers

Ashwin Ravindra Bharadwaj[1]

[1]Northeastern University, Khoury College of Computer Science

Abstract

We propose a novel approach for solving reinforcement learning problems under domain shift by leveraging a domain-aware Decision Transformer. Instead of training a single policy across all environments or fine-tuning per domain, our method learns to condition the policy on the domain configuration itself. We construct a dataset of expert trajectories across diverse domains and train a transformer model to autoregressively predict actions, states, and returns. Our method achieves strong generalization performance in previously unseen domain settings, offering an efficient and scalable alternative to domain randomization and per-domain policy optimization.

1 Introduction

Reinforcement learning (RL) has achieved remarkable success in simulation and control tasks; however, generalization across domain variations remains a critical challenge. Most RL agents are sensitive to small shifts in environmental parameters such as friction, mass, or external forces. This sensitivity is particularly problematic when deploying policies trained in simulation to real-world systems or when the dynamics themselves vary across deployment contexts.

Two common strategies have emerged to address this problem. The first is extensive domain randomization [15, 10], where a simple simulator is parameterized with a wide range of randomized physics settings, and an RL agent is trained to learn a policy that performs adequately across all of them. While this can result in robust average performance, the learned policy is often suboptimal for any specific domain and can degrade under unseen or rare configurations.

The second strategy is to construct highly detailed and domain-specific simulators that model every nuance of the environment [1, 12]. Policies trained in such simulators can achieve high performance in the exact training domain but tend to generalize poorly to others, limiting scalability and applicability in dynamic or uncertain real-world settings.

In this work, we propose a third path: we introduce the *Domain-Aware Decision Transformer*, a model that conditions its policy on the current domain configuration. Instead of attempting to learn a single policy for all domains or overfitting to one, we model the policy as a function of the domain itself. At deployment time, the model infers or is provided with the domain variables and outputs an optimal or near-optimal action trajectory for that configuration. Our approach combines the robustness of domain randomization with the specialization of per-domain optimization, yielding higher average performance across a variety of RL environments.

2 Preliminaries

This section outlines the foundational concepts behind our approach, including reinforcement learning (RL), transformer architectures, and domain adaptation techniques.

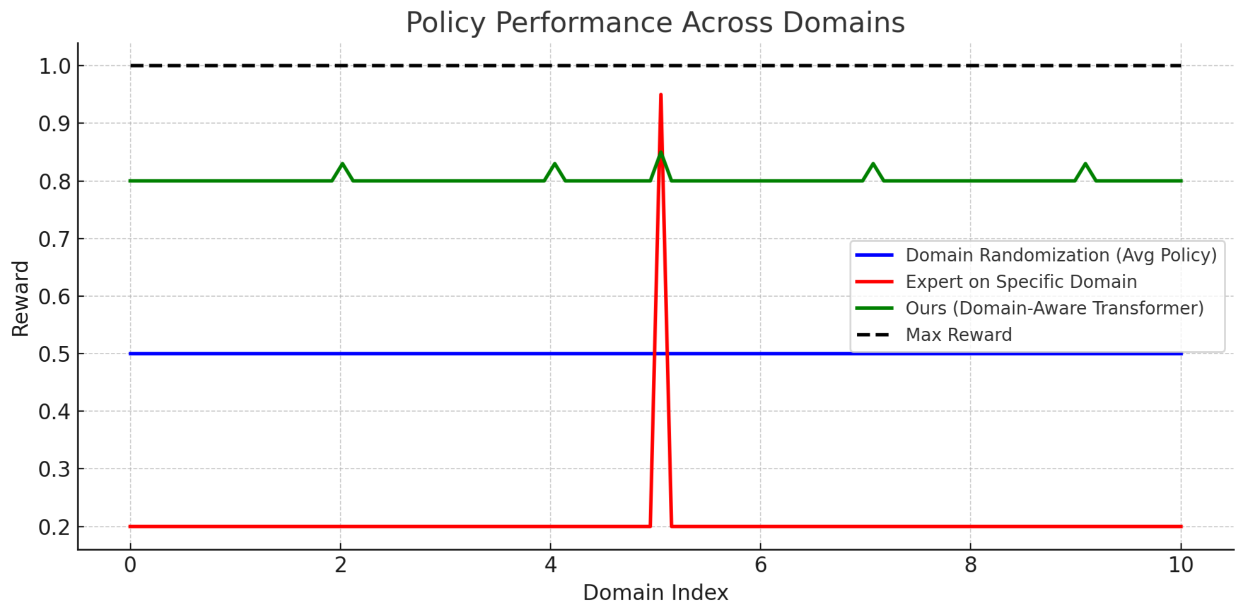

Figure 1: The above image indicates the expected results that we wish to see. The small spikes that are seen on the DADT cureve are the points where the DADT was trained on expert trajectories.

2.1 Reinforcement Learning and Online Learning

Reinforcement Learning (RL) is a framework where agents learn to maximize expected cumulative rewards through interactions with an environment [14]. In online RL, learning and data collection happen simultaneously, often resulting in instability but enabling real-time adaptation. Our approach differs in that it leverages offline trajectories from domain-specific experts, yet incorporates inference steps that resemble online adaptation.

2.2 Transformers for Sequential Modeling

Transformers [16] have become the de facto architecture for modeling sequences due to their scalability and ability to capture long-range dependencies. Recent adaptations in decision-making, such as Decision Transformers [3], apply these architectures to RL by predicting actions as part of an autoregressive trajectory model. We adopt a GPT-2 variant and extend it with multi-modal embeddings for action, state, return, and domain configurations.

2.3 Domain Variation and Invariant Learning

Domain adaptation addresses the challenge of deploying models trained in one domain to another with differing dynamics. Domain randomization [15], meta-learning [6], and feature alignment [8] are common strategies. In our work, we leverage domain-parameterized transitions and infer the likely domain configuration through prediction-based matching, offering an implicit domain adaptation mechanism.

3 Problem Formulation

We model robotic manipulation tasks as a family of Markov Decision Processes (MDPs). Each MDP represents a specific instantiation of an environment with fixed physics and object properties [13, 10, 15, 3].

This problem setup reflects a general trend in domain-adaptive RL [11, 19, 5], where a single agent is expected to operate robustly across variations in physical parameters such as friction, mass, and gravity.

Formally, an MDP is defined as:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma) \tag{1}$$

where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function,
- $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward function (state-only dependent),
- $\gamma \in [0, 1]$ is the discount factor.

We introduce a domain configuration variable $\mathcal{D} \in \mathcal{D}_{\text{space}}$, which alters the environment dynamics. Each domain configuration $\mathcal{D}$ induces a unique transition function $\mathcal{T}_{\mathcal{D}}$, resulting in a domain-specific MDP:

$$\mathcal{M}_{\mathcal{D}} = (\mathcal{S}, \mathcal{A}, \mathcal{T}_{\mathcal{D}}, \mathcal{R}, \gamma) \tag{2}$$

We assume a one-to-one mapping between $\mathcal{D}$ and $\mathcal{T}$:

$$\forall \mathcal{D} \in \mathcal{D}_{\text{space}}, \exists!\ \mathcal{T}_{\mathcal{D}} \tag{3}$$

Given that $\mathcal{R}$ and γ are unchanged across domains, only the dynamics differ between environments. As a result, the optimal policy $\pi^*_{\mathcal{D}}$ varies with $\mathcal{D}$:

$$\pi^*_{\mathcal{D}} = \arg\max_{\pi} \mathbb{E}_{\pi, \mathcal{T}_{\mathcal{D}}} \left[\sum_{t=0}^{T} \gamma^t \mathcal{R}(s_t) \right] \tag{4}$$

Our transformer-based policy model is trained to learn this mapping by conditioning on both the current state s_t and the domain configuration $\mathcal{D}$:

$$\pi_\theta(a_t | s_t, \mathcal{D}) \tag{5}$$

Moreover, our transformer also predicts the expected next state $\hat{s}_{t+1}$ and the expected reward $\hat{r}_t$:

$$(a_t, \hat{s}_{t+1}, \hat{r}_t) = \text{Transformer}_\theta(s_t, \mathcal{D}) \tag{6}$$

This allows the system to infer the most probable domain configuration $\mathcal{D}$ by comparing its predictions against actual observations. Specifically, after executing a few actions and observing the true state transitions and rewards, we compute a likelihood over candidate domain configurations:

$$\mathcal{D}^* = \arg\max_{\mathcal{D}} \prod_{t=0}^{K} P(\hat{s}_{t+1} \approx s_{t+1}, \hat{r}_t \approx r_t | \mathcal{D}) \tag{7}$$

The above equation can be written as

$$P_t(d_i) = \frac{P_{t-1}(d_i)\ L(d_i \mid s_t, a_t, s_{t+1}, r_t)}{\sum_j P_{t-1}(d_j)\ L(d_j \mid s_t, a_t, s_{t+1}, r_t)}$$

where L represents the loss between the predicted states and predicted rewards and the states and rewards given from the environment. In our case we used MSE for the above loss for continuous spaces. P_i represents the probability of a particular domain after i steps of exploration.

This framework enables the transformer to dynamically estimate $\mathcal{D}$ during deployment and adapt its policy accordingly.

Domain Inference via Prediction Consistency

To better illustrate our domain inference procedure, consider a simple scenario where the environment could belong to one of two domains: **Domain A**, which has no wind, and **Domain B**, where a wind force constantly pushes the agent to the left.

During the exploratory phase, the transformer predicts the outcome of actions under each domain hypothesis:

- Under **Domain A (no wind)**, the predicted trajectory is:

$$S_{00} \xrightarrow{A_{\text{Right}}} S_{01} \xrightarrow{A_{\text{Up}}} S_{11}$$

- Under **Domain B (with wind)**, the predicted trajectory is:

$$S_{00} \xrightarrow{A_{\text{Right}}} S_{00} \xrightarrow{A_{\text{Up}}} S_{10}$$

Suppose the actual environment corresponds to Domain B, and the observed trajectory is:

$$S_{00} \xrightarrow{A_{\text{Right}}} S_{00} \xrightarrow{A_{\text{Up}}} S_{10}$$

The model compares predicted and actual trajectories under each domain using cosine similarity. Since the prediction under Domain B aligns more closely with the observed trajectory, it has a higher similarity score (or equivalently, a lower prediction error).

To update the probability of each domain being correct, we use the Bayesian rule as given in the formula, where the likelihood term is computed as the inverse of mean squared error (MSE) between predicted and observed states. Assuming an initial uniform prior of $P_0(D_A) = P_0(D_B) = 0.5$, this update will result in a higher posterior probability for Domain B:

$$P_2(D_B) > P_2(D_A)$$

Through this mechanism, the model gradually refines its belief about the current domain using only the differences between observed and predicted transitions.

Future Direction: Model-Based Inference

While currently we have used a static setup where the exploration of the agent to identify the domain is fixed we wish to use a more active algorithm that would balance both exploration and exploitation to search for the ideal domain. This approach will also overcome the other drawback being the inability to handle environments that change over time.

4 Methodology

We evaluate our framework on two environments to capture diverse physical phenomena and dynamics:

4.1 Environment Descriptions

Ball-on-Plate Balancer: A robotic agent controls a plate by tilting it along the X and Y axes. The objective is to maintain a ball at the center of the plate despite external disturbances. Domain variables include surface friction, wind components in X and Y directions, and an action scaling factor ranging from -1 to 1.

Quadrupedal Locomotion (Unitree Go1): A walking robot must traverse a terrain under varying physical settings. Domain variables include the maximum torque of each motor, total body mass, surface inclination angle, and ground friction. These factors alter the gait dynamics and contact interactions.

Both of the above environments was built using the above encironments were built using the mujoco playground [17].

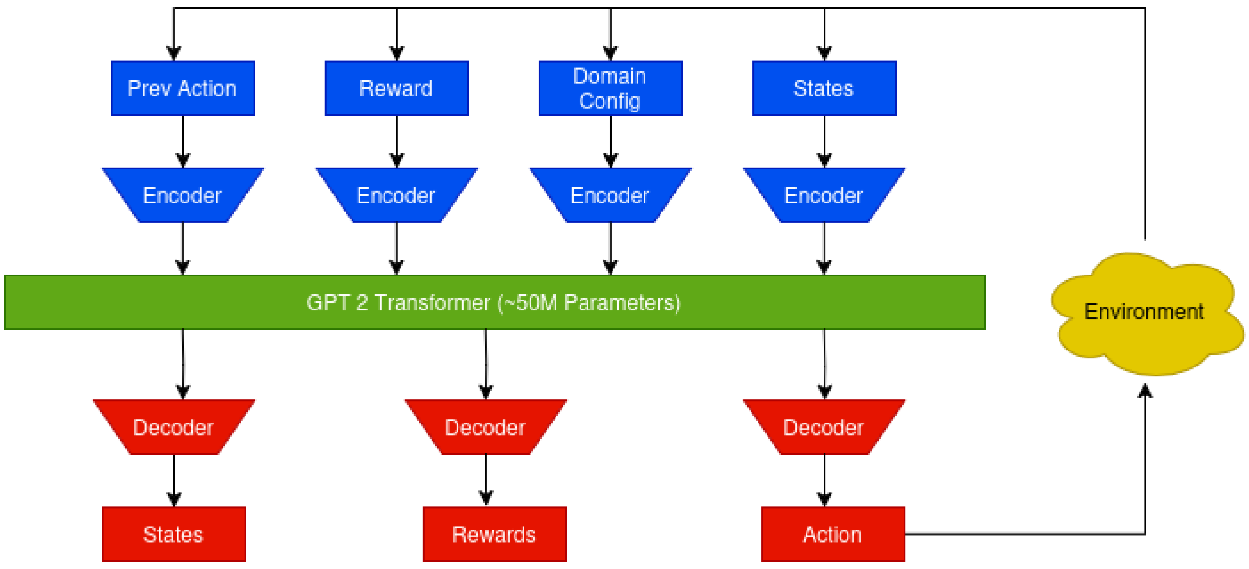

Figure 2: Model Architecture

4.2 Dataset Generation

We define a grid over the domain variable space and sample n domain configurations uniformly to ensure balanced coverage. For each sampled domain configuration $\mathcal{D}$, we fine-tune a standard reinforcement learning agent until convergence. The algorithms we developed in Brax to take advantage of the hardware to speed up the training [7]. We used PPO for the Quadrupedal Locomotion (Unitree Go1) environment and SAC for Ball-on-Plate Balancer. We considered that the models had converged if they were able to achieve 85% of the maximum reward possible in the given time steps. These expert agents are then used to generate datasets of successful trajectories:

$$\mathcal{D}_{\text{traj}} = \{(s_t, a_t, R_t, \mathcal{D})\}_{t=0}^{T} \tag{8}$$

Where R_t denotes the return-to-go from time t:

$$R_t = \sum_{k=t}^{T} r_k \tag{9}$$

Each tuple captures a full transition within a particular domain. This dataset mimics the format proposed in Decision Transformers [3], which recast reinforcement learning as sequence modeling using autoregressive transformers.

4.3 Model Architecture

Our model architecture builds on GPT-2 and integrates modifications inspired by advances in computer vision [4] and audio modeling [2]. The input to the model at each step consists of a tuple: $(s_t, a_t, R_t, \mathcal{D})$, which we refer to as a "step token." Each component is embedded separately, and positional encodings are applied at the step level rather than to individual tokens.

The model processes these sequences autoregressively, predicting future actions, states, and returns-to-go. We introduce normalization layers between transformer blocks to stabilize training across modalities, following strategies shown effective in multi-modal transformer architectures [4, 2].

Algorithm 1 Transformer Forward Pass

```
Require: States $s \in \mathbb{R}^{B\times T\times d_s}$, Actions $a \in \mathbb{R}^{B\times T\times d_a}$, Returns-to-go $R \in \mathbb{R}^{B\times T\times 1}$, Timesteps $t \in \mathbb{N}^{B\times T}$, Config $\mathcal{D} \in \mathbb{R}^{B\times T\times d_d}$
Ensure: Predicted states $\hat{s}$, actions $\hat{a}$, returns $\hat{R}$
1: Embed each modality to $\mathbb{R}^{B\times T\times H}$: $e_s, e_a, e_R, e_D$
2: Add time embeddings $e_t$ to each modality embedding
3: Stack $[e_R, e_s, e_a, e_D] \to \mathbb{R}^{B\times 4T\times H}$
4: Apply layer normalization and attention mask
5: Pass through GPT-2 transformer: $x \in \mathbb{R}^{B\times 4T\times H}$
6: Reshape $x \to \mathbb{R}^{B\times 4\times T\times H}$
7: Predict: • $\hat{R}_t = f_R(x_{action}) \in \mathbb{R}^{B\times T\times 1}$ • $\hat{s}_{t+1} = f_s(x_{action}) \in \mathbb{R}^{B\times T\times d_s}$ • $\hat{a}_t = f_a(x_{state}) \in \mathbb{R}^{B\times T\times d_a}$
8: return $\hat{s}, \hat{a}, \hat{R}$ =0
```

4.4 Training Procedure

We adopt a multi-task learning objective with three separate losses:

- $\mathcal{L}_a$: Mean squared error for continuous action prediction
- $\mathcal{L}_s$: Mean squared error for next state prediction
- $\mathcal{L}_r$: Mean squared error for reward-to-go prediction

To prevent imbalance due to differing output scales, we normalize each loss using running statistics. Specifically, we adopt the dynamic weighting scheme proposed by [9], which adjusts task weights according to their observed variance during training:

$$\mathcal{L}_{\text{total}} = \alpha_t \mathcal{L}_a + \beta_t \mathcal{L}_s + \gamma_t \mathcal{L}_r \tag{10}$$

Here, $\alpha_t, \beta_t, \gamma_t$ are computed online based on standard deviation normalization, allowing the network to balance learning progress across modalities.

Algorithm 2 Autoregressive Training Loop for Transformer Policy

```
1: for each epoch do
2:   for each batch $(s, a, r, R, t, \mathcal{D})$ from the dataset do
3:     Shift inputs by one step: use tokens from $t = 0$ to $T-2$
4:     Forward pass through transformer:
           $(\hat{s}_{t+1}, \hat{a}_t, \hat{R}_{t+1}) = f_\theta(s_{0:T-2}, a_{0:T-2}, R_{0:T-2}, t_{0:T-2}, \mathcal{D}_{0:T-2})$
5:     Define targets as ground truth from $t = 1$ to $T-1$
6:     Compute losses:
           $\mathcal{L}_s = \mathrm{MSE}(\hat{s}_{t+1}, s_{1:T-1})$
           $\mathcal{L}_a = \mathrm{MSE}(\hat{a}_t, a_{1:T-1})$
           $\mathcal{L}_r = \mathrm{MSE}(\hat{R}_{t+1}, R_{1:T-1})$
7:     Normalize losses using running statistics:
           $\mathcal{L}_{\text{total}} = \alpha_t \mathcal{L}_s + \beta_t \mathcal{L}_a + \gamma_t \mathcal{L}_r$
8:     Update model parameters with gradient descent
9:   end for
10: end for=0
```

4.5 Evaluation and Domain Identification

During evaluation, we perform an exploratory phase in which the model is initialized with a randomly selected domain configuration $\mathcal{D}$. Using this setting, it predicts future states and returns. These predictions are compared to the true observations, and the model selects the configuration $\mathcal{D}^*$ that minimizes prediction error over a predefined exploration horizon:

$$\mathcal{D}^* = \arg\min_{\mathcal{D}} \sum_{t=0}^{K} \left[\|\hat{s}_{t+1} - s_{t+1}\|^2 + (\hat{r}_t - r_t)^2 \right] \tag{11}$$

Once the optimal domain estimate $\mathcal{D}^*$ is identified, the policy uses only the predicted action outputs for deployment:

$$\hat{a}_t = \pi_\theta(s_t, \mathcal{D}^*) \tag{12}$$

Although this inference step is currently done offline with fixed-length exploration, future work will focus on making it online and adaptive. I think it can be done in a few days. Stuck with some of the graphs.

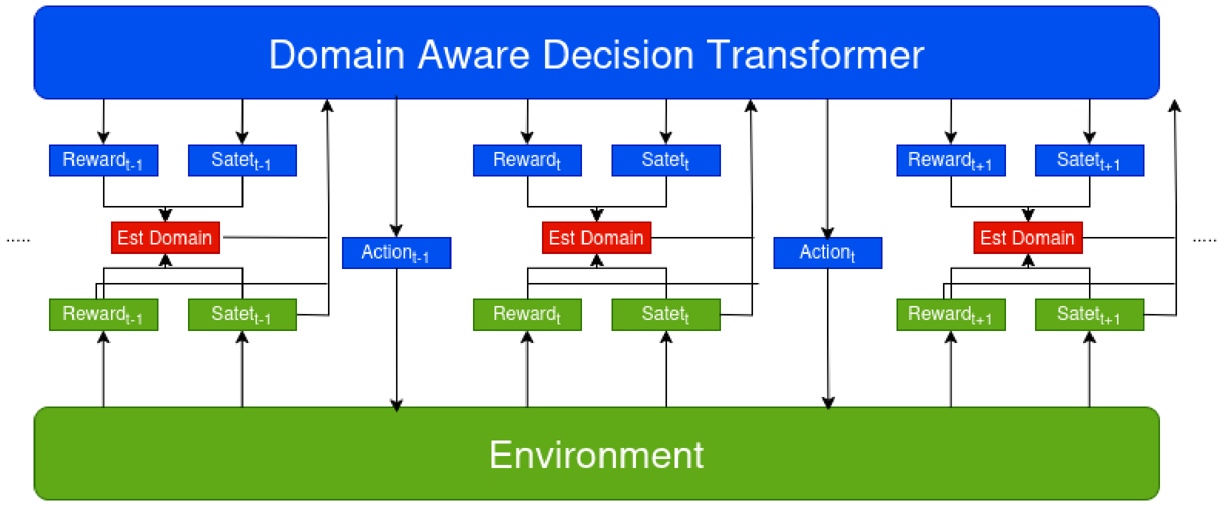

Figure 3: The above picture represents the interaction between the model. The time steps increase to the right.

Algorithm 3 Infer Most Likely Domain

```
0: function InferDomain(search_steps, $(d_{low}, d_{high})$, num_candidates, starting_domain)
0:   $candidates \leftarrow$ SampleUniform($d_{low}, d_{high}$, num_candidates)
0:   $scores \leftarrow$ zeros(num_candidates)
0:   $state \leftarrow$ env.reset()
0:   $history \leftarrow [\,]$
0:   for $t = 1$ to search_steps do
0:     $action \leftarrow$ model.get_action($state$)
0:     $(next\_state, reward, done) \leftarrow$ env.step($action$)
0:     Append $(state, action, next\_state, reward)$ to $history$
0:     $state \leftarrow next\_state$
0:     if $done$ then
0:       break
0:     end if
0:   end for
0:   for $i = 1$ to num_candidates do
0:     $domain \leftarrow candidates[i]$
0:     $total\_error \leftarrow 0$
0:     for each $(s, a, s', r)$ in $history$ do
0:       $(\hat{s'}, \hat{r}) \leftarrow$ model.predict_next($s, a, domain$)
0:       $total\_error$ += MSE($\hat{s'}, s'$) + MSE($\hat{r}, r$)
0:     end for
0:     $scores[i] \leftarrow total\_error$
0:   end for
0:   $best\_idx \leftarrow \arg\min(scores)$
0:   return $candidates[best\_idx]$
0: end function=0
```

5 Results

We present a comprehensive evaluation of our domain-adaptive transformer model across three environments: Ball-on-Plate Balancer, Quadrupedal Locomotion, and [Third Environment Placeholder]. Each environment contains multiple domain configurations characterized by physical parameters such as friction, mass, and external disturbances. Our results are presented through three key types of visualizations.

5.1 Reward Landscape Across Domain Grid

To evaluate generalization and robustness across domain variations, we visualize the reward surface achieved by both expert RL agents and our transformer-based policy in a compressed domain space. The high-dimensional domain configuration $\mathcal{D}$ is reduced to two dimensions using Principal Component Analysis (PCA), allowing for effective visualization while preserving structure among domain configurations [**?**].

We discretize the 2D PCA-projected domain space into a uniform grid. For each point:

- **Expert RL Agents:** We evaluate agents trained specifically on selected domain configurations and record the average reward over 10 episodes. These points are marked explicitly on the grid.
- **Transformer Model:** For interpolated regions between expert points, we sample random domain configurations and evaluate our transformer model, which is conditioned on the known domain vector $\mathcal{D}$. The model is evaluated for 10 episodes at each location.

The transformer is provided the correct domain configuration during inference and does not perform any additional domain estimation in this experiment. This isolates and evaluates the model's capacity to generalize across dynamics variations once $\mathcal{D}$ is known.

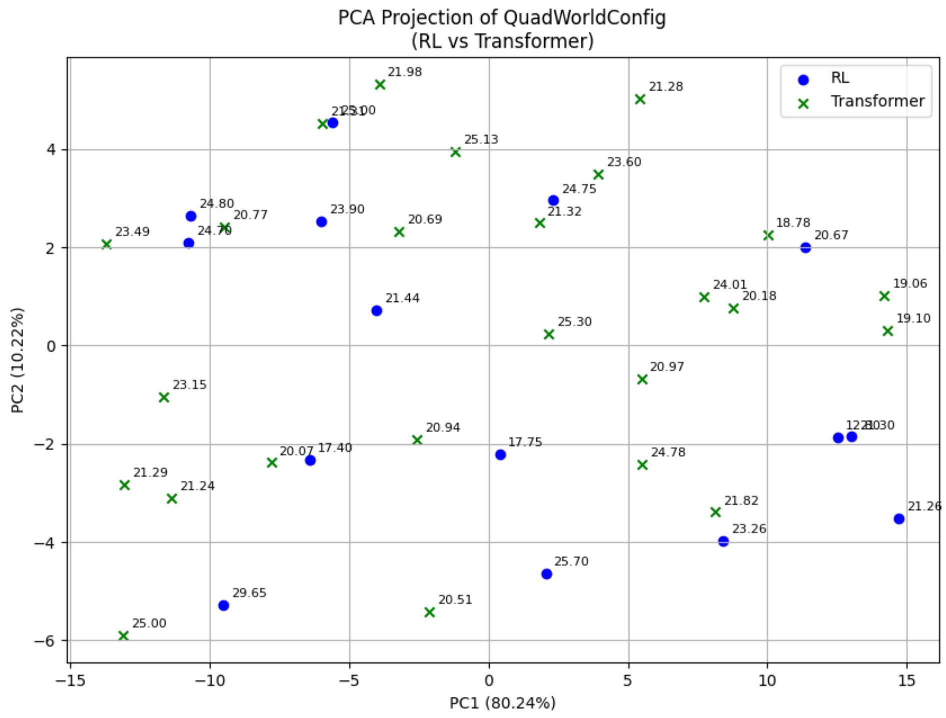

Figure 4: Comparison of expert RL agents and transformer policy in the 2D-projected domain space for the Quadrupedal Locomotion environment. The mean reward for the RL agent on its trained domains was 22.29 (std: 3.96), and for the transformer model across interpolated domains it was 21.83 (std: 1.91).

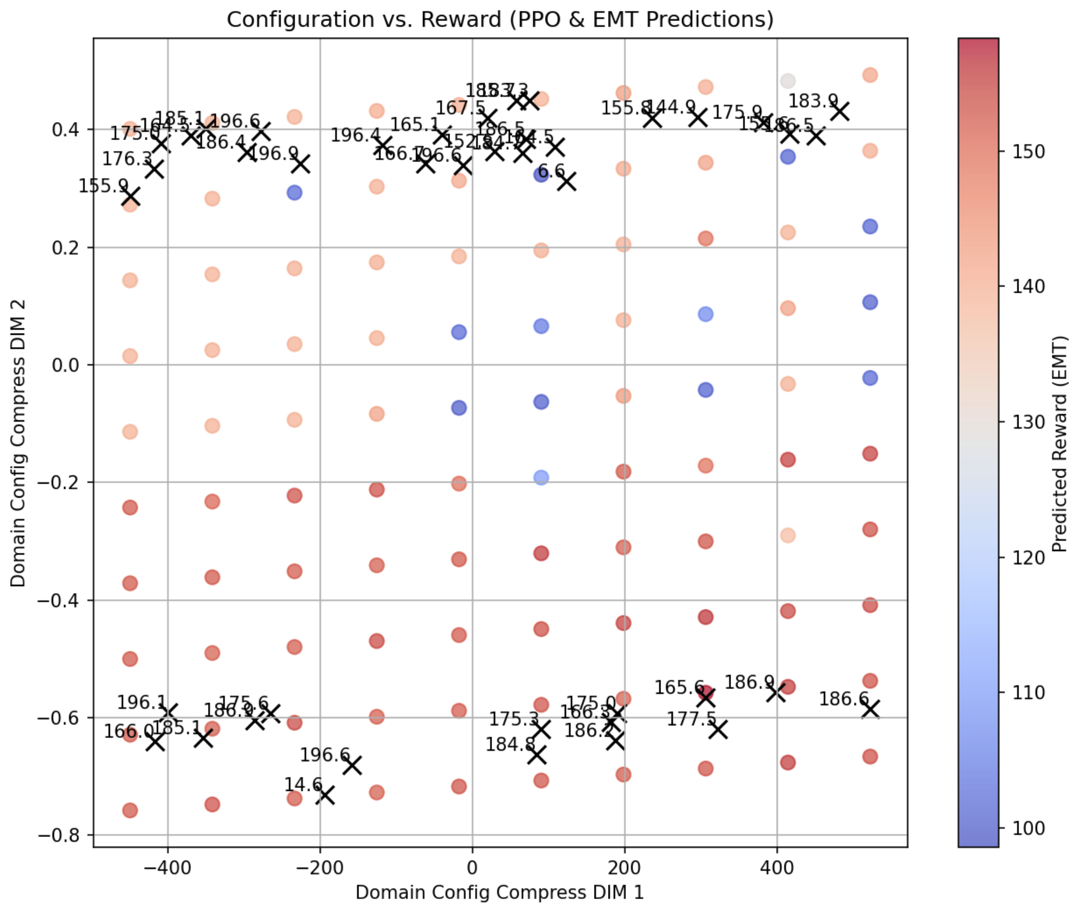

Figure 5: Reward surface comparison between expert RL and transformer model in the Ball-on-Plate Balancer environment.

These plots demonstrate that the transformer model, when conditioned on an accurate domain configuration, is capable of producing behavior that closely approximates or matches that of expert RL agents trained directly in those environments, despite never having seen those domain configurations during training.

5.2 Comparison of Average Rewards

We compare the performance of three different models across all environments using bar plots:

1. **Domain-Aware Transformer (Ours)**: Evaluated on 10 randomly sampled, unseen domain configurations.

2. **Expert RL Agent**: Trained and evaluated on each sampled domain.

3. **Unadapted RL Agent**: Trained on one specific domain and evaluated on unseen ones.

4. **Decision Transformer**: Trained a ordinary decision transformer for each particular domain and then evaluated it.

All of the above models were tested on unseen domains except for the Expert RL Agent. For the DADT we allowed the model to infer the domain. For these results we used an exploration period of 50 steps to optain the best resullts.

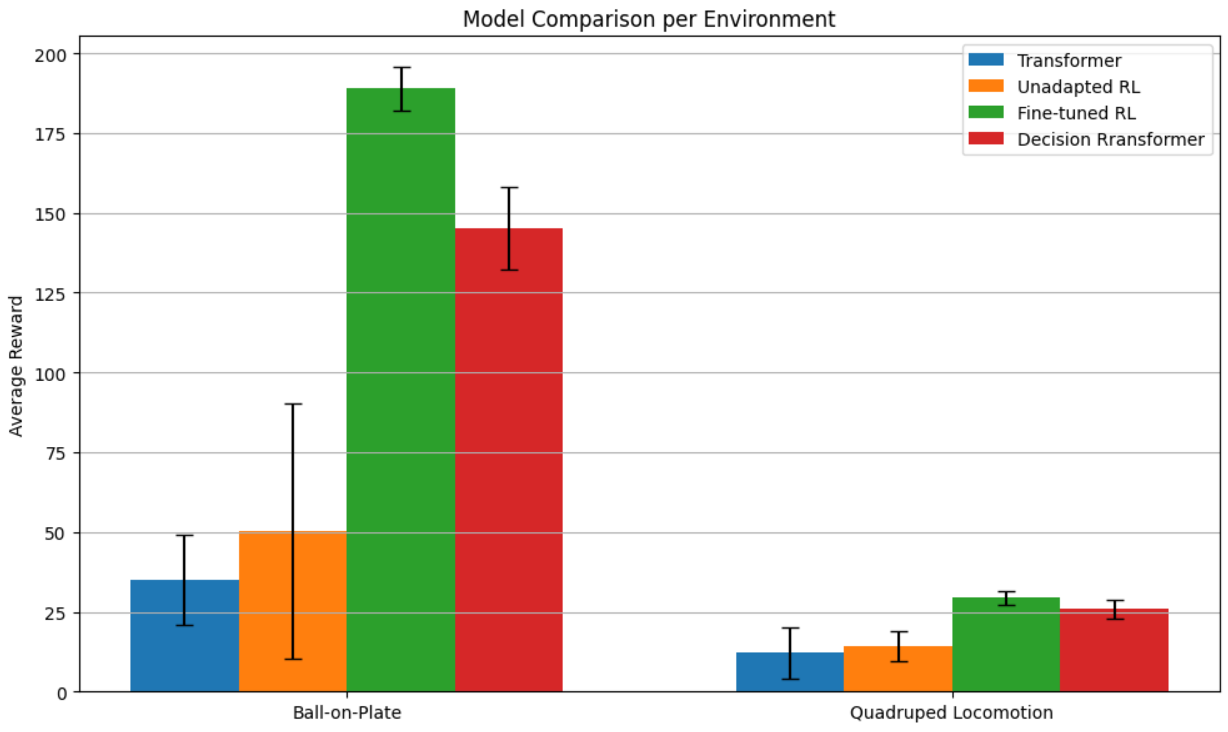

Figure 6: Average returns comparisons with other algorithms on the two environment.

This comparison helps quantify the average performance of our transformer-based approach versus standard RL agents under both matched and mismatched training/test domain conditions. The results are reported separately for each environment.

From the graph it is easy to see that the standard divination of the models that do not know the domain or use a policy that is not specific for the domain is the highest. This is expected as these models use a constant policy that might or might not be the optimal policy for that specific tested domain.

It can also be seen that a specified RL agent that is fine tuned for a specific domain performs best compared to all the other tested methods, but when the same model is exposed to unseen domains the average expected reward is drastically reduced. Important to note, that the RL fine tune RL agent only see one domain which it was tested on, while the rest of the methods are tested with domains that were not seen during training.

We can see that out of all the methods where the domain was unknow to the model before the start of the testing phase, the DADT gave the highest reward.

5.3 Adaptation Efficiency

We analyze how quickly our model infers the correct domain configuration $\mathcal{D}^*$ by plotting the predicted domain performance as a function of the number of exploratory episodes allowed during domain inference. We run evaluations for 25 to 50, episodes with increments of 5 and record the resulting MSE between the predicted domain and the actual domain. The expected and predicted domains are considered to be vectors and are compared to each other using the cosine similarity score. The domain values are normalized before measuring the difference.

As can be seen the DADT can infer the domain it is in and this ability to infer increases with the number of exploratory steps. The above results are the mean values that were collected after 10 runs at every datapoint.

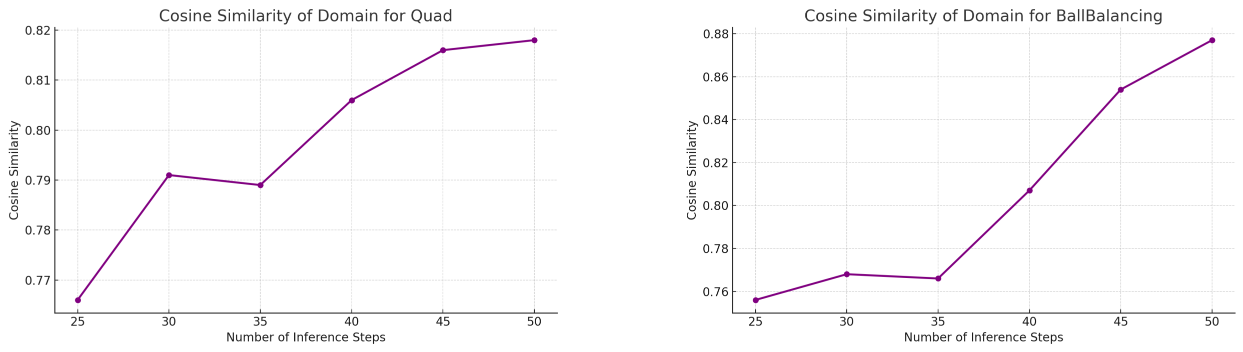

Figure 7: Domain inferences scores compared to number of exploratory steps

6 Conclusion

We introduced a domain-adaptive Decision Transformer capable of robust manipulation of environments across varied physics configurations. By combining sequence modeling with domain randomization, our approach generalizes well to previously unseen environments and can serve as a foundation for real-world deployment. Our next steps would be to recreate the above results in a few more environments using Mujoco Menagerie[18]. We are also currently experimenting we a few algorithms that would give the model the ability to infer a changing domain over time by utilizing a sliding window.

References

[1] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[2] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, 33:12449–12460, 2020.

[3] Lili Chen, Kevin Lu, Aravind Srinivas Rutter, Dhruv Tirumala, and Pieter Abbeel. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, volume 34, pages 15084–15097, 2021.

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[5] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[7] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.

[8] Yaroslav Ganin and Victor Lempitsky. Domain-adversarial training of neural networks. In *Journal of Machine Learning Research*, volume 17, pages 1–35, 2016.

[9] Biswajit Paria, Tianhe Yang, Mengye Lin, Timothy M Hospedales, Marc'Aurelio Ranzato, and Jian Tang. Minimax weighting of losses for multi-task learning. In *International Conference on Learning Representations*, 2022.

[10] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[11] Sahand Rahmatizadeh, Lirui Weng, Peter Allen, et al. Promptadapt: Domain adaptation via prompting for vision-language robot policies. *arXiv preprint arXiv:2206.15320*, 2022.

[12] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, Igor Mordatch, and Vikash Kumar. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Conference on Robot Learning*, pages 255–263. PMLR, 2017.

[13] Richard S Sutton. Integrated modeling and control based on reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 471–478, 1990.

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

[15] 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)*, pages 23–30. IEEE, 2017.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[17] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A. Kahrs, Carmelo Sferrazza, Yuval Tassa, and Pieter Abbeel. Mujoco playground, 2025.

[18] Kevin Zakka, Yuval Tassa, and MuJoCo Menagerie Contributors. MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo, 2022.

[19] Weiyang Zhang, Julian Kuo, Alexey Pashevich, Pete Florence, Anthony Brohan, Florent Tasse, and Silvio Savarese. Domain adaptation of visual policies with a single demonstration. *arXiv preprint arXiv:2407.16820*, 2024.