GenHyper : Adversarial Generative Hypernetworks for Efficient Policy Adaptation

Jayaram Reddy*[1], Sanket Kalwar*[1], Anant Garg*[1],
Vishal Mandadi[1], Brojeshwar Bhowmick[2], Snehasis Banerjee[2], Arun Singh[3], K Madhava Krishna[1]

***Abstract*— Reinforcement Learning (RL) often requires large number of environment interactions to generalize to unseen in-distribution tasks, particularly when policy initialization is suboptimal. Existing meta-RL and transformer-based methods adapt to unseen tasks with few demonstrations but usually require training on many tasks (usually 85% of tasks in task distribution). To address this challenge, we propose a novel framework that leverages adversarial hypernetworks to generate strong policy initializations on unseen tasks, enabling rapid adaptation with minimal interactions, even when pre-trained on as minimum as 30% of tasks. We demonstrate the effectiveness of our approach on MuJoCo continuous control tasks, showcasing strong zero-shot policy initialization and rapid adaptation on unseen tasks. Additionally, we demonstrate that our framework can be extended to Multi-Task RL (MTRL) setting, where it outperforms existing hypernetwork based methods on manipulation tasks from MetaWorld benchmark. Through rigorous experimentation, we show that our framework outperforms the prior competitive baselines from in-context RL and meta RL on zero-shot transfer and enables efficient adaptation to unseen in-distribution tasks.**

Project page: https://GenHypernetworks.github.io/

I. INTRODUCTION

Robots deployed in factories and particularly in domestic environments such as houses are expected to perform wide variety of tasks and variations in tasks. Each of these tasks may vary in reward functions, robot-world interaction model or both, and its required to adapt to these variations in the tasks efficiently. Developing a framework that can adapt to tasks unseen during training time, preferably zero-shot and achieve optimal performance with minimal fine-tuning is essential, especially in the environments where unexpected changes may occur.

Existing approaches on efficient adaptation to unseen tasks can be broadly classified into two catgories. The first category includes meta-reinforcement learning (meta-RL) methods [1]–[3] which take an optimization based approach, where a meta-policy is trained to enable rapid adaptation to unseen tasks with minimal gradient updates to meta policy. The second category focuses on architectural innovations, such as those proposed in [4], [5]. Our work falls into the latter category, using hypernetwork-based architectural design to enable efficient adaptation with minimal online interactions. Hypernetworks are neural networks that generate the weights for another network for efficient adaptation to downstream tasks by conditioning them on task-specific context. While the prior hypernetwork-based approaches mainly focus on zero-shot generalization, they often require large number of training tasks and are prone to overfitting. This can lead to performance degradation on majority of unseen tasks when trained on limited number of tasks. Additionally, these methods overlook importance of efficient fine-tuning for adapting to unseen tasks.

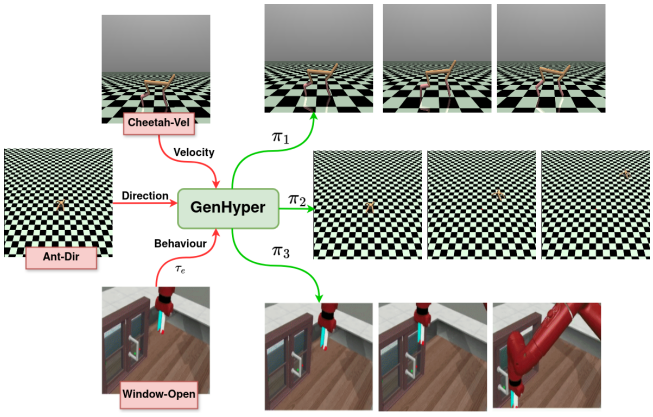

Fig. 1: GenHyper uses adversarial hypernetworks to generate strong policy initializations for unseen tasks by conditioning on task-specific information (e.g., velocity in Half-Cheetah), enabling rapid adaptation with minimal interactions.

To overcome these limitations, we propose **GenHyper**, a two-stage pipeline designed for efficient policy adaptation. In the first stage, we train a Generative Adversarial Network (HypLatent) to learn the space of policy weights for a set of tasks, effectively mapping Markov Decision Process (MDP) parameters to policy weights. At the inference time, we can sample from HypLatent by conditioning on MDP parameters to generate policy weights as shown in Figure 1. We show that these predictions can already achieve good zero-shot transfer across unseen in-distribution tasks. In the second stage, we aim to enhance the performance further using Reinforcement Learning based fine-tuning with minimal interactions. Specifically, in our implementation, we use Soft Actor-Critic (SAC) [6], wherein we fine-tune both the actor (initialized through HypLatent) and critic network (initialized randomly) using TD regularized fine-tuning. Our fine-tuned performance outperforms prior state-of-the-art transformer

* denotes equal contribution
[1] Robotics Research Center, IIIT Hyderabad, India. ramreddyai1010, sankethkalwar@gmail.com, mkrishna@iiit.ac.in
[2] TCS Research, India. b.bhowmick@tcs.com, snehasis.banerjee@tcs.com
[3] University of Tartu, Estonia. arun.singh@ut.ee

and hypernetwork-based methods, even when the HypLatent has been trained on policies from a limited number of prior related tasks.

To summarize, our key contributions are:

1) We propose **GenHyper**, a Generative hypernetwork based framework for efficient adaptation to unseen in-distribution tasks.
2) Through extensive experiments on MuJoCo continuous control tasks, we demonstrate that GenHyper effectively generalizes to unseen tasks, even when trained on a limited number of related tasks. Additionally, we show that our initialization on unseen tasks allows for sample-efficient fine-tuning to achieve near-optimal reward with minimal online interactions.
3) Our experiments on MetaWorld show that GenHyper can also be applied to multi-task reinforcement learning, surpassing existing hypernetwork-based methods in performance.

II. PRELIMINARIES:

A. Problem Formulation:

We consider an agent interacting with the environment that can perform various tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\} \sim \mathcal{P}(\tau)$ where $\mathcal{P}(\tau)$ is the task distribution and each task τ_i in $\mathcal{T}$ is modeled as MDP $\mathcal{M}_i = (S, A, \mu_0, R_i, P_i, \gamma)$ where S is the state space, A is the action space, μ_0 is the initial state distribution, and γ is the discount factor. The reward function R_i is parameterized by reward parameters (e.g., target velocity, movement direction of the agent) and the transition dynamics P_i are parameterized by physical parameters (e.g., torso length, height of the agent).

In this setting, a set of training tasks $\mathcal{T}_{train} = \{\tau_i\}_{i=1}^{N}$ are given from the task distribution $\mathcal{P}(\tau)$ and for each task τ_i, we have access to the expert data samples or policy. We specifically consider an offline setting similar to offline meta-RL [7], [8] where the learning algorithm has access to the data and is not allowed to interact with the environment during training. However, in contrast to the offline meta-RL setting, we make use of expert policies (SAC [6]) instead of replay buffer of transitions for each task τ_i. During the fine-tuning phase, an unseen task τ_{test} from the same task distribution $\mathcal{P}(\tau)$ is sampled and the goal is to learn a hypernetwork that can generalize or give the return close to best possible return on any τ_{test} with minimal interactions during the fine-tuning phase. We condition our framework on the MDP parameters $\mathcal{M}_i^{\text{params}} = (R_i, P_i)$, which encompass both reward parameters and dynamics parameters to generate policy parameters. These MDP parameters - $\mathcal{M}_i^{\text{params}}$ fully characterize the task variations in our experimental setting. This contrasts with offline meta-RL, where the policy must infer the test task from few-shot demonstrations and adapt.

III. RELATED WORKS

Meta Reinforcement Learning: Efficient policy adaptation to unseen tasks has been a key focus in meta-RL. Goal of meta-RL is to learn a meta policy such that it can achieve optimal policy on new unseen in-distribution task with few gradient updates. Prior approaches, such as MAML [1], learn a meta-policy by iterating over a task-specific inner loop for adaptation and an outer loop for updating the meta-policy parameters. However, this process is computationally expensive due to the reliance on second-order gradients. Several extensions have been proposed to improve efficiency and scalability such as [2], [3].

Multi-task Reinforcement Learning: Multi Task reinforcement learning (MTRL) is a paradigm which aims to learn a single policy that can perform multiple tasks. Natural way to solve MTRL is to have shared parameters which captures the common representations such as skills or the objects being manipulated among various tasks [9], [10]. Sharing the parameters across various tasks can lead to conflicts in gradients if the tasks are not aligned [11], [12]. This can lead to under performance on certain tasks. Over the years, many works have tackled this challenge by developing methods that manipulate task-specific gradients to enable efficient learning across multiple tasks [13]–[16], and is still an active area of research. On the other hand, CARE [16] proposes learning diverse representations—skills, behaviors, or objects through a mixture of encoders, combined using attention mechanism based on context such as language. MOORE [17] further enhances representation diversity with Gram-Schmidt orthogonalization. PACO [18] learns a policy subspace where task-specific policies are composed by interpolating the learned parameters. However, scaling to many tasks increases the learnable parameters.

Hypernetworks in Reinforcement-Learning Hypernetworks [19] have gained increasing attention over the years for their soft parameter sharing capabilities and have been applied across diverse domains [20]–[22] but are relatively less explored in the context of Reinforcement learning [5], [23]–[25]. Hypernetworks generate the weights for a target network, enabling it to adapt to specific tasks or contexts. This capability can be used to generate weights for the related tasks just by conditioning on task-specific information. Recent works, such as [5], [23], [25], [26], have demonstrated the potential of hypernetworks for efficient adaptation and multi-task learning. Leveraging their soft weight-sharing property, we aim to further explore the potential of hypernetworks in zero-shot generalization and efficient fine-tuning to unseen tasks, even when trained on limited number of tasks.

IV. PROPOSED METHOD

We propose a two-stage pipeline for efficient policy adaptation. At the first stage, which is the pretraining phase, we propose *HypLatent* (Fig.2(b)), an adversarial generative hypernetwork based on Adversarial autoencoder [27]. *HypLatent* takes the MDP parameters of a task as input and outputs diverse latent policy parameters ($\hat{Z}$) specific to that task as shown in the architecture Figure 2. In the second stage, we fine-tune the actor generated by HypLatent and critic initialized randomly and update them through online interactions using SAC by gradually increasing the actor's learning rate to allow the critic to adapt effectively.

In our framework, only Part (b) of the architecture is used for zero-shot generalization to unseen tasks, whereas Parts (a), (b), and (c) are used in the Multi-Task RL setting, as shown in Figure 3. We describe the extension of our framework to the Multi-Task RL setting later in Section V.

A. Stage 1: HypLatent Training for Policy Generation

We propose an end-to-end training framework for sampling diverse latent policy representations conditioned on the MDP parameters. HypLatent consists of 2 key components as shown in Figure 2.

Encoder and Decoder: Our architecture for the encoder and decoder is inspired by Make-An-Agent (MAA) [23]. Encoder E_ϕ takes policy parameters X from trained expert policies and maps them to a latent representation Z. The decoder D_θ reconstructs the original policy parameters from Z, ensuring that the learned latent space effectively captures task-specific policy distributions. The reconstruction loss is defined as:

$$\mathcal{L}_{recon}(X_i, D_\theta(E_\phi(X^i))) = \frac{1}{M}\sum_{i=1}^{M}(D_\theta)(E_\phi(X^i)) - X^i)^2 \tag{1}$$

where M is the number of training samples. The encoder output Z serves as the input to the generative model.

Generative Model: To generate diverse policies, we use a latent generative adversarial network conditioned on MDP parameters $\mathcal{M}_i^{\text{params}} = (R_i, P_i)$, which correspond to velocity for the Cheetah-Vel task and direction for the Ant-Dir task as shown in Figure 1. This conditioning differs in the Multi-Task RL setting, which we discuss in detail in Section V. The generator G_ζ takes as input $\mathcal{M}_i^{\text{params}}$ and a random noise vector n, and outputs a synthetic latent policy representation $\hat{Z}$, i.e $\hat{Z} = G_\zeta(\mathcal{M}_i^{\text{params}}, n)$ and the discriminator F_μ distinguishes real latent representations Z from generated ones $\hat{Z}$. Essentially, F_μ tries to predict whether the generated $\hat{Z}$ belong to real latent policy parameter manifold, i.e $p_Z = F_\mu(\hat{Z})$. For training generator G_ζ we minimize the objective function V_ζ below,

$$V_\zeta(G_\zeta, F_\mu) = \nabla_\zeta \frac{1}{N}\sum_{i=1}^{N}\log\left(1 - F_\mu(\hat{Z}^i)\right) + \lambda\mathcal{L}_{recon}(X_i, D_\theta(G_\zeta(\mathcal{M}_i^{\text{params}}, n))) \tag{2}$$

where $\mathcal{L}_{recon}(X_i, D_\theta(G_\zeta(\mathcal{M}_i^{\text{params}}, n)))$ is the reconstruction loss between the input policy parameters X_i and the generated (reconstructed) output policy parameters $D_\theta(G_\zeta(\hat{Z}))$, and λ is a weighting factor. Here, we train decoder D_θ together with generator G_ζ.

And, maximize the objective V_μ for training discriminator F_μ using following equation,

$$V_\mu(G_\zeta, F_\mu) = \nabla_\mu \frac{1}{N}\sum_{i=1}^{N}[\log\left(F_\mu(Z^i)\right) + \log\left(1 - F_\mu(\hat{Z}^i)\right)] \tag{3}$$

Algorithm 1 Training HypLatent

Require: Expert policies $\{X_i\}$, MDP parameters $\mathcal{M}_i^{\text{params}} = (R_i, P_i)$, learning rates η_G, η_F

Initialize generator G_ζ, discriminator F_μ, auxiliary network Q
for n = 1 to N epochs **do**
  for each $\mathcal{M}_i^{\text{params}}$ **do**
    $n \sim \mathcal{N}(0, I), \quad \hat{Z}_i = G_\zeta(\mathcal{M}_i^{\text{params}}, n)$
    $V_\mu = \log F_\mu(X_i) + \log(1 - F_\mu(\hat{X}_i))$
    $\mu \leftarrow \mu + \eta_F \nabla_\mu V_\mu$
    $V_\zeta = \log(1 - F_\mu(\hat{X}_i))$
    $\zeta \leftarrow \zeta - \eta_G \nabla_\zeta V_\zeta$
    $L_{\text{reg}} = \lambda_{\text{reg}}\left(Q(F_\mu(\hat{X}_i)) - \mathcal{M}_i^{\text{params}}\right)^2$
    $Q \leftarrow Q - \eta_F \nabla_Q L_{\text{reg}}$
  end for
end for

Note that encoder E_ϕ is updated together with discriminator. For further grounding the generated latent $\hat{Z}$, we regularize the generator and discriminator by reconstructing the input $\mathcal{M}_i^{\text{params}}$ back from the discriminator auxiliary head Q. We do this by minimizing L_{reg} in Equation (4).

$$L_{reg} = \frac{\lambda_{reg}}{N}\sum_{i=1}^{N}(Q(F_\mu(\hat{Z}^i)) - \mathcal{M}_i^{\text{params}})^2 \tag{4}$$

The training procedure is outlined in Algorithm 1,

B. Stage 2: TD-regularized Fine-tuning

After training HypLatent, the generator can now produce diverse latent policies conditioned on the MDP parameters of unseen tasks, offering strong initialization for new tasks. Our goal is to efficiently fine-tune these policies to optimal performance with minimal online interactions.

Overestimation of Q values: During online finetuning, we randomly initialize the critic network and finetune both actor and critic simualtanously using SAC update rule. However, the actor's performance degrades a lot initially due to distribution shift, a challenge commonly addressed in offline-to-online adaptation methods such as [28]. This issue arises because the critic is not initially well-aligned with the actor. To address this, we start with a lower learning rate for the actor, allowing the critic to adapt, and then gradually increase the actor's learning rate from 3×10^{-6} to 3×10^{-4} as training progresses. This stabilizes training.

The critic objective remains unchanged during fine-tuning, but we introduce an additional TD regularization loss term to optimize the actor, in cases of Q-value overestimation.

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\frac{1}{2}\left(Q_\theta(s,a) - y\right)^2\right],$$
$$y = r + \gamma(1-d)\left(\min_{i=1,2} Q_{\theta'}(s', a') - \alpha \log \pi_\phi(a'|s')\right),$$
$$\theta \leftarrow \theta - \lambda_Q \nabla_\theta J_Q(\theta). \tag{5}$$

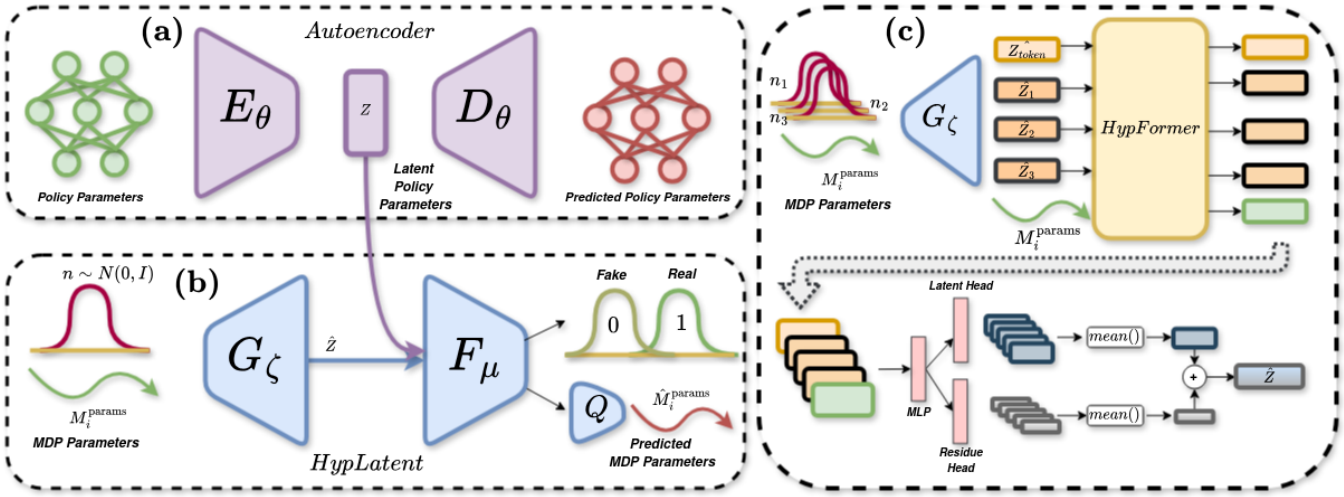

Fig. 2: **Architecture:** The figure in a section **(a)** consists of an *Autoencoder* which learns to map policy parameters to latent space. Section **(b)** shows $HypLatent$ architecture in which there is a Generator G_ζ and a Discriminator F_μ, objective of a generator is to learn latent policy parameter similar to the $Autoencoder's$ encoder output, given a behaviour embedding τ_e and noise $n \sim N(0, I)$, discriminator assesses whether the samples generated by the generator belong to the true latent policy parameter distribution, also there is a auxillary network Q which reconstructs the trajectory embedding given the output features of the discriminator, refer IV-A for in detail explanation. We also show $HypFormer$ in the section **(c)** where we use the trained generator to produce diverse latent policy parameters. For a given behavior embedding τ_e, multiple noise samples $n_1, n_2, n_3 \sim N(0, I)$ are used to generate multiple latent policy parameters. These are then processed by $HypFormer$, which predicts the ground truth latent policy parameters using the latent head and residue head. For a detailed explanation, see Section V.

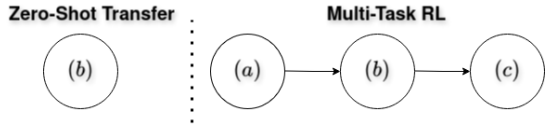

Fig. 3: **Framework Overview**. In zero-shot transfer experiments, only Part (b) of the architecture is used for direct policy generation from HypLatent, enabling zero-shot transfer to unseen tasks. In the Multi-Task RL setting, Parts (a), (b), and (c) are utilized as described in Section V.

where y is the TD target, r is the reward, and s' is the next state, while employing the standard practice of using two Q-functions to mitigate overestimation bias. The actor is updated to maximize the conservative Q-value while incorporating TD regularization as shown in the Figure 4:

$$\mathcal{L}_{\text{TD}} = \frac{1}{2}\delta^2, \quad \text{where } \delta = r + Q_{\theta'}(s', a') - Q_\theta(s, a)$$
$$J_\pi(\phi) = \mathbb{E}_{s\sim D, a\sim\pi_\phi}\left[\alpha \log \pi_\phi(a|s) - Q_\theta(s, a)\right] + \lambda_{\text{TD}}\mathcal{L}_{\text{TD}}$$
$$\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi) \quad (6)$$

where D is the replay buffer, and $\mathcal{L}_{\text{TD}}$ is the TD regularization term that prevents overestimation during fine-tuning. By integrating TD regularization, we ensure stability in policy improvement while mitigating value overestimation, leading to more reliable fine-tuning.

First the policies are compressed to the latent space as shown in the (Fig.2(a)) and we train HypLatent with latent policy parameters Z as shown in Figure 3. We can now use the generator G_ζ to generate latent policy parameter $\hat{Z}$ by conditioning on task information, which we refer to as the behavior embedding τ_e as shown in the Figure 2. The behavior embedding τ_e is used to capture information about the task. For MuJoCo tasks, the input to HypLatent is a simple scalar representing the task variable (e.g., target velocity). However, in multi-task RL benchmarks like MetaWorld, task information is more complex and cannot be represented by a single scalar. Instead, we train a behavior embedding τ_e to encode task-specific information.

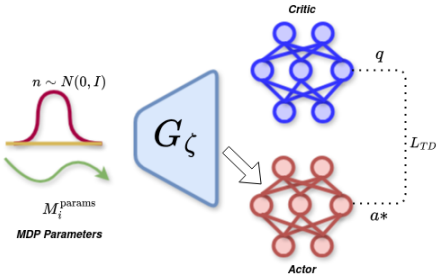

Fig. 4: TD-regularized fine-tuning process following Hyplatent generation. The critic is randomly initialized, and both the actor and critic are fine-tuned jointly, with a gradually increasing actor learning rate to stabilize training.

V. Extension to Multi-Task RL

A. Training of Behavior embedding

These embedding are used to provide conditional information to $HypLatent$, enabling the generation of diverse, task-specific policies. This approach is adapted from MakeAnAgent(MAA) [23]. Consider the trajectory of N steps, we now define n step trajectory τ^n : $(s_1, a_1, a_2, a_3, ..., s_{n-2}, a_{n-2}, a_{n-1}, a_n)$, and post success states $\hat{\tau}$: $(s_K, s_{K+1}, s_{K+2}, .., s_{K+M})$ which is collected

after the success step K till M fixed steps. The objective is to maximize the mutual information $\mathbb{I}(\hat{\tau}; \tau^n)$ between post success states and the n step trajectory. We train behaviour embedding using contrastive loss defined as shown in equation (7):

$$L_{behaviour} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{h_i^T W v_i}{\sum_{j=1}^{N} h_j^T W v_j} \tag{7}$$

where, $h_i = \phi(\tau_i^n)$ and $v_i = \psi(\hat{\tau}_i)$ along with learned similarity weights W between h_i and v_i which finally forms behaviour embedding $\tau_e = (h_i, v_i)$. We encourage readers to refer to MAA [23] for additional details on training.

B. HypFormer

Using G_ζ alone will generate diverse latent policy parameters, which might not be close to the ground truth latent policy embedding. This reduces success rate on an average. $HypFormer$ tackles this by using **soft-weighted aggregation** to refine the generated latent policy parameters toward the expert policy parameters, as detailed below.

In this stage, we use the generator G_ζ to generate latent policy parameters $S_{\hat{Z}} = (\hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \ldots, \hat{Z}_L)$ by conditioning it on the behavior embedding τ_e and sampled noise $(n_1, n_2, \ldots, n_L) \sim N(0, I)$. $S_{\hat{Z}}$ is then input to $HypFormer$ along with the learnable token $\hat{Z}_{token}$ and the behaviour embedding τ_e. The learnable token $\hat{Z}_{token}$ functions similarly to the $[CLS]$ token in the Vision Transformer [29], where it acts as a global representation of the input sequence. Now, $HypFormer$ takes the combination $(\hat{Z}_{\text{token}}, S_{\hat{Z}}, \tau_e)$ as input tokens, applies self-attention to produce enhanced latent policy parameters for each trajectory τ_e. This self-attention helps us to perform soft-weighted aggregation of latent policy parameters. Applying MSE loss to align $HypFormer$ predictions with ground truth resulted in unstable training. To address this, the enhanced tokens are processed by a shared MLP, which then splits into two branches: the latent Head and the residue Head. Latent Head learns to predict ground truth latent policy parameters, while the residue Head predicts the residue between Latent Head prediction and ground truth latent policy parameters, after which residue tokens and predicted policy tokens are averaged to get per trajectory single residue token ∇^i_{pred} and a single latent policy parameter token Z^i_{pred}. As we have ground truth latent policy parameter token Z^i_{gt} along with the residue token ∇^i_{gt} for each trajectory τ^i_e, We apply cosine similarity loss between Z^i_{gt} and Z^i_{pred}.

$$L_{sim} = \frac{1}{N}.\sum_{i=1}^{N}(1 - \frac{Z^i_{pred}.Z^i_{gt}}{max(||Z^i_{pred}||_2, ||Z^i_{gt}||_2, \epsilon)}) \tag{8}$$

Note that $Z^i_{gt} = \mathbb{E}_\theta(X^i_{gt})$, where X^i_{gt} is the expert SAC policy of the task corresponding to the behavior embedding τ^i_e. For residue token prediction we minimize L_{res} as follows,

$$L_{res} = \frac{1}{N}\sum_{i=1}^{N}(\nabla^i_{pred} - \nabla^i_{gt})^2 \tag{9}$$

Finally to ensure that residue head and latent head output's are consistent with each other consistency loss is applied on $\hat{Z}^i_{pred} = Z^i_{pred} + \nabla^i_{pred}$,

$$L_c = \frac{1}{N}\sum_{i=1}^{N}(\hat{Z}^i_{pred} - Z^i_{gt})^2 \tag{10}$$

So, final objective to minimize is as follows,

$$L_{HypFormer} = \lambda_{sim}L_{sim} + \lambda_{res}L_{res} + \lambda_c L_c \tag{11}$$

The size of L is typically a power of 2; in our case, it is set to 2^3 during training. N is the number of trajectories in the batch while training. λ_{sim}, λ_{res}, λ_c are all set to $1e3$.

VI. EXPERIMENTAL SETUP

In our experiments, we aim to evaluate and answer the following: 1.) Does our method zero-shot generalize to related but unseen tasks which are in-distribution? How does the quality of zero-shot transfer change when trained on a smaller set of tasks? 2.) How data efficient is fine-tuning with hypernetwork-based policy initialization for unseen tasks, particularly when HypLatent is trained on a limited number of tasks? 3.) How well does our method perform on the set of seen tasks in a multi-task RL setting?

A. Environment details

We evaluate zero-shot transfer and fine-tuning capabilities on two MuJoCo control tasks:

1.) **Cheetah-Vel & Ant-Dir**: These tasks involve achieving target velocities in $[0, 3]$ (Cheetah-Vel) or moving in a goal direction in $[0, 2\pi]$ (Ant-Dir). For Cheetah-Vel, we train on 35 tasks with 5 held out for testing, while for Ant-Dir, we use 45 training tasks with 5 for testing. We conduct ablations by training the HypLatent on fewer tasks (Cheetah-Vel: 10, 20, 25; Ant-Dir: 15, 25, 35) to assess generalization and sample efficiency.

We assess Multi-Task RL (MTRL) performance on the MetaWorld benchmark [30]:

2.) **MetaWorld**: MetaWorld is a benchmark suite for robotic tabletop manipulation, where the Sawyer robotic arm executes diverse motions and object interactions. We consider 8 training (seen) tasks as shown in Table II and 8 evaluation (unseen) tasks as shown in Table IV, assessing GenHyper's success rate on seen tasks and its zero-shot performance on unseen tasks.

B. Dataset collection

For MuJoCo tasks, we collect a dataset of 500 expert policies and corresponding critics for each training task to train our hypernetworks for actor and critic. This dataset is generated by training Soft Actor-Critic (SAC) [6], with the first policy and critic saved at 500k training steps, followed by checkpoints recorded every 500 steps thereafter. Hyperparameters for SAC policies are borrowed from CleanRL [31].

For MetaWorld, once the test success rate reaches 1, we collect policies every 500 steps, gathering a total of 500 SAC policies per task.

C. Training Setup

We train the $HypLatent$ using a 1D UNet architecture [32] for the generator and a 3-layer MLP for the discriminator. For regularization, an auxiliary network consisting of a 2-layer MLP is used. The noise sample size is set to 128 for MetaWorld tasks and 4 for Half-Cheetah. The behavior embedding size is 128 for MetaWorld, while for Half-Cheetah, the conditioning information is represented by a single scalar value (velocity).

We utilize a single-layer Transformer [33] encoder without positional encoding, with a token size of 256 and 128 heads in the multi-head attention layer.

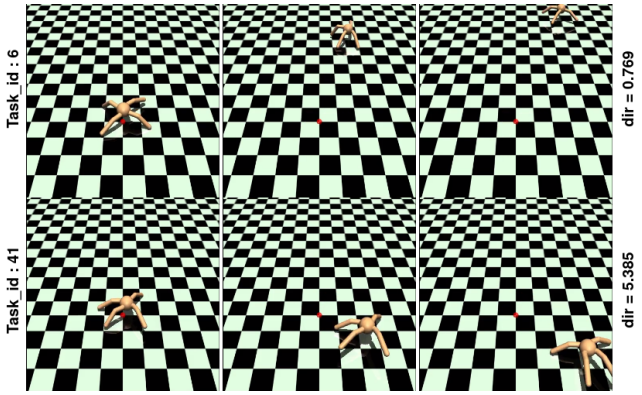

Fig. 5: Zero-shot rollout of HypLatent trained on only 15 out of 50 tasks, evaluated on unseen task IDs (6, 41) in Ant-dir task of MuJoCo. Achieved rewards, averaged across 3 seeds, are 480.56 and 437.36, respectively. Direction input to HypLatent is in radians.

D. Baselines

Prompt Decision Transformer (PDT) [4]: Prompt-based Decision Transformer for offline few-shot RL. We use it as a baseline to assess our framework's zero-shot generalization without additional fine-tuning.

MACAW [7]: A sample efficient offline meta-RL algorithm. We compare our framework's zero-shot transfer with MACAW's online fine-tuning upto 20k iterations as it is a meta-RL approach.

PEARL [34]: PEARL is an off-policy meta-RL algorithm that infers task context via variational inference. Similar to MACAW, we compare our framework's zero-shot transfer with its online fine-tuning upto 20k iterations.

VII. RESULTS AND ANALYSES

A. Zero-shot generalization results

We quantitatively evaluate our framework on the Cheetah-Vel and Ant-dir datasets to demonstrate its effectiveness in zero-shot generalization to unseen tasks by varying the reward functions. For Cheetah-Vel, episodic returns are averaged across the test task indices - (2, 7, 15, 23, 26) for 5 different seeds. Similarly for Ant-Dir, returns are averaged across the test task indices - (6, 17, 23, 30, 41) for 5 different seeds, as shown in Table I. Qualitative results for Ant-Dir task on two random unseen tasks are shown in the Figure 5. These results demonstrate that HypLatent enables strong zero-shot transfer to unseen tasks surpassing the prior competitive baselines.

TABLE I: Comparing zero-shot transfer of HypLatent to baselines on Cheetah-Vel and Ant-Dir tasks.

Methods	Avg Returns (Cheetah-Vel)	Avg Returns (Ant-Dir)
Offline PEARL [34] (Iter. 0)	-273.6	-135.3
Offline PEARL [34] (Iter. 20K)	-135.3	123.9
MACAW [7] (Iter. 0)	-121.6	251.9
MACAW [7] (Iter. 20K)	-60.5	376.5
Prompt-DT [4]	**-34.43**	409.81
HypLatent (Ours) (15 training tasks)	-40.33	452.32
HypLatent (Ours) (25 training tasks)	-37.05	478.45
HypLatent (Ours) (35 training tasks)	-34.84	496.65
HypLatent (Ours) (45 training tasks)	N/A	**505.64**

B. Fine-tuning results on unseen tasks

After obtaining policy initializations from HypLatent for unseen tasks, we finetune those policies as described in Section IV-B. Figure 6 shows the fine-tuning plots and reward curves for two unseen tasks in both Ant-Dir and Cheetah-Vel. Notably, for Ant-Dir, our approach achieves a reward of 600 within 100k iterations, whereas SAC reaches 400 even after 200k iterations. Similarly, for Cheetah-Vel, we reach near-optimal rewards within just 50k iterations, while SAC requires more than 100k iterations to achieve comparable performance. The shaded regions in the plot represent the standard deviation across three different seeds.

Fig. 6: Fine-tuning performance on unseen tasks in Ant-Dir and Cheetah-Vel benchmarks. (N_train:x) indicates that HypLatent is trained on x number of tasks (please zoom in for better clarity)

Figure 6 also shows ablations on HypLatent trained on 15, 25, 35 tasks. Even when HypLatent is trained on only 15 tasks, it achieves competitive zero-shot performance and efficiently fine-tunes policies on unseen tasks as shown in the above figure. The initialization from HypLatent trained on 35 tasks adapts faster to unseen tasks compared to those trained on 15 and 25 tasks.

C. Multi-Task RL results

For the MetaWorld dataset, the success rate is measured by the proportion of the task completed given a trajectory which is applicable to both seen and unseen tasks. Specifically, we assess our method's performance in MTRL by evaluating it

on the seen MetaWorld tasks and then testing the zero-shot generalization capability of the learned representations on the unseen MetaWorld tasks. We show the comparison of our method directly with MAA as it is the closest method to our approach. Our framework GenHyper achieves a 34.3% improvement over MAA and GenHyper without $HypFormer$ outperforms by 15% on MetaWorld seen test tasks. Table III shows that our top-5 and top-10 generated policies achieve 100% success rate on seen test environments. For more broad comparison, we have compared against CARE [16], Decision Transformer DT [35]. Table IV shows the success rate on completely unseen tasks of MetaWorld.

TABLE II: GenHyper Success Rate(%) on the MetaWorld dataset. **Bold** numbers highlights the top achieved success-rate on the task, while the *italics* shows the 2nd best achieved success-rate.

MTRL Tasks	MAA	GenHyper (w/o HypFormer)	GenHyper (w/ HypFormer)
window-open	33	*51*	**64**
door-open	27	*35*	**62**
drawer-open	*42*	40	**78**
dial-turn	23	*36*	**48**
plate-slide	45	*66*	**88**
button-press	32	*38*	**58**
handle-press	50	*62*	**82**
faucet-close	45	*77*	**82**
Avg. Success Rate	36	*51*	**70.3**

TABLE III: GenHyper Success Rate(%) on MetaWorld dataset. **Bold** numbers highlights the top achieved success-rate, while the *italics* shows the 2nd best achieved success-rate. Unless explicitly stated otherwise, such as for top-10 or top-5, the success rate reported in table below represents the average over 100 policies.

Methods	Seen Task	Unseen Task
CARE [16]	*82.1*	58.5
DT [35]	80.3	60.4
MAA [23]	36	16.25
GenHyper w/o HypFormer	51	12.13
GenHyper	70.3	19.8
MAA (top 5) [23]	95.88	**88**
MAA (top 10) [23]	90.88	77
GenHyper w/o HypFormer (top 10)	**100**	54.38
GenHyper w HypFormer (top 10)	**100**	*80.63*
GenHyper w/o HypFormer (top 5)	**100**	75.1
GenHyper w HypFormer (top 5)	**100**	87.5

TABLE IV: Success Rate(%) on unseen tasks of MetaWorld. **Bold** numbers highlights the top achieved success-rate on the task, while the *italics* shows the 2nd best achieved success-rate.

Zero-Shot RL Tasks	MAA	GenHyper (w/o HypFormer)	GenHyper (w/ HypFormer)
drawer-close	*55*	53	**80**
handle-press-side	*4*	**6**	0
door-lock	**13**	*6*	*6*
window-close	*10*	0	**12**
reach-wall	**13**	6	*10*
coffee-button	*8*	3	**9**
button-press-wall	*11*	2	**14**
faucet-open	*16*	21	**27**
Avg.Success Rate	*16.25*	12.13	**19.8**

D. Ablation Studies

We present design decisions of GenHyper framework in the context of MetaWorld tasks. We demonstrate the impact of number of tokens on task performance and provide PCA analysis to further substantiate our design choices.

a) Varying number of Tokens: We train $HypFormer$ using 8 tokens, which results in the best performance on the seen MTRL tasks. This is demonstrated in Figure 8, where the left image with the blue bar graph highlights this setup. As number of tokens increases, the performance on MTRL tasks decreases asymptotically due to increase in redundant latent embeddings which makes it harder to attend to latent policy embeddings closer to the ground truth embedding Z^i_{gt}.

b) PCA Analysis: We present a PCA analysis on the latent policy parameters generated by $HypLatent$ and $HypFormer$, as illustrated in Figure 7 for three seen tasks from MetaWorld: 'Button-Press,' 'Dial-Turn,' and 'Door-Open.' In the 2D PCA plots, the latent policy parameters predicted by $HypFormer$ are more closely aligned with the ground truth compared to those generated by $HypLatent$, demonstrating the effectiveness of $HypFormer$.

VIII. CONCLUSION

In this paper, we propose a framework for efficient adaptation to unseen in-distribution tasks by leveraging adversarial hypernetworks to train a diverse prior over task policies. Our experiments on MuJoCo control tasks demonstrate that the framework exhibits strong zero-shot generalization to unseen tasks even when trained on limited number of tasks and outperforms prior competitive baselines on zero-shot transfer. We show that policy initialization from HypLatent enables sample-efficient fine-tuning on unseen tasks, rapidly achieving near-optimal reward with minimal online interactions. We further extend our framework to Multi-task RL setting where we use a single layer transformer architecture to guide the policy priors from HypLatent towards expert policy parameters. Our framework outperforms related hypernetwork-based baselines in Multi-task RL setting.

IX. FUTURE WORK

We currently rely on expert policies during training, making it crucial to explore performance when training with suboptimal policies. Second, minimizing the impact of distribution shift problem in offline to online learning during fine-tuning remains a key challenge. Another promising future direction is to apply GenHyper to larger variety of tasks in the real world setting.

REFERENCES

[1] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*, pp. 1126–1135, PMLR, 2017. 1, 2

[2] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," *Advances in neural information processing systems*, vol. 32, 2019. 1, 2

[3] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "Promp: Proximal meta-policy search," *arXiv preprint arXiv:1810.06784*, 2018. 1, 2

[4] M. Xu, Y. Shen, S. Zhang, Y. Lu, D. Zhao, J. Tenenbaum, and C. Gan, "Prompting decision transformer for few-shot policy generalization," in *international conference on machine learning*, pp. 24631–24645, PMLR, 2022. 1, 6

[5] S. Rezaei-Shoshtari, C. Morissette, F. R. Hogan, G. Dudek, and D. Meger, "Hypernetworks for zero-shot transfer in reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 9579–9587, 2023. 1, 2

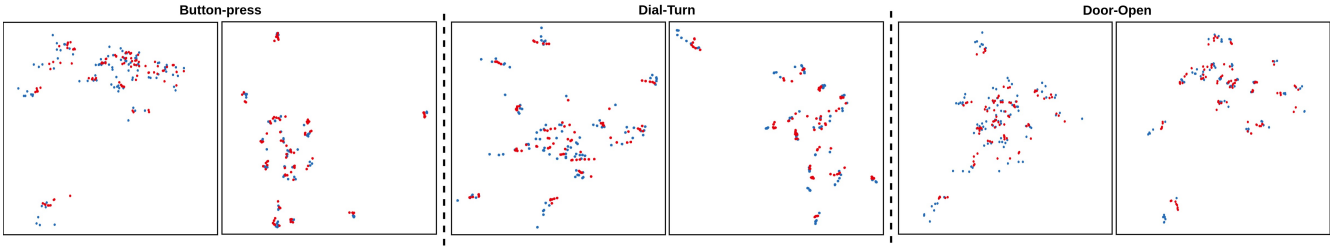

Fig. 7: Visualization of the predicted latent policy parameters for three MetaWorld tasks. In each task, the image on the left represents the output of the $HypLatent$ generator, while the image on the right shows the output of $HypFormer$. The red color points are the predicted latent policy parameters while the blue color points indicate the ground-truth latent policy parameters.

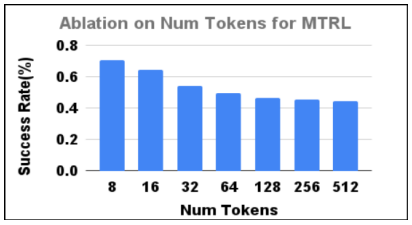

Fig. 8: Performance of GenHyper on MetaWorld MTRL seen tasks varying token size from 8 to 512. Success rate is normalized to 1.0

[6] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018. 1, 2, 5

[7] E. Mitchell, R. Rafailov, X. B. Peng, S. Levine, and C. Finn, "Offline meta-reinforcement learning with advantage weighting," in *International Conference on Machine Learning*, pp. 7780–7791, PMLR, 2021. 2, 6

[8] V. H. Pong, A. V. Nair, L. M. Smith, C. Huang, and S. Levine, "Offline meta-reinforcement learning with online self-supervision," in *International Conference on Machine Learning*, pp. 17811–17829, PMLR, 2022. 2

[9] C. D'Eramo, D. Tateo, A. Bonarini, M. Restelli, and J. Peters, "Sharing knowledge in multi-task deep reinforcement learning," *arXiv preprint arXiv:2401.09561*, 2024. 2

[10] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4767–4777, 2020. 2

[11] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International conference on machine learning*, pp. 794–803, PMLR, 2018. 2

[12] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7482–7491, 2018. 2

[13] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018. 2

[14] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020. 2

[15] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, "Conflict-averse gradient descent for multi-task learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18878–18890, 2021. 2

[16] S. Sodhani, A. Zhang, and J. Pineau, "Multi-task reinforcement learning with context-based representations," in *International Conference on Machine Learning*, pp. 9767–9779, PMLR, 2021. 2, 7

[17] A. Hendawy, J. Peters, and C. D'Eramo, "Multi-task reinforcement learning with mixture of orthogonal experts," *arXiv preprint arXiv:2311.11385*, 2023. 2

[18] L. Sun, H. Zhang, W. Xu, and M. Tomizuka, "Paco: Parameter-compositional multi-task reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 21495–21507, 2022. 2

[19] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016. 2

[20] R. K. Mahabadi, S. Ruder, M. Dehghani, and J. Henderson, "Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks," *arXiv preprint arXiv:2106.04489*, 2021. 2

[21] J. Von Oswald, C. Henning, B. F. Grewe, and J. Sacramento, "Continual learning with hypernetworks," *arXiv preprint arXiv:1906.00695*, 2019. 2

[22] N. Ruiz, Y. Li, V. Jampani, W. Wei, T. Hou, Y. Pritch, N. Wadhwa, M. Rubinstein, and K. Aberman, "Hyperdreambooth: Hypernetworks for fast personalization of text-to-image models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6527–6536, 2024. 2

[23] Y. Liang, T. Xu, K. Hu, G. Jiang, F. Huang, and H. Xu, "Make-an-agent: A generalizable policy network generator with behavior-prompted diffusion," *arXiv preprint arXiv:2407.10973*, 2024. 2, 3, 4, 5, 7

[24] D. Zhao, S. Kobayashi, J. Sacramento, and J. von Oswald, "Meta-learning via hypernetworks," in *4th Workshop on Meta-Learning at NeurIPS 2020 (MetaLearn 2020)*, NeurIPS, 2020. 2

[25] J. Beck, M. T. Jackson, R. Vuorio, and S. Whiteson, "Hypernetworks in meta-reinforcement learning," in *Conference on Robot Learning*, pp. 1478–1487, PMLR, 2023. 2

[26] J. Beck, R. Vuorio, Z. Xiong, and S. Whiteson, "Recurrent hypernetworks are surprisingly strong in meta-rl," *Advances in Neural Information Processing Systems*, vol. 36, pp. 62121–62138, 2023. 2

[27] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015. 2

[28] Z. Yu and X. Zhang, "Actor-critic alignment for offline-to-online reinforcement learning," in *International Conference on Machine Learning*, pp. 40452–40474, PMLR, 2023. 3

[29] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020. 5

[30] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," in *Conference on robot learning*, pp. 1094–1100, PMLR, 2020. 5

[31] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. AraÃšjo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. 5

[32] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241, Springer, 2015. 6

[33] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017. 6

[34] 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*, pp. 5331–5340, PMLR, 2019. 6

[35] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, "Decision transformer: Reinforcement learning via sequence modeling," *Advances in neural information processing systems*, vol. 34, pp. 15084–15097, 2021. 7