AMT-Hyp : Adversarial Multi-Task Reinforcement Learning Framework for Policy Generation using Hypernetworks

Sanket Kalwar*[1], Jayaram Reddy*[1], Brojeshwar Bhowmick[2], Arun Singh[3], K Madhava Krishna[1]

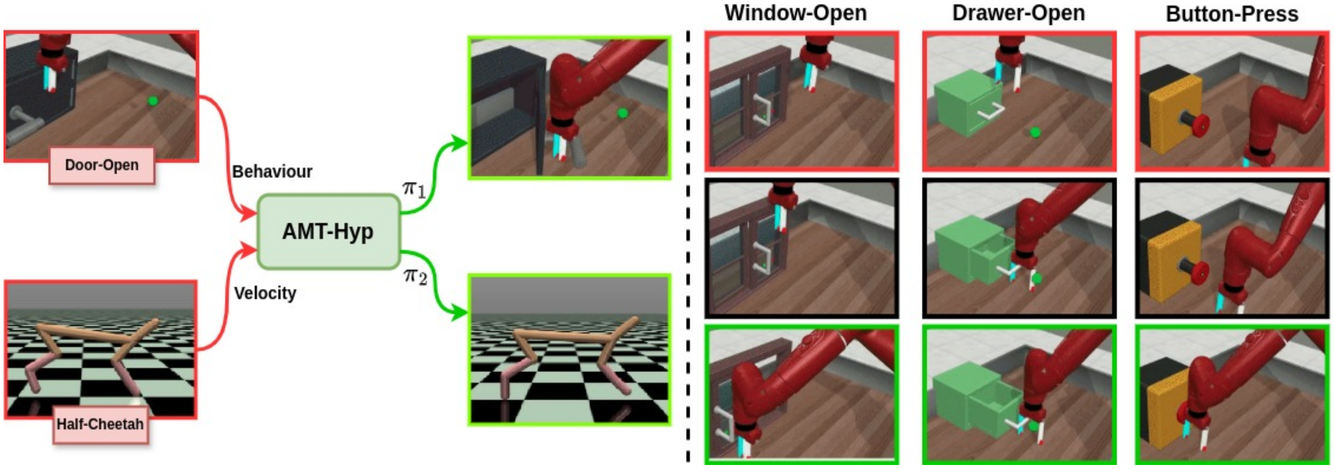

Fig. 1: **Overview**: We present AMT-Hyp, a framework for directly generating policies π using conditioning information (Example: Behavior embeddings for MetaWorld and Velocity inputs for Half-Cheetah). These generated policies can be directly deployed to successfully accomplish the task. On the right side are the directly deployed policies generated by AMT-Hyp on the MetaWorld tasks.

***Abstract*— A key challenge in building generalist agents is enabling them to perform multiple tasks while simultaneously adapting to variations across the tasks efficiently, particularly in a zero-shot manner. Multi-task Reinforcement Learning (MTRL) is a paradigm that enables agents to learn a single policy that can be deployed to perform multiple tasks in a given environment. A straightforward approach like parameter sharing introduces challenges such as conflicting gradients and determining the optimal way to distribute shared parameters across tasks. In this work, we introduce AMT-Hyp, a framework for training hypernetworks in MTRL that consists of HypLatent, an adversarial autoencoder that generates diverse task-conditioned latent policy parameters, and HypFormer, a single-layer transformer that performs soft-weighted aggregation on these priors towards expert policy parameters. Our approach not only outperforms previous hypernetwork based methods but also performs comparably to the existing state-of-the-art methods in MTRL on MetaWorld benchmark. Additionally, experiments on MuJoCo continuous control tasks demonstrate the framework's strong zero-shot learning capabilities, allowing it to generalize to unseen in-distribution tasks without additional fine-tuning. Our framework also achieves performance comparable to state-of-the-art offline meta-RL methods.**
Project page: https://amt-hyp.github.io/

I. INTRODUCTION

Robots deployed in factories and particularly in domestic environments such as houses are expected to perform wide variety of tasks. Each of these tasks may vary in reward functions, dynamics or both and its required to adapt to these variations in the tasks. Developing a framework which can perform multiple tasks while generalizing to variations in these tasks is crucial especially in the environments where changes can arise unexpectedly.

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 [1], [2]. Sharing the parameters across various tasks can lead to conflicts in gradients if the tasks are not aligned [3]–[5]. 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 [6]–[11], and is still an active area of research. On the other hand, CARE [12] proposes learning diverse representations—skills, behaviors, or objects through a mixture of encoders, combined using attention mechanism based on context such as language. MOORE [13] further enhances representation diversity with Gram-Schmidt orthogonalization. PACO [14] 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 [15] have gained increasing attention over the years for their soft parameter sharing capabilities and have been applied across diverse domains [16]–[20] but are relatively less explored in the context of Reinforcement learning [21]–[24]. Hypernetworks generate the weights for

* denotes equal contribution
[1]Robotics Research Center, IIIT Hyderabad, India {sankethkalwar, ramreddyai1010}@gmail.com, mkrishna@iiit.ac.in
[2]TCS Research, India b.bhowmick@tcs.com
[3]University of Tartu, Estonia arun.singh@ut.ee
[†]Code and Website: https://amt-hyp.github.io/

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. Leveraging their soft weight-sharing property, we aim to explore the potential of hypernetworks in MTRL and assess their zero-shot generalization capabilities.

To summarize, our key contributions are:

1) We propose **AMT-Hyp**, a framework for training hypernetworks for MTRL. It consists of:
   - **HypLatent**, an adversarial autoencoder which learns to generate **diverse** latent policy parameters conditioned on the task information.
   - **HypFormer** is a single-layer transformer network that performs **soft-weighted aggregation** on the prior generated by $HypLatent$, refining it towards expert policy parameters.
2) We regularize hypernetworks by reconstructing task embeddings. Here, the hypernetwork generates task-specific weights while a discriminator ensures consistency with the task distribution. This method improves generalization by encouraging coherent and adaptable embeddings across tasks.
3) Experiments in MuJoCo control tasks shows that our framework has impressive zero-shot generalization capabilities on unseen tasks that are in distribution.

II. PRELIMINARIES:

The reinforcement Learning task is defined as a Markov Decision Process (MDP) $\mathcal{M}$ which is modeled as a tuple $(S, A, \mu_0, R, P, \gamma)$, where S is the state space, A is the action space, $\mu_0 : S \rightarrow [0, 1]$ is the initial state distribution, $R : S \times A \rightarrow \mathbb{R}$ is the reward function, $P : S \times A \times S \rightarrow S$ is the transition probability function where $P(s, a, s')$ is the probability of reaching state s' from state s by taking action a, and $\gamma \in [0, 1)$ is the discount factor. At any time t starting from the initial time, the agent observes a state $s_t \in S$, takes an action $a_t \in A$, transitions to a new state $s_{t+1} \in S$ according to P and gets a reward $r_t = r(s_t, a_t) \sim R$. The objective of the agent is to optimize policy $\pi_\theta(a_t \mid s_t)$ by maximizing the expected discounted return $J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$ where $s_0 \sim \mu_0$.

A. Multi-Task Reinforcement-Learning:

We consider an agent interacting with the environment to 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 $\mathcal{M}_i$. MTRL can be formalized as Block Contextual Markov Decision Process BC-MDP [12], [25] $\langle C, S, A, M' \rangle$ where C is the context space that is shared across all the tasks in $\mathcal{T}$ and each element in C denotes a specific task and its related components, S is the universal state space shared across the tasks, A is the shared action space, M' maps $c \in C$ to the MDP parameters $M'(c) = \{R_c, T_c, S_c\}$ where S_c is the state space associated with the context c. An example of such a scenario is the MetaWorld benchmark [26], where the state space S_c associated with each context c varies due to differences in behaviors and objects manipulated across tasks. A more simpler setting is the case where only the reward function changes across the tasks but the transition dynamics remains the same, MuJoCo [27] continuous control tasks such as *Cheetah-vel*, *Cheetah-dir*, *Ant-dir* [28], [29] are few examples in this scenario.

MTRL solves multiple MDP's associated with the tasks from $\mathcal{T}$ at once, and the objective is to learn a single policy π_{MTRL} that maximizes the summation of expected discounted return $J_{MTRL}(\pi) = \sum_{i=1}^{N} J_i$ for all the tasks in $\mathcal{T}$. We assume that the tasks in $\mathcal{T}$ share a common universal state space $\mathcal{S}$, with consistent dimensionality across all tasks. However, each individual task $\mathcal{T}_i$ may have a unique, disjoint state space $\mathcal{S}_i$, which depends on the specific objects or features relevant to that task.

B. Zero-Shot Generalization in Reinforcement-Learning:

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 [30], [31] where the learning algorithm has access to the data and is not allowed to interact with the environment during training. In contrast to the offline meta-RL setting, we make use of fixed number of expert policies(SAC [32]) instead of replay buffer of transitions for each task τ_i. During the testing phase, an unseen task τ_{test} from the same task distribution $\mathcal{P}(\tau)$ is sampled and the goal is to learn a policy on $\mathcal{T}_{train}$ i.e during training phase which can generalize or give the return close to best possible return on any τ_{test} without additional finetuning unlike offline meta-RL where policy can be adapted during the testing phase. We condition our framework with privileged information about the test task, such as the target velocity the agent needs to achieve in the *Cheetah-vel* task. This contrasts with offline meta-RL, where the policy must infer the test task from few-shot demonstrations and adapt accordingly.

III. PROPOSED METHOD

We propose a three-stage pipeline as shown in Fig.2 . In the first stage, an autoencoder [33] maps policy parameters to the latent space, whose latent features are then used in the second stage, $HypLatent$ which is based on Adversarial autoencoder [34]. $HypLatent$ seeks to approximate the autoencoder's latent distribution through adversarial training. The generator aims to map behavior embeddings and noise sampled from a normal distribution, where the inclusion of noise enables the generator to learn diverse latent features. These diverse features are then utilized in the third stage, $HypFormer$. $HypFormer$ uses the diverse latent features generated by $HypLatent$ for each behavior embedding that performs soft-weighted aggregation and refines them towards expert policy parameters. A transformer encoder applies self-attention across both the latent features and behavior embeddings, grounding the latent features through behavior embedding interactions. Ultimately, the latent features are guided by learned residuals. The subsequent sections provide detailed discussions on the autoencoder training, behavior embeddings, $HypLatent$, and $HypFormer$.

A. Stage 1: Autoencoder and Behaviour Embedding:

Autoencoder: Takes trained SAC policies of multiple tasks as input, mapping them to a latent space. These policies are randomly sampled from various tasks during training and passed through autoencoder, which learns a latent space that captures the distribution of task-specific policies. It consists of encoder network E_θ which takes the policy parameters X as input and outputs latent representation of the policy parameters Z, and a decoder network D_θ which takes the output latent Z from the encoder E_θ and reconstructs the policy parameters $\hat{X} = D_\theta(E_\theta(X))$. The objective function for training autoencoder is shown in equation (1).

$$L_{autoencoder} = \frac{\lambda}{M}\sum_{i=1}^{M}(\hat{X}^i - X^i)^2 \tag{1}$$

where M is the number of training samples for SAC policies, and λ is the scaling factor set to $1e3$. This will learn the distribution of policy parameters effectively and the encoder's output is used in Stage-2 by $HypLatent$ during training to generate diverse samples of latents.

Behaviour 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) [22]. Consider the trajectory of N steps, we now define n step trajectory $\tau^n \ : \ (s_1, a_1, a_2, a_3, ..., s_{n-2}, a_{n-2}, a_{n-1}, a_n)$, 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 (2):

$$L_{behaviour} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{h_i^T W v_i}{\sum_{j=1}^{N} h_j^T W v_j} \tag{2}$$

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 $\tau_e = (h_i, v_i)$.

After retrieving latent policy parameter Z from autoencoder and behaviour embedding τ_e, both are used in the stage-2 for training *HypLatent* network.

B. Stage 2: $HypLatent$ Generative Adversarial Network

We use the $HypLatent$ to learn the latent manifold Z from the autoencoder's encoder output. This is conditioned on the behavior embedding τ_e and a sampled noise $n \sim N(0, I)$. The network consists of generator G_ζ which is conditioned on τ_e along with sampled noise n to generate $\hat{Z}$, i.e $\hat{Z} = G_\zeta(\tau_e, n)$, and a discriminator 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 G_ζ we minimize the objective function V_ζ below,

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

And, maximize the objective V_μ for F_μ,

$$V_\mu(G_\zeta, F_\mu) = \nabla_\mu \frac{1}{N}\sum_{i=1}^{N}[\log\left(F_\mu(Z^i)\right) + \log\left(1 - F_\mu(\hat{Z}^i)\right)] \tag{4}$$

For further grounding the generated latent $\hat{Z}$, we regularize the generator and discriminator by reconstructing the input τ_e back from the discriminator auxiliary head Q. We do this by minimizing L_{reg} in Equation (5).

$$L_{reg} = \frac{\lambda_{reg}}{N}\sum_{i=1}^{N}(Q(F_\mu(\hat{Z}^i)) - \tau_e^i)^2 \tag{5}$$

Now, we use the generator G_ζ to generate latent policy parameter $\hat{Z}$. Using G_ζ alone will generate diverse latent policy parameters, resulting in various behaviors, meaning different ways of performing the same task but won't guarantee high 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 in the section below.

C. Stage 3: HypFormer

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 trajectory 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. 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 ∇_{pred}^i and a single latent policy parameter token Z_{pred}^i. As we have ground truth latent policy parameter token Z_{gt}^i along with the residue token ∇_{gt}^i for each trajectory τ_e^i, We apply cosine similarity loss between Z_{gt}^i and Z_{pred}^i.

$$L_{sim} = \frac{1}{N}.\sum_{i=1}^{N}(1 - \frac{Z_{pred}^i . Z_{gt}^i}{max(||Z_{pred}^i||_2, ||Z_{gt}^i||_2, \epsilon)}) \tag{6}$$

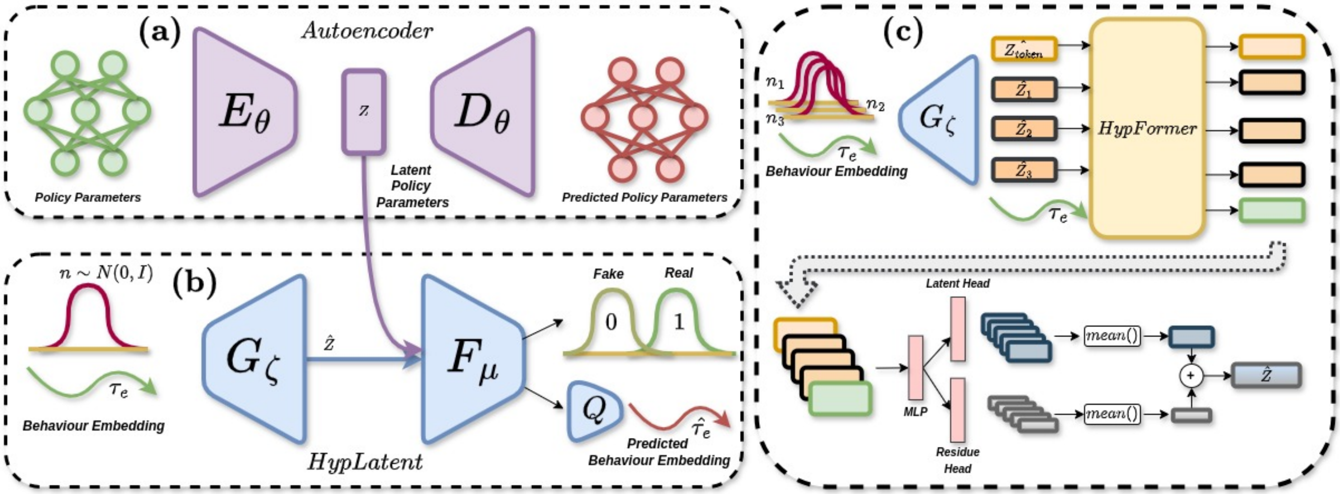

Fig. 2: **Overview:** The figure in a section **(a)** consists of an *Autoencoder* which learns to map policy parameters to latent space, more details is discussed at III-A. Section **(b)** shows $HypLatent$ architecture in which there is a Generator G_ζ and a Discriminator F_μ, objective of a generator 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 III-B 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 III-C.

For residue token prediction we minimize L_{res} as follows,

$$L_{res} = \frac{1}{N} \sum_{i=1}^{N} (\nabla^i_{pred} - \nabla^i_{gt})^2 \tag{7}$$

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

$$L_c = \frac{1}{N} \sum_{i=1}^{N} (\hat{Z}^i_{pred} - Z^i_{gt})^2 \tag{8}$$

So, final objective to minimize is as follows,

$$L_{HypFormer} = \lambda_{sim} L_{sim} + \lambda_{res} L_{res} + \lambda_c L_c \tag{9}$$

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 $1e3$.

IV. EXPERIMENTAL SETUP

In our experiments, we aim to evaluate and answer the following: 1.) Performance of our method in Multi-task Reinforcement Learning (MTRL) on seen tasks. 2.) Does our method zero-shot generalize to related but unseen tasks which are in-distribution? 3.) How well do the representations learned by the hypernetwork perform on entirely unseen tasks, specifically in terms of out-of-distribution task performance in MTRL?

A. Datasets and Environments details

We evaluate MTRL performance on MetaWorld [26] and zero-shot generalization capabilities on MuJoCo control tasks.
1.) **MetaWorld:** The MAA (MakeAnAgent) splits are utilized for training and evaluating MTRL performance on both seen and unseen tasks. Dataset is sourced from MAA.
2.) **Cheetah-Vel:** Following [30], [35], the task involves achieving target velocities sampled from [0-3], with 35 velocities in the training set and 5 in the test set. Ablation studies are conducted on training with 10, 20, and 25 random velocities, while the test set remains unchanged, demonstrating the framework's generalization and sample efficiency. We collect a dataset of 800 expert policies for each training velocity by training SAC [32]. The first policy is saved at 150k training steps, followed by checkpoints taken every 500 training steps thereafter.

B. Training Setup

We train the $HypLatent$ using a 1D UNet architecture [36] 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 [37] encoder without positional encoding, with a token size of 256 and 128 heads in the multi-head attention layer.

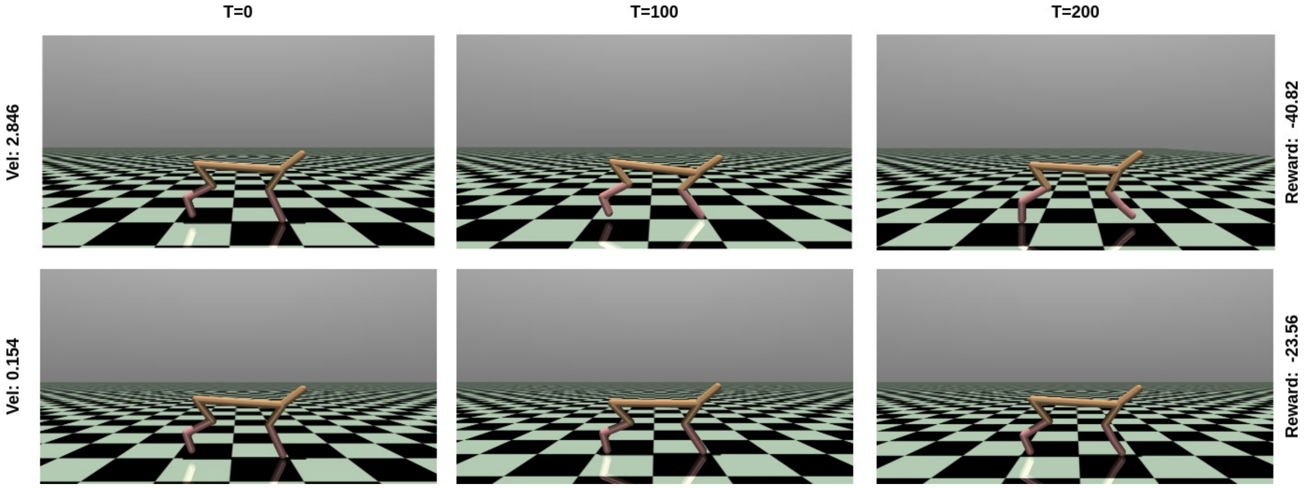

Fig. 3: Qualitative results for Half-Cheetah showcasing policies generated by AMT-Hyp at two extreme, previously unseen velocities, along with their corresponding rewards.

V. RESULTS AND ANALYSES

A. *Qualitative Analysis*

We qualitatively verify AMT-Hyp performance on the MetaWorld and MuJoCo continous control task (Half-Cheetah) as shown in the Fig.1 and Fig.3. Policy generated by AMT-Hyp when deployed exhibits correct behaviour qualitatively.

B. *Quantitative Analysis*

We quantitatively evaluate our framework on the MetaWorld and Cheetah-vel datasets to demonstrate its effectiveness in both Multi-task reinforcement learning (MTRL) and zero-shot generalization in RL. 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. In Table I, it is evident that our AMT-Hyp is better than MAA by 34.3% directly and AMT-Hyp without $HypFormer$ is better by 15% on MetaWorld Seen test tasks. Table II 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 [12], Decision Transformer DT [38]. Table III shows the success rate on completely unseen tasks of MetaWorld.

We evaluate zero-shot generalization capability on the Cheetah-vel dataset by averaging the returns across the test velocity seeds-(2, 7, 15, 23, 26) following [30], [35]. Our proposed model, AMT-Hyp, without $HypFormer$, achieves the best average return of -33.44 when trained using only 10 random velocities from the training set, outperforming state-of-the-art methods such as MACAW [30] and Prompt-DT [35], which utilize 35 seeds for training. When the number of training seeds is scaled from 10 to 35, the average return further improves to -29.6, emphasizing the robust zero-shot generalization capability of our framework.

TABLE I: AMT-Hyp 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	AMT-Hyp (w/o HypFormer)	AMT-Hyp (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 II: AMT-Hyp 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	*82.1*	58.5
DT	80.3	60.4
MAA	36	16.25
AMT-Hype w/o HypFormer	51	12.13
AMT-Hype	70.3	19.8
AMT-Hype w/o HypFormer (top 10)	**100**	54.38
AMT-Hype w HypFormer (top 10)	**100**	*80.63*
AMT-Hype w/o HypFormer (top 5)	**100**	75.1
AMT-Hype w HypFormer (top 5)	**100**	**87.5**

C. *Ablation Studies*

We present design decisions of AMT-Hyp framework in the context of MetaWorld tasks. We demonstrate the impact

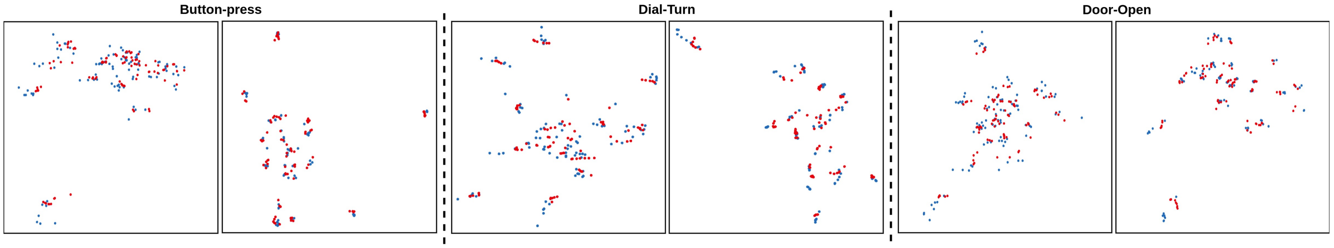

Fig. 4: Visualization of the predicted latent policy parameters for three MetaWorld tasks. In each task, the image on the left represents the output of the $HypLatent$ generator, while the image on the right shows the output of $HypFormer$. The red color points are the predicted latent policy parameters while the blue color points indicate the ground-truth latent policy parameters.

TABLE III: 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**	**AMT-Hyp (w/o HypFormer)**	**AMT-Hyp (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**

TABLE IV: Comparing AMT-Hyp to the offline Meta-RL works on Cheetah-Vel task.

Methods	**Avg. Return**
MACAW (Iter. 0)	-121.6
MACAW (Iter. 20K)	-60.5
Prompt-DT	-34.43
AMT-Hyp (w/o HypFormer) (10 seeds)	*-33.44*
AMT-Hyp (w/o HypFormer) (35 seeds)	**-29.6**

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 5, where the left image with the blue bar graph highlights this setup. As number of tokens increases, the performance on MTRL tasks decreases asymptotically. In contrast, for unseen tasks, the model achieves optimal performance with 32 tokens. Increasing the number of tokens from 8 to 32 results in a corresponding improvement in performance, but beyond 32, the performance begins to degrade as number of tokens continues to increase.

b) PCA Analysis: We present a PCA analysis on the latent policy parameters generated by $HypLatent$ and $HypFormer$, as illustrated in Figure 4 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 $HypFormer$.

Table IV demonstrates the effectiveness of $HypLatent$ in the Cheetah-vel setup. In summary, $HypLatent$, together with $HypFormer$, constitutes a key component of the AMT-Hyp design framework.

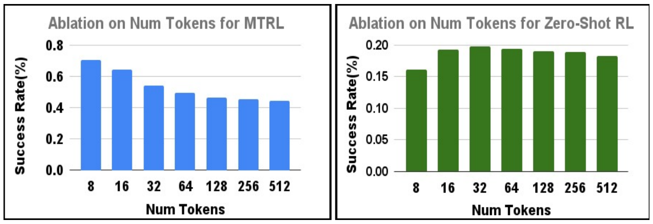

Fig. 5: Performance of AMT-Hyp on MetaWorld MTRL and Zero-Shot RL tasks on varying token size from 8 to 512. Better viewed at 2x zoom.

VI. CONCLUSION

In this paper, we propose a framework for training hypernetworks in the context of Multi-Task Reinforcement Learning (MTRL). By training a prior over the various task policies in an adversarial fashion, we encourage diversity of the generated latent policy parameters. We then use a single layer transformer architecture to guide the prior towards expert policy parameters. Our framework outperforms related hypernetwork-based baselines in MTRL and achieves performance comparable to state-of-the-art MTRL methods. Additionally, our experiments on MuJoCo control tasks demonstrate that the framework exhibits strong zero-shot generalization to unseen tasks within the same task distribution.

We believe that our framework is an important step towards tackling MTRL while retaining zero shot generalizability to in-distribution tasks.

VII. FUTURE WORK

We currently rely on expert policies during training, making it crucial to explore performance when training with suboptimal policies. Another promising future direction is to enhance the scalability of the framework to handle a larger number of tasks.

REFERENCES

[1] 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. 1

[2] 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. 1

[3] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?," in *International conference on machine learning*, pp. 9120–9132, PMLR, 2020. 1

[4] 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. 1

[5] 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. 1

[6] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multiobjective optimization," *Comptes Rendus Mathematique*, vol. 350, no. 5-6, pp. 313–318, 2012. 1

[7] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *Advances in neural information processing systems*, vol. 31, 2018. 1

[8] 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. 1

[9] 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. 1

[10] L. Liu, Y. Li, Z. Kuang, J. Xue, Y. Chen, W. Yang, Q. Liao, and W. Zhang, "Towards impartial multi-task learning," iclr, 2021. 1

[11] A. Navon, A. Shamsian, I. Achituve, H. Maron, K. Kawaguchi, G. Chechik, and E. Fetaya, "Multi-task learning as a bargaining game," *arXiv preprint arXiv:2202.01017*, 2022. 1

[12] 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. 1, 2, 5

[13] A. Hendawy, J. Peters, and C. D'Eramo, "Multi-task reinforcement learning with mixture of orthogonal experts," *arXiv preprint arXiv:2311.11385*, 2023. 1

[14] 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. 1

[15] D. Ha, A. Dai, and Q. V. Le, "Hypernetworks," *arXiv preprint arXiv:1609.09106*, 2016. 1

[16] 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. 1

[17] J. Von Oswald, C. Henning, B. F. Grewe, and J. Sacramento, "Continual learning with hypernetworks," *arXiv preprint arXiv:1906.00695*, 2019. 1

[18] 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. 1

[19] Y. Alaluf, O. Tov, R. Mokady, R. Gal, and A. Bermano, "Hyperstyle: Stylegan inversion with hypernetworks for real image editing," in *Proceedings of the IEEE/CVF conference on computer Vision and pattern recognition*, pp. 18511–18521, 2022. 1

[20] A. Zhmoginov, M. Sandler, and M. Vladymyrov, "Hypertransformer: Model generation for supervised and semi-supervised few-shot learning," in *International Conference on Machine Learning*, pp. 27075–27098, PMLR, 2022. 1

[21] 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

[22] 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. 1, 3

[23] 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. 1

[24] 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. 1

[25] A. Hallak, D. Di Castro, and S. Mannor, "Contextual markov decision processes," *arXiv preprint arXiv:1502.02259*, 2015. 2

[26] 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. 2, 4

[27] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012. 2

[28] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020. 2

[29] 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. 2

[30] 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, 4, 5

[31] 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

[32] 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. 2, 4

[33] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. 2

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

[35] 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. 4, 5

[36] 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. 4

[37] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017. 4

[38] 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. 5