Representations in Zero-Shot Meta-Reinforcement Learning

Jacob Beck

Linacre College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2025

Abstract

A long-standing goal in the field of Artificial Intelligence (AI) is to create autonomous agents capable of making a sequence of decisions to interact with the world. When interacting with its environment, an agent will encounter new situations that it has not previously encountered and for which we cannot provide an explicit representation of the environment. In this case, the agent must learn about its environment through interaction allowing trial and error – a problem well modeled by the field of Reinforcement Learning (RL). A long-standing goal in reinforcement learning is to create an agent capable of solving a diverse set of learning problems (i.e., broad generalization), and also learn quickly from few samples (i.e., rapid adaptation). While RL has made remarkable progress in generalization across diverse applications, from Go to nuclear fusion control, it remains sample-inefficient and slow to adapt.

In this thesis, we explore meta-reinforcement learning (meta-RL) as a solution to these challenges. In meta-RL, an agent is directly trained to rapidly adapt over a distribution of tasks. Still, generalization over a broad task distribution in meta-RL remains a significant hurdle. To address this, in Chapter 4, we propose the use of hypernetworks, i.e., neural networks that generate the weights and biases for other neural networks. In order to train hypernetworks stably, we introduce a novel initialization scheme that enables hypernetworks to improve generalization performance. We explore the implications of hypernetworks further in Chapter 5. We also show that, surprisingly, when used with our hypernetworks, simple end-to-end objectives can outperform more complex ones. We then explore the role of sequence models in meta-RL in Chapter 6. We specifically investigate sequence models that are permutation-invariant with respect to historical transitions. This inductive bias is theoretically sufficient for representing the optimal policy and can ameliorate gradient decay. However, we find that introducing controlled permutation variance improves architectural robustness and allows for the representation of sub-optimal policies, which can serve as stepping stones to the optimal solution. Finally, we consider how these methods might differ in practice, by considering an application of meta-supervised learning to protein fitness prediction in Chapter 7.

In summary, this thesis provides representations that critically improve the capabilities of meta-RL agents. We show that such agents can be trained with hypernetworks, but that doing so requires stable initialization. We show that end-to-end objectives provide effective supervision, but only when using hypernetworks. And, we show that permutation invariance is useful, but only when carefully combined with permutation variance as well. By integrating these insights, we offer a path forward for RL agents capable of both rapid adaptation and broad generalization, advancing a long-standing goal in the field of AI.

Acknowledgements

To my advisor, Shimon Whiteson, thank you for your guidance on my research, especially when encouraging me to pivot topics, and for your editorial insight, especially when framing the stories for my papers. You provided plenty of academic space to be independent, but also guidance when needed. A special thanks for editing our many drafts of a monstrously large survey paper, for which a single read was no small feat of stamina and labor.

To my collaborators, Risto Vuorio, Matthew Jackson, Luisa Zintgraf, Tarun Gupta, Zheng Xiong, Evan Liu, Mingfei Sun, and Chelsea Finn, thank you for making my research a reality. To my lab members, Kristian Hartikainen, Clemence Grislain, Nasma Dasser, Benjamin Ellis, Alex Goldie, Alex Zakharov, Bei Peng, Wendelin Boehmer, Mattie Fellows, Jelena Luketina, Matt Smith, Vitaly Kurin, Anuj Mahajan, Shangtong Zhang, and Maximilian Igl, thank you for your support and never-ending company at Opera Cafe. Tarun Gupta, thank you for your endless support and friendship and willingness to drive to Linacre at any hour. Risto Vuorio, thank you for being an outstanding person and researcher, with equally good advice on research and motorcycles. Kristian Hartikainen, thank you for all the road trips and vacations from work. Matthew Jackson, thank you for helping me unblock my very first project. Wendelin Boehmer, thank you for providing guidance and company during the pandemic. Luisa Zintgraf, thank you for your kind advice and for laying the foundations that I built off of in this thesis.

An immense thank you to my friends and family – I couldn't ask for any better. To my parents, thank you for always providing love and support (and funding). You made my PhD possible, made me believe anything was achievable, and always aimed to make my life easier. To my brother, Tyler, thank you for always providing support as well, especially when I didn't agree with the advice from our parents. To Uncle Kenny, thank you for your advice and laughter over FaceTime. To Rachel Hecht, thank you for your endless love and support while writing up and planning my life and career from here. To Margaret Norton, Lael and Martha Edwards-Costa, Dani Advani, Eva Orszaghova, Christoffer Alexandersen, Christoforos Kassianides, Saidu Kamara, Varnika Agarwal, Andrea Moncada, Anakaren Cervantes, Malavika Gode, Francesco Fiorani, Zofia Varyova, and Stefano Sferrazza, thank you for your love and support. Thank you to everyone who helped make Oxford a home.

Thank you to the people who started me on my academic journey. To Katja Hofmann, thank you for believing in me at Microsoft Research, and to Matt Cooper and Zoe Papakipos, thank you for your collaboration in Brown's academics and self-driving car lab. Finally, to Michael Littman, you will always have my deep gratitude for your time and encouragement that led me to research and machine learning. Your legendary teaching and attempt at using my Onewheel have left an indelible mark on my career. Thank you.

Contents

1 Introduction

Contents

A long-standing goal in the field of Artificial Intelligence (AI) is the development of agents that can interact with the world autonomously. When doing so, agents will encounter novel problems that necessitate learning from interaction to devise new solutions. This problem is often formalized by the field of Reinforcement Learning (RL), in which agents act to maximize future expected discounted reward [Sutton et al., 1999]. To function autonomously, these agents must be able to both generalize broadly over a diverse set of RL problems, and adapt rapidly to new RL problems, or *tasks*, by learning.

While significant progress has enabled RL to tackle a broad range of problems, such as mastering the game of Go [Silver et al., 2016] and solving all 57 Atari games to

super-human performance [Badia et al., 2020], these algorithms remain remarkably sample-inefficient. For example, solving all Atari games required 90 billion frames, which amounts to 48 years playing at a rate of 60 frames per second. While AI systems may be able to play games rendered much faster and in parallel, the slow speed of learning still becomes a major impediment to solving these problems in the physical world, which cannot be rendered faster. However, meta-RL offers a solution: One way to obtain more efficient RL algorithms is to *learn* to make more efficient RL algorithms.

1.1 Learning to learn quickly

Meta-reinforcement learning (meta-RL) considers a family of machine learning (ML) methods that *learn to reinforcement learn*. That is, meta-RL methods use sample-inefficient ML to learn sample-efficient RL algorithms, or components thereof. As such, meta-RL is a special case of meta-learning [Vanschoren, 2018, Hospedales et al., 2020, Huisman et al., 2021], with the property that the learned algorithm is an RL algorithm. Meta-RL has been investigated as a machine learning problem for a significant period of time [Schmidhuber, 1987, Schmidhuber et al., 1997, Thrun and Pratt, 1998, Schmidhuber, 2007].

Meta-RL has the potential to overcome some limitations of existing human-designed RL algorithms. There has been significant progress in deep RL over the last several years, with success stories such as mastering the game of Go [Silver et al., 2016], stratospheric balloon navigation [Bellemare et al., 2020], and robot locomotion in challenging terrain [Miki et al., 2022]. Still, RL remains highly sample inefficient, which limits its real-world applications. Meta-RL can produce (components of) RL algorithms that are much more sample efficient than existing RL methods, or even provide solutions to previously intractable problems.

1.2 Example application

Consider, as a conceptual example, the task of automated cooking with a robot chef. When such a robot is deployed in an individual's kitchen, it must learn a kitchen-specific policy, since each kitchen has a different layout and appliances. This challenge is compounded by the fact that not all items needed for cooking are in plain sight; pots might be tucked away in cabinets, spices could be stored on high shelves, and utensils might be hidden in drawers. Therefore, the robot needs not only to understand the general layout but also remember where specific items are once discovered. Training the robot directly in a new kitchen from scratch is too time consuming and potentially dangerous due to random behavior early in training. One alternative is to pre-train the robot in a *single* training kitchen and then fine-tune it in the new kitchen. However, this approach does not take into account the subsequent fine-tuning procedure. In contrast, meta-RL would train the robot on a *distribution* of training kitchens such that it can adapt to any new kitchen in that distribution. This may entail learning some parameters to enable better fine-tuning, or learning the entire RL algorithm that will be deployed in the new kitchen. A robot trained this way can both make better use of the data collected and also collect better data, e.g., by focusing on the unusual or challenging features of the new kitchen. This meta-learning procedure requires more samples than the simple fine-tuning approach, but it only needs to occur once, and the resulting adaptation procedure can be significantly more sample efficient when deployed in the new test kitchen.

This example illustrates how, in general, meta-RL may be particularly useful when the need for efficient adaptation is frequent, so the cost of meta-training is relatively small. This includes, but is not limited to, safety-critical RL domains, where efficient data collection is necessary and exploration of novel behaviors is prohibitively costly or dangerous. In many cases, a large investment of sample-inefficient learning upfront

(either with oversight, in a laboratory, or in simulation) is worthwhile to enable subsequent improved adaptation behavior.

1.3 Challenges

The promise of improved sample efficiency comes with several costs. Notably, meta-learning requires significantly more data than standard learning, as it trains an entire learning algorithm (often across multiple tasks). While meta-RL can enable sample-efficient learning at deployment, it does so at the expense of increased sample complexity during meta-training. For this reason, meta-RL is only applicable when upfront data collection is relatively cheap, or adaptation during deployment is prohibitively expensive. For example, this trade-off is justifiable when a simulator is available for meta-training or when frequent and efficient adaptation is required after meta-training. While this limitation is restrictive, it is not significantly more restrictive than the standard use case for reinforcement learning. Importantly, the methods proposed in this thesis maintain or improve the efficiency of meta-training, while also improving the speed of adaptation afterward.

Additionally, meta-learning fits a learning algorithm to meta-training data, which may reduce its ability to generalize to other data. A learning algorithm that does not generalize to a broad array of learning problems would defeat our goal of producing autonomous agents. Fortunately, this problem decreases as we increase the amount of computation and data used during meta-training. Thus, by increasing the scale of meta-training, we can produce agents that generalize broadly and adapt rapidly. While generalizing over a broad distribution of problems for meta-training, comes with its own set of challenges, this thesis makes significant advances in enabling such generalization over a diverse meta-training distribution.

1.4 Representations in Meta-Reinforcement Learning

In this thesis we tackle the problem of meta-RL, with a specific focus on representations that enable generalization when learning to learn rapidly. We consider the setting requiring the fastest adaptation, where performance is measured from the very first attempt at a problem, i.e., *zero-shot*. In this setting, an agent is directly trained to rapidly adapt over a distribution of tasks. Still, generalization over a broad task distribution in meta-RL remains a significant hurdle. This thesis tackles this problem, and others, using hypernetworks [Ha et al., 2017] in conjunction with other representations, to improve zero-shot meta-RL algorithms.

We begin by noting the difficulty of training a representation to generalize across the task distribution. On one hand, learning a separate policy for each task is feasible but does not solve the generalization problem. On the other hand, training a single policy across many tasks can create significant representation challenges. Despite this, most existing meta-RL algorithms train a single policy that differs between tasks only in terms of an input vector representing the task [Humplik et al., 2019, Duan et al., 2016, Wang et al., 2016, Zintgraf et al., 2021b]. In contrast, hypernetworks are a promising path forward since they replicate the ability to produce separate policies while also allowing for generalization across tasks. However, evidence from supervised learning suggests hypernetwork performance is highly sensitive to the initialization [Chang et al., 2020]. In Chapter 4, we 1) show that hypernetwork initialization is also a critical factor in meta-RL, and that naive initializations yield poor performance; 2) propose a novel hypernetwork initialization scheme that matches or exceeds the performance of a state-of-the-art approach proposed for supervised settings, as well as being simpler and more general; and 3) use this method to show that hypernetworks can improve performance in meta-RL by evaluating on multiple simulated robotics

benchmarks. In this work, we primarily aim to improve the performance of meta-RL methods that use an objective to explicitly infer the task.

In contrast to such objectives, meta-RL may train the sequence model, not to infer the task, but rather train the sequence model end-to-end on the return achieved by the meta-RL agent [Wang et al., 2016, Duan et al., 2016]. Recent work suggests that end-to-end learning in conjunction with an off-the-shelf sequential model, such as a recurrent network, is a surprisingly strong baseline [Ni et al., 2022]. However, such claims have been controversial due to limited supporting evidence, particularly in the face of prior work establishing precisely the opposite. In Chapter 5, we conduct an empirical investigation. While we likewise find that a recurrent network can achieve strong performance, we demonstrate that the use of hypernetworks is crucial to maximizing their potential. Surprisingly, when combined with hypernetworks, the recurrent baselines that are far simpler than existing specialized methods actually achieve the strongest performance of all methods evaluated.

In addition to specialized objectives, specialized sequence models have been proposed to take advantage of the structure present in the meta-RL problem. Specifically, many methods use sequence models that are permutation-invariant with respect to historical transitions [Rakelly et al., 2019, Korshunova et al., 2020, Raileanu et al., 2020, Imagawa et al., 2022]. This inductive bias is theoretically sufficient for representing the optimal policy and can ameliorate gradient decay. However, in Chapter 6, we find that introducing controlled permutation variance actually improves architectural robustness and allows for the representation of sub-optimal policies, which can serve as stepping stones to the optimal solution. Thus, we propose a novel sequence model to carefully combine both permutation invariant and permutation variant components. By combining this insight with hypernetworks, along with our improvements to the initialization and supervision, we arrive at a representation that significantly advances

the capabilities of generalization and adaptation in zero-shot meta-RL.

While this representation advances the capabilities of meta-RL agents, we note that methodologies may need to vary significantly in practice. For example, real-world applications might leverage large pre-trained models for transfer, incorporate fine-tuning at test time, or use meta-supervised learning rather than meta-reinforcement learning. We explore such practicalities by investigating meta-learning applied to the domain of protein design in Chapter 7. While practical limitations imposed by the application necessitate the use of meta-supervised learning rather than meta-reinforcement learning, protein engineering remains a preferable choice due to the availability of an accepted public benchmark, the ability to train without the challenges of bridging the simulation-to-reality gap, the lack of existing meta-learning solutions, and the far-reaching implications for the physical sciences of any improvement achieved. Furthermore, the focus on protein design was reinforced by the constraints of a research internship, which provided the necessary collaboration and computation. Combined with our research on meta-RL agents, this thesis offers both comprehensive improvements to the capabilities of meta-RL agents and practical insights into considerations that arise in real-world applications.

1.5 Thesis Structure and Contributions

This thesis is structured as follows:

- Chapter 1, i.e., this chapter, explains the motivation and goals behind meta-RL. We present several challenges entailed by meta-RL and discuss how this thesis makes progress on the most critical challenge facing the field. Finally, this chapter outlines the research in the following portion of this thesis. The content of this chapter is based on Beck et al. [2025b], published at Foundations and Trends in Machine Learning.

- Chapter 2 introduces the problem setting and formalism for RL and meta-RL. In this chapter, we give the technical background required to understand RL and meta-RL. We describe the Markov Decision Process (MDP), and then define meta-RL in terms of a distribution of MDPs. We define some terminology including the inner-loop, outer-loop, meta-training, meta-testing, and meta-parameters. We walk through some example algorithms and categorize meta-RL based on the number of episodes available for adaptation. Finally, we draw connections to related fields, devoting the majority of this chapter to exploring connections with meta-supervised learning. The content of this chapter is based on Beck et al. [2025b], published at Foundations and Trends in Machine Learning.

- Chapter 3 categorizes the two main types of algorithms used to solve the zero-shot meta-RL problem and presents related literature. We break up existing methods into the categories of black-box and task-inference meta-RL. We describe each in terms of the parameters they adapt, their inner-loop representations, their respective outer-loop algorithms, and their associated trade-offs. We conclude with a discussion on works towards exploration in meta-RL. The content of this chapter is based on Beck et al. [2025b], published at Foundations and Trends in Machine Learning.

- Chapter 4 presents the first methodological contribution, by investigating the use of hypernetworks, in conjunction with novel initialization methods, for broad task distributions. We find naive initializations for Hypernetworks yield poor performance, and we propose a novel initialization scheme that matches or exceeds the performance of a state-of-the-art approach, as well as being simpler and more general. The content of this chapter is based on Beck et al. [2022], published at The 6th Conference on Robot Learning.

- Chapter 5 presents findings on the use of different objectives in conjunction with

the hypernetwork representation. While many specialized methods have been proposed for meta-RL, we conduct an empirical investigation to compare the end-to-end objectives with the more specialized methods. Surprisingly, even after we further improve the specialized methods, we find the end-to-end baseline to be both more performant and easier to implement. Moreover, we show the use of hypernetworks to be critical to enabling the superior performance of end-to-end methods. The content of this chapter is based on Beck et al. [2023b], published at The 37th Conference on Neural Information Processing Systems.

- Chapter 6 investigates the sequence model that is used to summarize learning experience before being passed to the hypernetwork. We specifically investigate sequence models that are permutation-invariant with respect to historical transitions. While permutation invariance is sufficient for representing the optimal policy in principle, we find surprising benefits from introducing controlled permutation variance. Not only does integration of permutation variance improve architectural robustness, but we find that it also allows for the representation of sub-optimal policies, which can serve as stepping stones on the path to learning the optimal policy. The content of this chapter is based on Beck et al. [2024a], published at The First Reinforcement Learning Conference.

- Chapter 7 explores meta-supervised learning applied to the task of protein fitness prediction. In this setting, existing methods use large protein language models (PLMs) that are typically pre-trained on general protein sequence modeling tasks and then fine-tuned for protein fitness prediction. When no task data is available, the models make strong assumptions about the correlation between the protein sequence likelihood and fitness scores. In contrast, we propose meta-learning over a distribution of standard fitness prediction tasks, and demonstrate positive transfer to unseen fitness prediction tasks. The content of this chapter is based

on Beck et al. [2025a], published at The Thirteenth International Conference on Learning Representations.

- Finally, in Chapter 8, we conclude with a discussion of the advances in representation proposed in this work that critically improve the capabilities of meta-RL agents, and we engage in a reflection on open problems in the field.

We hope that by integrating the insights provided, we offer a path forward for building RL agents capable of both rapid adaptation and broad generalization, advancing a long-standing goal in the field of Artificial Intelligence.

1.6 Statement of Authorship

Chapter 1 is based on content from **A Tutorial on Meta-Reinforcement Learning** [Beck et al., 2023a, 2025b].

Content from Beck et al. [2023a, 2025b] is additionally included in the thesis: Vuorio, Risto. "Advances in Meta-Reinforcement Learning and Imitation Learning." PhD diss., University of Oxford, 2024.

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the co-lead author on this manuscript. I was responsible for the writing, structure, and surveying, along with the other co-lead. I had a particular responsibility for content related to few-shot and zero-shot meta-RL, whereas the other co-lead focused on many-shot methods, not germane to this thesis. I additionally reviewed, edited, and re-wrote all content as needed.

Citation:

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. "A Tutorial on Meta-Reinforcement Learning." *Foundations and Trends in Machine Learning*, 2025.

2

Background

Contents

Meta-RL can be broadly described as learning part or all of an RL algorithm. In this chapter we define and formalize meta-RL. To do so, we start by defining RL. We define some terminology including the inner-loop, outer-loop, meta-training, meta-testing, and meta-parameters. We walk through some example algorithms and categorize meta-RL based on the number of episodes available for adaptation. Finally, we draw connections to related fields.

2.1 Reinforcement learning

An RL algorithm learns a policy to take actions in a Markov decision process (MDP), also referred to as the agent's *environment*. An MDP is defined by a tuple $\mathcal{M} =$

$\langle \mathcal{S}, \mathcal{A}, P, P_0, R, \gamma, T \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $P(s_{t+1}|s_t, a_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ the probability of transitioning from state s_t to state s_{t+1} after taking action a_t, $P_0(s_0) : \mathcal{S} \to \mathbb{R}_+$ is a distribution over initial states, $R(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\gamma \in [0, 1]$ is a discount factor, and T is the horizon. We give the definitions assuming a finite horizon for simplicity, though many of the algorithms we consider also work in the infinite horizon setting. A policy is a function $\pi(a|s) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ that maps states to action probabilities. The interaction of the policy with the MDP takes place in *episodes*, which start from initial states sampled from P_0 followed by T transitions between states where the actions are sampled from the policy π and states from the dynamics P. After the T transitions, a new episode begins starting from a freshly sampled initial state. The reward for each transition is defined by the reward function $r_t = R(s_t, a_t)$. This defines a distribution over episodes

$$P(\tau) = P_0(s_0) \prod_{t=0}^{T} \pi(a_t|s_t) P(s_{t+1}|s_t, a_t). \tag{2.1}$$

Sometimes we refer to the data $\tau = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T}$ collected during an episode as a *trajectory*.

The objective of RL is to learn a policy that maximizes the expected discounted return within an episode,

$$J(\pi) = \mathbb{E}_{\tau \sim P(\tau)} \left[\sum_{t=0}^{T} \gamma^t r_t \right], \tag{2.2}$$

where r_t are the rewards along the trajectory τ. In the process of optimizing this objective, multiple episodes are gathered. If H episodes have been gathered, then $\mathcal{D} = \{\tau^h\}_{h=0}^{H-1}$ is all of the data collected in the MDP. An RL algorithm is a function that maps the data to a policy. This includes choosing the policies used for data collection during training. These intermediate policies are not necessarily greedy with respect to the RL objective but may take exploratory actions instead. In this survey, we mostly consider parameterized policies with parameters $\phi \in \Phi$. Therefore, we define

an RL algorithm as the function $\phi = f(\mathcal{D})$, which maps data to policy parameters. In practice, the data may include a variable number of episodes of variable length.

2.2 Meta-RL definition

RL algorithms are traditionally designed, engineered, and tested by humans. The idea of meta-RL is instead to learn (parts of) an algorithm f using machine learning. Where RL learns a policy, meta-RL learns the RL algorithm f that outputs the policy. This does not remove all of the human effort from the process, but shifts it from directly designing and implementing the RL algorithms into developing the training environments and parameterizations required for learning them in a data-driven way.

Due to this bi-level structure, the algorithm for learning f is often referred to as the *outer-loop*, while the learned f is called the *inner-loop*. Since the inner-loop and outer-loop both perform learning, we refer to the inner-loop as performing *adaptation* and the outer-loop as performing *meta-training*, for the sake of clarity. The learned inner-loop, that is, the function f, is assessed during *meta-testing*. We want to meta-learn an RL algorithm, or an inner-loop, that can adapt quickly to a new MDP. This meta-training requires access to a set of training MDPs. These MDPs, also known as *tasks*, come from a distribution denoted $p(\mathcal{M})$. In principle, the task distribution can be supported by any set of tasks. However, in practice, it is common for $\mathcal{S}$ and $\mathcal{A}$ to be shared between all of the tasks and the tasks to only differ in the reward $R(s, a)$ function, the dynamics $P(s'|s, a)$, and initial state distributions $P_0(s_0)$. Meta-training proceeds by sampling a task from the task distribution, running the inner-loop on it, and optimizing the algorithm to improve the policies it produces. The interaction of the inner-loop with the task, during which the adaptation happens, is called a *meta-episode*, *lifetime*, or *trial* and is illustrated in Figure 2.1. A trial can consist of multiple episodes. After an episode ends, a new one begins with the initial state of the new episode sampled from the initial state distribution $P_0(s_0)$.

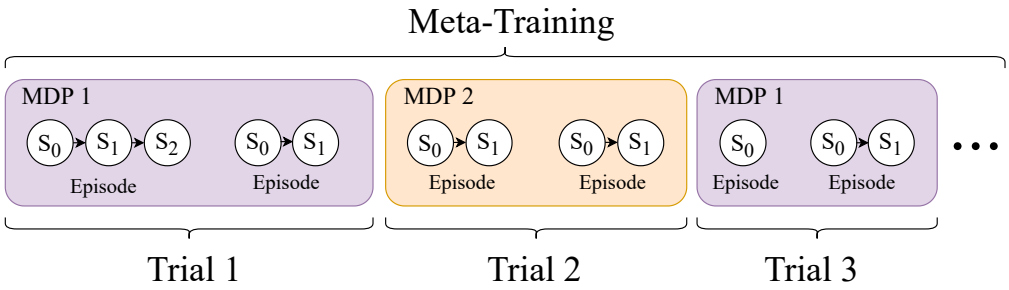


Figure 2.1: Meta-training consists of trials (or meta-episodes), each broken up into multiple episodes from a single task (MDP). In this example, each trial consists of two episodes ($H = 2$).

Parameterization We generally parameterize the inner-loop f_θ with *meta-parameters* θ, and learn these parameters in the outer loop to maximize a meta-RL objective. Hence, f_θ outputs the parameters of π_ϕ: $\phi = f_\theta(\mathcal{D})$. Note that while $\mathcal{D}$ contains multiple trajectories, these can be flattened into in a single *meta-trajectory* that contains multiple episodes of interaction, and so it is common to use the same symbol as in RL, τ, but refer to it as a *meta-trajectory*. Thus, we can write $\phi = f_\theta(\tau)$, where τ is implied to cross the boundaries between episodes. We refer to the policy π_ϕ as the *base policy*. Some meta-RL algorithms use f_θ to map τ to ϕ at every MDP step, while others do so less frequently.

Meta-RL objective In standard RL, a Markov policy suffices to maximize the RL objective given by Equation 2.2. In contrast, in meta-RL, the inner-loop is an entire RL algorithm that outputs parameters of policies as a function of the meta-trajectory. The performance of a meta-RL algorithm is measured in terms of the returns achieved by the policies π_ϕ produced by the inner-loop during a meta-episode on tasks $\mathcal{M}$ drawn from the task distribution. Formally, the meta-RL objective is:

$$\mathcal{J}(\theta) = \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}\left[\mathbb{E}_\tau\left[G(\tau)\middle|\pi(\cdot;\phi = f_\theta(\tau)), \mathcal{M}\right]\right], \tag{2.3}$$

where $G(\tau) = \sum_{t=0}^{T}\gamma^t r_t$ is the discounted return in the MDP $\mathcal{M}$ along the meta-trajectory τ.

Note that the return, G, sums reward from the very first time step. Depending on the application, slightly different objectives are considered. For some applications, we can afford a free exploration or adaptation period, during which the performance of the policies produced by the inner-loop is not important as long as the final policy found by the inner-loop solves the task. The episodes during this phase can be used by the inner-loop for freely exploring the task. The number of such "free" episodes, in the meta-episode, is called the *shot*. For applications that require rapid adaptation, such as those considered by this thesis, a free exploration period is not possible and correspondingly the agent must maximize the expected return from the first timestep it interacts with the environment. Thus, this thesis considers the zero-shot setting.

The meta-RL objective defined by Equation 2.3 is evaluated in expectation over samples from the task distribution, $p(\mathcal{M})$. During meta-training, samples are taken from $p(\mathcal{M})$, generally without assuming access to the true distribution. This results in a generalization problem in meta-testing, as testing tasks may not match the training tasks. This problem is made worse when the meta-testing task distribution does not share support with the training task distribution. Such out-of-distribution cases are often studied in meta-RL.

2.3 Exploration and Bayesian RL

In Reinforcement Learning, there is an exploration-exploitation trade-off, where the agent must balance taking exploratory actions to learn about the new task while exploiting what it already knows to achieve high rewards. For example, the agent may need to explore to gather information during the first few episodes in order to know how to solve the task in later episodes. However, in doing so, the agent may be sacrificing short-term rewards, which presents a trade-off.

In general, when the number of episodes available to the inner loop is large, more

exploration is optimal, as sacrificing short-term rewards to learn a better policy for higher later returns pays dividends. In contrast, when the number of episodes is small, the agent must exploit more to obtain any reward it can, before time runs out. For the adaptation to be sample-efficient, the exploration must be efficient as well. Meta-RL can be seen as a way of learning to make this trade-off in a way that is optimal with respect to Equation 2.3, for a particular distribution of tasks. Solutions to this dilemma are reviewed in Section 3.3.

Another field of study concerned with optimal exploration is *Bayesian RL* [Duff and Barto, 2002, Ghadirzadeh et al., 2021]. The problem setting defining meta-RL, posed by Equation 2.3, can additionally be viewed as a special case of a partially observable Markov decision process (POMDP) [Duan et al., 2016, Humplik et al., 2019]. In particular, Bayesian RL solves this POMDP by explicitly maintaining a posterior over tasks and updating it using Bayesian inference. The Bayesian framework provides a convenient method for incorporating prior knowledge and explicitly maintaining uncertainty. Using this framework, a new MDP can be constructed, where the state includes the posterior over POMDP hidden states, or equivalently, the MDP state and a posterior over tasks. In this case, the resulting MDP is called a Bayes-adaptive Markov decision process (BAMDP). This construction allows learning a Markovian policy to solve the meta-RL problem and therefore explores optimally. However, since Bayesian RL methods must explicitly model and update a distribution over MDPs, they are tractable only in simple domains without strong approximations. For example, even scalable Bayesian RL methods may be limited to discrete-space MDPs [Guez et al., 2013]. Instead of engineering approximations for the Bayesian posterior, meta-RL methods may learn to model these components as needed. Moreover, most meta-RL methods only require sample access to the prior instead of the prior being explicitly known, so the meta-RL agent may learn to implicitly model the prior over tasks from samples. Still, many meta-RL methods explicitly attempt to infer

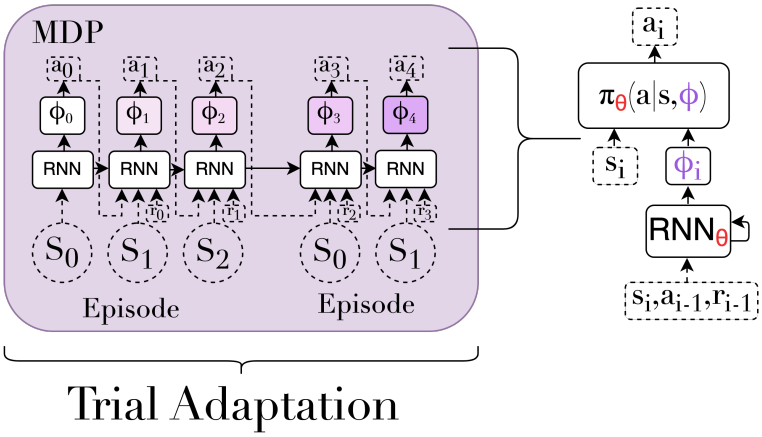


Figure 2.2: RL^2 in the problem setting (left) and conceptually (right). The inner-loop algorithm is implemented by an RNN parameterized by the meta-parameters θ. The RNN takes as input the states, actions, and rewards from the environment. The RNN hidden state ϕ_i defines the task parameters at each timestep, which are passed as input to the MLP policy. The hidden state is not reset during a trial and instead carries over across episode boundaries. The outer-loop is a standard RL algorithm.

a posterior distribution over tasks, inspired by Bayesian RL [Zintgraf et al., 2020], which are reviewed in Section 3.2.

2.4 Example algorithm

We now describe a canonical meta-RL algorithm that optimize the objective given by Equation 2.3: Fast RL via Slow RL (RL^2), which uses a history dependent policy [Duan et al., 2016, Wang et al., 2016]. Many meta-RL algorithms build on concepts and techniques similar to those used in RL^2, which makes it an excellent entry point to meta-RL.

RL^2, structures its inner-loop, f_θ, to approximate a general function of history. In this case, the inner-loop f_θ and policy π_ϕ, can be viewed together as a single object, forming a history-dependent policy. When considering the objects together as a history-dependent policy, we indicate this using the notation, $\pi_\theta(a|s,\phi)$, to emphasize that the inner-loop parameters are folded inside the policy, with ϕ representing a summary statistic of history. History-dependent policies can, in fact, solve a more

general class of problems, called a Partially Observable Markov Decision Process (POMDP), and the formal problem setting defining meta-RL can be viewed as a special case of a partially observable Markov decision process (POMDP) [Duan et al., 2016, Humplik et al., 2019]. While such history-dependent policies do not take into account all of the specific structure of the meta-RL problem beyond that of a more general POMDP, they are sufficient for solving the meta-RL problem.

RL^2 is dependent on *the entire history* of its interaction with the environment and is trained on tasks from the task distribution end-to-end with RL. This history includes all of the states, actions, and rewards the policy encounters during a meta-episode. This results in learning an adaptive policy that changes its behavior as it gathers more information about the environment. The history dependent policy is generally represented as a recurrent neural network (RNN). While in this example we focus on RNNs, any history dependent policy could be used.

Concretely, RL^2 treats the meta-episode as a single continuous sequence, during which the RNN hidden state is not reset, even if the trial spans multiple episodes in the underlying MDP. The meta-parameters θ are the parameters of the RNN and other neural networks used in processing the inputs and outputs of f_θ. The task parameters ϕ are the ephemeral hidden states of the RNN, which may change after every timestep. The operation of the algorithm is illustrated in Figure 2.2.

RL^2 directly approximates the optimal policy of the meta-RL objective, given by equation 2.3. This policy, known as Bayes-optimal, is the best policy for the task distribution. Bayes-optimal policies always choose actions that maximize the expected return under uncertainty about the MDP identity, and thus represent the fastest class of adaptation.

2.5 Related Fields

There are many fields closely related to meta-RL, which we briefly discuss in order to situate meta-RL as an area of study. One such field is few-shot learning Wang et al. [2020b]. Few-shot learning considers the aim of rapid adaptation, whereas meta-RL specifically requires a task-distribution for learning to perform rapid adaptation. AutoRL considers methods for adapting to new MDPs automatically, but generally targets the many-shot setting (not the few-shot setting considered here) and also includes non-learned algorithms for adapting to new MDPs Parker-Holder et al. [2022]. Generalization in RL is another field, which includes meta-RL methods, but also includes both non-learned algorithms and problems that do not require adaptation, i.e., problems where a single policy can perform optimally without specialization to the given task Kirk et al. [2023]. Finally, continual RL can be seen as an extension of the meta-RL problem setting, where the tasks cannot be sampled independently from $p(\mathcal{M})$, but are generated sequentially according to some stochastic process. Continual learning can therefore be viewed as a more difficult version of meta-RL, but meta-RL can also be used as a solution to continual RL, if we assume access to independent samples from a distribution over stochastic processes [Vuorio et al., 2018, Ritter et al., 2018a, Javed and White, 2019]. Of all the related machine learning fields, meta-supervised learning (meta-SL) and multi-task RL, are the most closely related.

Meta-Supervised Learning There are many points of similarity between meta-RL and meta-SL research. Like meta-RL, meta-SL considers a distribution of tasks. However, whereas the tasks in meta-RL are defined by MDPs, the tasks in meta-SL are defined by fixed datasets. Some algorithms have been proposed as methods for both meta-SL and meta-RL [Finn et al., 2017, Mishra et al., 2018]. Moreover, many algorithmic choices have analogues in both settings. For example, in the few-shot setting, the use of attention over prior states in meta-RL [Ritter et al., 2018b, Fortunato

et al., 2019, Team et al., 2023], has analogues in the kernel-based methods of supervised meta-SL [Santoro et al., 2016, Vinyals et al., 2016, Sung et al., 2018]. Additionally, in the many-shot setting, meta-learning optimizers and objective functions has been covered in both meta-RL [Houthooft et al., 2018, Kirsch et al., 2019, Oh et al., 2020, Bechtle et al., 2021, Lu et al., 2022, Lan et al., 2023] and meta-SL [Andrychowicz et al., 2016, Li and Malik, 2017, Bechtle et al., 2021]. Finally, imitation learning is considered a closely related pursuit to RL, since both consider sequential decision making problems.

In addition to similarities in the problem settings and methods, meta-RL and meta-SL have some key differences. In particular, meta-SL generally considers fixed datasets while meta-RL does not. In meta-RL, the adapted policy is responsible for collecting more data, which is fed back into the dataset for adaptation. This introduces a problem of data collection for adaptation to the given task. Moreover, since the meta-RL agent must adapt quickly, this requires that the agent also explores quickly. Fast exploration and the resulting challenges are unique to meta-RL.

Multi-Task RL Multi-task RL, like meta-RL, considers a distribution of MDPs, and can enable learning similar behaviors, such as meta-reasoning, when learning to plan [Budd et al., 2024]. However, whereas a meta-RL agent must identify the MDP that it encounters, in multi-task RL, the agent has access to a ground-truth representation of the task. While some solutions from multi-task RL, such as PopArt [Hessel et al., 2019], are immediately applicable to meta-RL [Grigsby et al., 2024], differences in evaluation also prevent some methods from being applicable. For example, multi-task RL is often evaluated over a finite set of discrete tasks, whereas meta-RL agents often encounter new tasks at test time [Yu et al., 2020, Teh et al., 2017]. This allows multi-task methods to train separate networks for each task [Teh et al., 2017], whereas such approaches are not immediately applicable in meta-RL.

Generally, multi-task RL can be seen as an easier version of meta-RL, in the sense that the MDP representation is known, so there is no need to learn it from data, nor to explore to get data for adaptation. In fact, an entire category of meta-RL methods tries to explicitly infer the MDP identity in order to reduce the meta-RL problem to the easier multi-task RL problem, which is discussed in Chapter 3. However, there can also be cases in which meta-RL is possible and multi-task RL is not. Since generalization to different tasks in multi-task RL occurs without conditioning on data at test time, it is always zero-shot generalization. Such generalization may not be possible, in practice, if the task representation is not sufficiently informative. For example, if the task is represented as a one-hot categorical vector, then it may be impossible to generalize to categories not seen during training. However, while multi-task RL would fail, meta-learning may still be viable in this case, if a more informative representation of the task can be inferred from data, or if additional learning at test time can compensate for sparsity in the training distribution.

2.6 Statement of Authorship

Chapter 2 is based on content from **A Tutorial on Meta-Reinforcement Learning** [Beck et al., 2023a, 2025b].

Content from Beck et al. [2023a, 2025b] is additionally included in the thesis: Vuorio, Risto. "Advances in Meta-Reinforcement Learning and Imitation Learning." PhD diss., University of Oxford, 2024.

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the co-lead author on this manuscript. I was responsible for the writing, structure, and surveying, along with the other co-lead. I had a particular responsibility for content related to few-shot and zero-shot meta-RL, whereas the other co-lead focused on many-shot methods, not germane to this thesis. I additionally reviewed, edited, and re-wrote all content as needed.

Citation:

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. "A Tutorial on Meta-Reinforcement Learning." *Foundations and Trends in Machine Learning*, 2025.

3

Literature Review

Contents

In this chapter, we discuss zero-shot adaptation, where the agent meta-learns across multiple tasks and, at meta-test time, must adapt to a new, but related, task as quickly as possible.

As a concrete example, recall the robot chef learning to cook in home kitchens. Training a new policy to cook in each user's home using reinforcement learning from scratch

would require many samples in each kitchen, which can be wasteful as general cooking knowledge (e.g., how to use a stove) transfers across kitchens. Wasting many data samples in the homes of customers who purchased a robot may be unacceptable, particularly if every action the robot takes risks damaging the kitchen. Meta-RL can automatically learn a procedure, from data, for adapting to the differences that arise in new kitchens (e.g., the location of the cutlery). During meta-training, the robot may train in many different kitchens in simulation or a setting with human oversight and safety precautions. Then, during meta-testing, the robot is sold to a customer and deployed in a new kitchen, where it must quickly learn to cook in it, making as few mistakes as possible. However, training such an agent with meta-RL involves unique challenges and design choices.

In particular, here we summarize the literature in terms of the parameterization of the inner loop. We categorize by the type of inner loop, since the majority of research considers different inner-loop parameterizations. We also consider exploration, since learning exploration is unique to meta-RL compared to meta-SL, as discussed in Chapter 2.

Meta-parameterization In this section, we first discuss two common categories of methods in the zero-shot setting. Methods in these categories are further classified in Table 3.1. Recall that meta-RL itself learns a learning algorithm f_θ. This places unique demands on f_θ and suggests particular representations for this function. We call this design choice the *meta-parameterization*, with the most common ones being the following:

- **Black box** methods impose little to no structure on f_θ. We review these in Section 3.1.
- **Task inference** methods structure f_θ to explicitly infer the unknown task. We review these in Section 3.2.

Black Box	
RL2-Like	Heess et al. [2015], Duan et al. [2016], Wang et al. [2016]
Adapt Policy Using Context Vector	Duan et al. [2016], Wang et al. [2016], Zintgraf et al. [2019], Humplik et al. [2019], Zintgraf et al. [2020], Fakoor et al. [2020], Liu et al. [2021]
Inner-Loop with Hebbian Learning	Miconi et al. [2018, 2019], Najarro and Risi [2020], Chalvidal et al. [2022], Rohani et al. [2022]
Inner-Loop with Attention	Oh et al. [2016], Ritter et al. [2018a], Mishra et al. [2018], Fortunato et al. [2019], Ritter et al. [2021], Melo [2022], Xu et al. [2022], Team et al. [2023], Elawady et al. [2024]
Task Inference	
Multi-task pre-training	Humplik et al. [2019], Kamienny et al. [2020], Raileanu et al. [2020], Peng et al. [2021], Liu et al. [2021]
Without privileged information	Guo et al. [2018], Rakelly et al. [2019], Zintgraf et al. [2020], Fu et al. [2021], Zhang et al. [2021], Luo et al. [2022]
Permutation invariance	Rakelly et al. [2019], Raileanu et al. [2020], Korshunova et al. [2020], Imagawa et al. [2022]
Adapt Policy Using Hypernetworks	Peng et al. [2021]

Table 3.1: Representative examples of zero-shot meta-RL research categorized by method. The majority of methods fall into one of two clusters, black box or task inference. These categories determine how the inner-loop is parameterized. Within each cluster, we further categorize the methods. Explanations of these methods can be found in Sections 3.1, and 3.2, respectively.

Note that other parameterizations are possible in other settings. For example, in the few-shot setting and many-shot setting, in which the agent is allowed "free" episodes just for exploration, *gradient-based methods* (or *parameterized policy-gradient methods*) build in the computation of a policy gradient into the inner-loop itself [Beck et al., 2023a]. These methods compute a gradient through this adaptation process, to learn a model initialization that can be adapted to new tasks with a few policy gradient updates [Finn et al., 2017, Li et al., 2017, Vuorio et al., 2019]. Such methods generalize better than black-box methods in principle (although not necessarily in practice [Xiong et al., 2021, Beck et al., 2025a]). However, the required computation of such meta-gradients is computationally expensive, the meta-gradients are subject to issues of bias and variance [Vuorio et al., 2021], and to get a sufficiently precise estimate of the policy gradient at deployment generally requires too much data to be competitive in the zero-shot setting [Zintgraf et al., 2020]. Also, note that black box methods and task-inference methods are sometimes referred to collectively as *context-based* methods [Rakelly et al., 2019].

Along with the parameterization of the inner-loop, related design choices include the representation of the base policy and the choice of the outer-loop algorithm. Each method must additionally specify which base policy parameters are adapted by the inner-loop, and which are meta-parameters. We call this design choice the adapted policy parameters. For each of the three types of meta-parameterizations in this section, we discuss the inner-loop, the outer-loop, and the adapted policy parameters.

Exploration While all these methods are distinct, they also share some challenges. One such challenge is that of *exploration*, the process of collecting data for adaptation. In zero-shot learning, exploration determines how an agent takes actions during its few shots. Subsequently, an agent must decide how to adapt the base policy using this collected data. For the adaptation to be sample efficient, the exploration must be efficient as well. Specifically, the exploration must target differences in the task distribution (e.g., different locations of cutlery). In Section 3.3, we discuss the process of exploration, along with ways to add structure to support it. While all zero-shot methods must learn to explore an unknown MDP, there is an especially tight relationship between exploration methods and task inference methods. In general, exploration may be used to enable better task inference, and conversely, task inference may be used to enable better exploration. One particular way in which task inference may be used to enable better exploration is by quantifying uncertainty about the task, and then choosing actions based on that quantification. This method may be used in order to learn optimal exploration.

3.1 Black Box Methods

Black box methods represent the inner-loop, f_θ, as a sequence model to process data for learning. Learning in the inner-loop occurs solely in the activations of this sequence model, a phenomenon sometimes called *in-context learning* [Brown et al., 2020, Moeini

et al., 2025]. Meta-learning f_θ is a way to explicitly elicit the phenomenon of in-context learning in a sequence model. In principle, black-box methods can learn any arbitrary learning procedure, since they represent f_θ with a neural network as a universal function approximator. Since f_θ represents an arbitrary function of history, the modified policy can represent an arbitrary history-dependent policy. As discussed in Chapter 2, such a policy is sufficient for learning an optimal policy for a POMDP, and thus for the meta-RL problem as well. Recall the RL2 algorithm [Duan et al., 2016] from Section 2.4. In this case, f_θ is represented by a recurrent neural network, whose outputs are an input vector to the base policy. While the use of a recurrent network to output a context vector is common, black box methods may also structure the base policy and inner-loop in other ways. In this section we survey black box methods. We begin by discussing architectures that are designed for different amounts of diversity in the task distribution. Each of these architectures has different adapted policy parameters. Then, we discuss distinct ways to structure the inner-loop and alternative outer-loop algorithms. Finally, we conclude with a discussion of the trade-offs associated with black box methods.

3.1.1 Adapted policy parameters

On one hand, some task distributions may require little adaptation for each task. For example, if a navigation task varies only by goal location, then it may be that not many policy parameters need to be adapted. Many black box methods have been applied to such task distributions [Wang et al., 2016, Humplik et al., 2019, Zintgraf et al., 2020]. In such a setting, it is sufficient to only adapt a vector, used as an input to the base policy, instead of all parameters of the base policy [Duan et al., 2016, Wang et al., 2016, Zintgraf et al., 2019, Humplik et al., 2019, Zintgraf et al., 2020, Fakoor et al., 2020, Liu et al., 2021]. This vector, ϕ, is called a *context vector*. The context is produced by a recurrent neural network, or any other network that

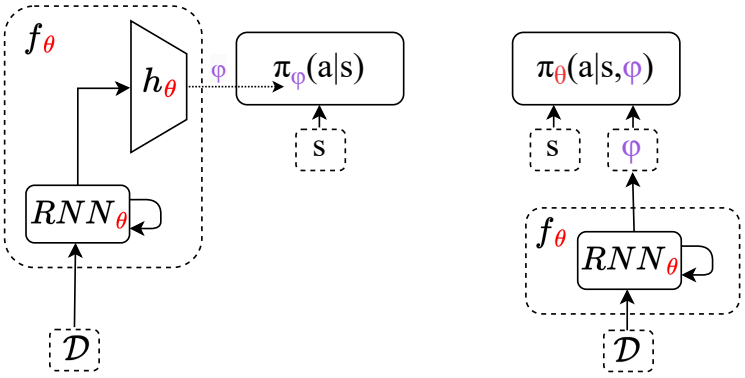


Figure 3.1: Illustration of black box meta-RL with hypernetwork (left) and with a context vector (right). When the inner-loop is a recurrent neural network, generating all the parameters of the policy, including its weights and biases, results in a hypernetwork. Alternatively, the RNN can output a context vector on which the policy is conditioned.

conditions on history, such as a transformer or memory-augmented network. Using a recurrent neural network, the inner-loop can be written $f_\theta(\mathcal{D}) = \text{RNN}_\theta(\mathcal{D})$. Recall that we can also flatten our data from multiple episodes to write this in terms of the meta-trajectory, $f_\theta(\tau) = \text{RNN}_\theta(\tau)$. In this case, the inner-loop (f_θ) and base policy ($\pi_\theta(a|s,\phi)$) together, can also be seen as a single object, forming a history-dependent policy. This is a common architecture for zero-shot adaptation methods [Duan et al., 2016, Wang et al., 2016, Humplik et al., 2019, Zintgraf et al., 2020, Fakoor et al., 2020, Liu et al., 2021].

On the other hand, some task distributions may require significant differences in behavior between the optimal policies. By conditioning a policy on a context vector, all of the weights and biases of π must generalize between all tasks. However, when significantly distinct policies are required for different tasks, forcing base policy parameters to be shared may impede adaptation, as we will see in Chapter 4. Alternatively, there may be no context vector, but the inner-loop may adapt the weights and biases of the base policy, directly. The weights and biases here are the task parameters, ϕ, output by the inner-loop. In this cases, using one network to map all data in the trial, $\mathcal{D}$, to weights and biases of another network defines a hypernetwork [Ha et al.,

2017]. The architecture can be written as $\pi_\phi(a|s)$, where $\phi = h_\theta(\text{RNN}_\theta(\tau))$, h is a hypernetwork, and ϕ are the weights and biases of the policy. See Figure 3.1 for an illustration.

3.1.2 Inner-loop representation

While many black box methods simply use recurrent neural networks as the inner-loop representation, [Heess et al., 2015, Duan et al., 2016, Wang et al., 2016, Fakoor et al., 2020], alternative representations have also been investigated. Some of these representations are biologically inspired [Bellec et al., 2018, Miconi et al., 2018]. One of the most common biologically inspired representations leverages Hebbian learning [Miconi et al., 2018, 2019, Najarro and Risi, 2020, Chalvidal et al., 2022, Rohani et al., 2022]. Hebbian learning [Hebb, 1949] is a biologically inspired method in which weight updates are a function of the associated activations in the previous and next layers. The update to the weight ($w_{i,j}^k$) from the ith activation in layer k (x_i^k), to the jth activation in layer $k+1$ (x_j^{k+1}) generally has the form

$$w_{i,j}^k := w_{i,j}^k + \alpha(a x_i^k x_j^{k+1} + b x_i^k + c x_j^{k+1} + d),$$

where α is a learning rate and α, a, b, c, d are all meta-learned parameters of the inner-loop learned in θ. While Hebbian learning is one of the most common biologically inspired representations, most methods aligned with recent machine learning literature use some form of attention [Graves et al., 2014, Vaswani et al., 2017] to parameterize the inner-loop [Oh et al., 2016, Ritter et al., 2018a, Mishra et al., 2018, Fortunato et al., 2019, Ritter et al., 2021, Wang et al., 2021, Emukpere et al., 2021, Melo, 2022, Xu et al., 2022, Team et al., 2023, Elawady et al., 2024].

Attention can be thought of as a soft form of a key-value lookup in a dictionary. Specifically, it is a mechanism for combining different vectors based on the similarity of their associated key vectors to a given query vector. Given a query vector, q, a

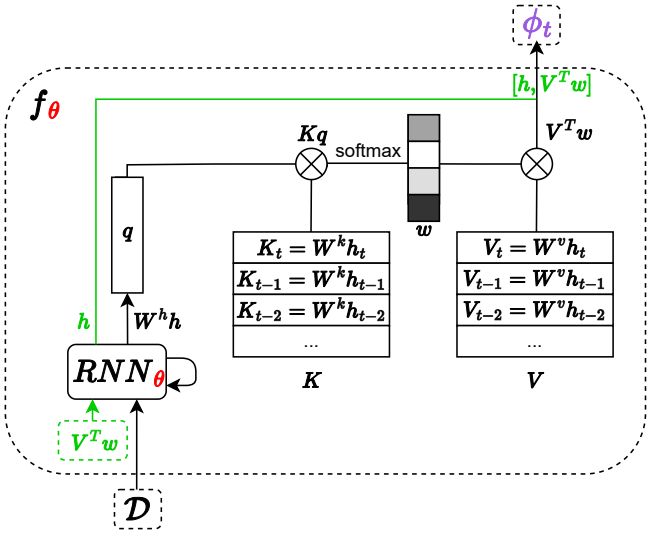


Figure 3.2: Attention over past recurrent states in the inner-loop of black box methods. One approach is to use the current hidden state as a query in attention over past recurrent states. The keys (K) and values (V) are all linear projections of the past recurrent states. The output, ϕ, is a a convex combination of the projected hidden states (V), where the combination is specified by this weight vector (w), computed from the similarity between the keys (K) and query (q). Components in green are optional. In order to provide long-term context to the RNN, the output from attention over past hidden states, $V^T w$, can be passed as an input to the RNN at the next timestep. Additionally, the output, ϕ, can be $V^T w$, h, or a concatenation of both. Passing the RNN state, h, as an output, with $V^T w$ as an input, allows multiple steps of attention to be integrated into the final output.

matrix, V, where each row is one of n value vectors over which to attend, and a matrix K, where each row is one of n key vectors, then attention can be written

$$\text{Attention}(q, K, V) = \sum_{i=0}^{i=n} w_i(q, K_i)V_i$$

where $w \in \Delta^n$ is a weight vector defining the convex combination of value vectors [Graves et al., 2014]. Generally, attention computes the weights as $w = \text{softmax}(Kq)$, leading to an attention mechanism that simplifies to

$$\text{Attention}(q, K, V) = V^T\text{softmax}(Kq) = (\text{softmax}(q^T K^T)V)^T.$$

Additionally, computing multiple queries, with each as a row vector in Q, we can write this as

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V.$$

Some methods combine attention with convolution [Mishra et al., 2018], or use attention over past recurrent states [Ritter et al., 2018a, Fortunato et al., 2019, Team et al., 2023, Oh et al., 2016], while others use self-attention alone [Ritter et al., 2021, Wang et al., 2021, Team et al., 2023]. For example, to attend to past recurrent states, q may be a function of the current hidden state of a recurrent network, while K and V may be (two different linear projections of) all prior hidden states computed over $\mathcal{D}$. A generalization of such methods can be seen in Figure 3.2. In contrast, self-attention models Q, K, and V all as linear projections of the inputs in $\mathcal{D}$ first, then as linear projections of the previous attention layer, for multiple attention layers in a row.

Attention mechanisms seem to aid in generalization to novel tasks outside of the distribution, $p(\mathcal{M})$, [Fortunato et al., 2019, Melo, 2022] and self-attention may be useful for complex planning [Ritter et al., 2021]. Still, attention is computationally expensive: whereas recurrent networks use $O(1)$ memory and compute per timestep, attention generally requires $O(t^2)$ memory and compute per timestep t, which may cross many episodes. While fast approximations of attention exist [Katharopoulos

et al., 2020], solutions in meta-RL often simply maintain only memory of a fixed number of the most recent transitions [Mishra et al., 2018, Fortunato et al., 2019]. Nonetheless, both transformers, which use self-attention, and recurrent neural networks can still struggle to meta-learn simple inductive biases, particularly for complex task distributions generated by simple underlying rules in low-dimensional spaces [Kumar et al., 2020, Chan et al., 2022].

3.1.3 Outer-loop algorithms

While many black box methods use on-policy algorithms in the outer-loop [Duan et al., 2016, Wang et al., 2016, Zintgraf et al., 2020], it is straightforward to use off-policy algorithms [Rakelly et al., 2019, Fakoor et al., 2020, Liu et al., 2021], which bring increased sample efficiency to RL. For discrete actions, it is also straightforward to use Q-learning [Fakoor et al., 2020, Liu et al., 2021, Zhang and Kan, 2022], in which case the inner-loop must change as well. In this case, the inner-loop estimates Q-values instead of directly parameterizing a policy. The policy can then act greedily with respect to these Q-values at meta-test time. This approach can be thought of as modifying recurrent Q-networks [Hausknecht and Stone, 2015] to fit the meta-RL setting, which compares favorably compared to other state-of-the-art meta-RL methods [Fakoor et al., 2020].

3.1.4 Black box trade-offs

Recall that there exist meta-parameterizations for the few-shot setting and many-shot setting that are not applicable in the zero-shot setting. Here, we briefly compare to the common approach that is applicable in the few-shot setting, but not the zero-shot setting, called *parameterized policy-gradient* methods, or *PPG* methods, for short. PPG methods build in the computation of a policy gradient into the inner-loop itself to learn a model initialization that can be adapted to new tasks with a few policy

gradient updates [Beck et al., 2023a]. We draw this comparison to highlight some of the failure modes of black box methods that we will address in this thesis.

One key benefit of black box methods is that they can rapidly alter their policies in response to new information, whereas PPG methods generally require multiple episodes of experience to get a sufficiently precise inner-loop gradient estimate. For example, consider an agent that must learn which objects in a kitchen are hot at meta-test time. While estimating a policy gradient, a PPG method may touch a hot stove multiple times before learning not to. In contrast, a black box method may produce an adaptation procedure that never touches a hot surface more than once. Black box methods can learn such responsive adaptation procedures because they represent the inner-loop as an arbitrary function that maps from the cumulative task experience to the next action.

However, black-box methods also present a trade-off. While black-box methods can tightly fit their assumptions about adaptation to a narrow distribution of data, $p(\mathcal{M})$, increasing specialization, they often struggle to generalize outside of $p(\mathcal{M})$ [Wang et al., 2016, Finn and Levine, 2018, Fortunato et al., 2019, Xiong et al., 2021]. Consider the robot chef: while it may learn to not touch hot surfaces, it is unlikely a black box robot chef will learn a completely new skill, such as how to use a stove, if it has never seen a stove during meta-training. In contrast, a PPG method could still learn such a skill at meta-test time with sufficient data. For this reason, it is critical to increase the generalization abilities of black box methods. One way to do so is by increasing the breadth of set of meta-training tasks, which we will consider in Chapter 4.

Additionally, there exist trade-offs in outer-loop optimization challenges between PPG and black box methods, providing insight into how to further improve black box methods. On one hand, PPG methods often estimate a meta-gradient, which is difficult to compute [Al-Shedivat et al., 2018], especially for long horizons [Wu

et al., 2018]. On the other hand, black box methods do not have the structure of optimization methods build into them, so they can be harder to train from scratch, and have associated outer-loop optimization challenges even for short horizons. Black box methods generally make use of recurrent neural networks, which can suffer from vanishing and exploding gradients [Pascanu et al., 2013]. Moreover, the optimization of recurrent neural networks can be especially difficult in reinforcement learning [Beck et al., 2020], while transformers can be even more problematic to train [Parisotto et al., 2020, Melo, 2022]. While large transformers have shown some notable success in meta-RL, such solutions require the use of curriculum learning and distillation to train stably [Team et al., 2023]. Thus, there also exists significant room to improve the representations of sequence modelling in black box methods, which will be the topic of Chapter 6.

3.2 Task Inference Methods

Closely related to black box methods are *task inference* methods, which often share the same parameterization as black box methods and thus can be considered a subset of them. However, parameterizations of the inner-loop may be specific to task inference methods [Rakelly et al., 2019, Korshunova et al., 2020, Zintgraf et al., 2020], which generally train the inner-loop to perform a different function by optimizing a different objective.

Task inference methods generally aim to identify the MDP, or task, to which the agent must adapt, in the inner-loop. In meta-RL, the agent must repeatedly adapt to an *unknown* MDP whose representation is not given as input to the inner-loop. While the agent ultimately acts to maximize reward, the entire purpose of the inner-loop can be described as identifying the task. The agent's belief about what it should do can be represented as a distribution over tasks. This posterior distribution constitutes a sufficient statistic, or information state [Subramanian et al., 2022], for the meta-RL

problem [Humplik et al., 2019, Beck et al., 2023a]. Since we already know the form of this sufficient statistic, the inner-loop can model it directly, instead of learning a mapping from history to action end-to-end. *Task inference* is the process of inferring this posterior distribution over tasks, conditional on what the agent has seen so far.

Consider the case where the agent has uniquely identified the task. Then, at this point, the agent knows the MDP and could in fact use classical planning techniques, such as value iteration, to compute the optimal policy directly. In this scenario, no further learning or data collection is required. More practically, if the task distribution is reasonably small and finite, we can avoid even having to explicitly plan, by learning a mapping from the task to the optimal policy directly, during meta-training. In fact, training a policy over a distribution of tasks, with the policy conditioned on the true task, can be taken as the definition of multi-task RL [Yu et al., 2020]. In the multi-task case, a mapping is learned from a known task to a policy. In meta-RL the only difference is that the task is not known. Thus task inference can be seen as an attempt to move a meta-learning problem into the easier multi-task setting.

When uncertainty remains in the distribution, instead of mapping a task to a base policy, task inference methods generally map a task distribution, given the current data, to a base policy. This can be seen as learning a policy conditioned on a (partially) inferred task. In this case, learning becomes the process of reducing uncertainty about the task. The agent must collect data that enables it to identify the task. That is, the agent must explore to reduce uncertainty in the posterior given by task inference. Task inference is therefore a useful way to frame exploration, and many task inference methods are framed as tools for exploration, which we discuss in Section 3.3.

In this section we discuss two methods for task inference that use supervised learning but also require assumptions about the information available for meta-training. We then discuss alternative methods without such assumptions, and how the inner-loops

are usually represented. Finally, we conclude with a discussion of the trade-offs concerning task inference methods. Table 3.1 summarizes these categories and task inference methods.

3.2.1 Task inference with privileged information

A straightforward method for inferring the task is to add a supervised loss so that a black box f_θ predicts some estimate of the task, $\hat{c}_\mathcal{M}$, given some known representation of the task, $c_\mathcal{M}$ [Humplik et al., 2019]. For example, a recurrent network may predict the task representation conditional on all data collected so far. Recall that ϕ, the task parameters, are the adapted policy parameters output by the inner-loop. Most commonly, ϕ is passed directly to the policy as an input vector: $\pi_\theta(a|s,\phi)$. In task inference, the vector ϕ is generally the predicted task estimate: $\pi_\theta(a|s,\phi = \hat{c}_\mathcal{M})$. For computing the supervised loss, this task representation must be known during meta-training, and so constitutes a form of *privileged information*. The representation may, for instance, be a one-hot representation of a task index, if the task distribution is discrete and finite. Or, it may be some parameters defining the MDP. For example, if kitchens differ in the location of the stove and refrigerator, the task representation, $c_\mathcal{M}$, could be a vector of all these coordinates. In this case, f_θ would predict these coordinates. Commonly, the task-inference objective, denoted here as J_{infer}, is given by the maximum likelihood estimate:

$$J_{\text{infer}}(\theta) = \mathbb{E}_\mathcal{M}[\mathbb{E}_{\mathcal{D}|\pi}[\log p_\theta(c_\mathcal{M})]].$$

When passing the task to the policy, there are a few important representation choices. First, instead of conditioning the policy on the task representation directly, we can pass a representation with more information about task uncertainty. This can be accomplished, for instance, by passing to the policy the penultimate layer when predicting $\hat{c}_\mathcal{M}$ [Humplik et al., 2019]. Then the task parameters, ϕ, passed to the

policy, are trained by inferring $\hat{c}_{\mathcal{M}}$, but only after a subsequent linear transformation: $\hat{c}_{\mathcal{M}} = L_\theta(\phi)$. This hidden layer may contain more information than $\hat{c}_{\mathcal{M}}$, and $\hat{c}_{\mathcal{M}}$ can always be computed from it by the policy. Second, it can be useful to add a stop-gradient to ϕ before passing it to the policy [Humplik et al., 2019, Zintgraf et al., 2021b], to prevent conflicting gradients.

3.2.2 Task inference with multi-task training

Some research uses the multi-task setting to improve task inference with privileged information [Humplik et al., 2019, Kamienny et al., 2020, Raileanu et al., 2020, Liu et al., 2021, Peng et al., 2021]. The task representation may contain little task-specific information (e.g., if it is one-hot representation) or task-specific information that is irrelevant to the policy (e.g the amount of oxygen in the kitchen). For example, consider the more concrete task of navigation to a goal on the perimeter of a circle. In this case, if the task representation is one-hot, it may be useful to instead have access to the $(x_{\text{goal}}, y_{\text{goal}})$ coordinates of the goal. Additionally, if the task representation contains the location of an additional irrelevant object, $(x_{\text{goal}}, y_{\text{goal}}, x_{\text{object}}, y_{\text{object}})$, then if may be useful to instead have access to a more parsimonious task descriptor, $(x_{\text{goal}}, y_{\text{goal}})$, that contains the goal location alone. In general, the entire MDP does not need to be uniquely identified. The agent only needs to identify the variations between MDPs that occur in the task distribution, $p(\mathcal{M})$. Even more specifically, the agent only needs to identify the subset of those variations that change the optimal policy.

To address uninformative and irrelevant task information, representations can be learned by pre-training in the multi-task setting [Humplik et al., 2019, Liu et al., 2021]. Let $g_\theta(c_{\mathcal{M}})$ be a function that encodes the task representation. First an informed policy, $\pi_\theta^{\text{multi}}(a|s, g_\theta(c_{\mathcal{M}}))$, can be trained. This is the multi-task phase, and enables the learning of g_θ. Since this representation, $g_\theta(c_{\mathcal{M}})$, is learned end-to-end,

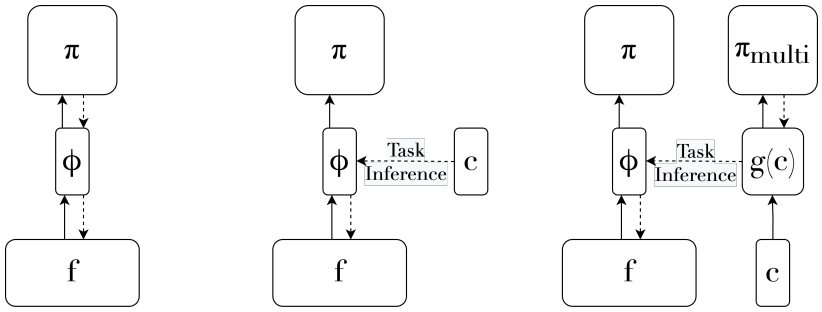


Figure 3.3: Illustration of normal meta-RL (left), task inference using privileged information (middle), task inference using multi-task training (right). Solid arrows represent forward propagation, and dashed arrows represent backpropagation. Task inference with privileged information uses a true task representation, c, and then backpropagates that inference to train ϕ. Task inference with multi-task training learns an encoding of the task, $g(c)$, to maximize the return of an informed policy, π_{multi}, then uses inference of this learned encoding to train ϕ.

it contains the information relevant for solving the task. For example, g_θ could transform $c_\mathcal{M}$ from $(x_{\text{goal}}, y_{\text{goal}}, x_{\text{object}}, y_{\text{object}})$ to $(x_{\text{goal}}, y_{\text{goal}})$. Often, an information bottleneck [Alemi et al., 2017] is used to ensure it contains only this information [Humplik et al., 2019, Liu et al., 2021]. After this, the inner-loop can infer the learned representation, $g_\theta(c_\mathcal{M})$, from the meta-trajectory, which we write as $\hat{g}_\theta(\mathcal{D})$ (see Figure 3.3). For example, in the circle navigation task, after learning the coordinate representation, $(x_{\text{goal}}, y_{\text{goal}})$, the inferred $\hat{g}_\theta$ could be represented by the inferred coordinates, $(\hat{x}_{\text{goal}}, \hat{y}_{\text{goal}})$. Alternatively, the informed multi-task RL may be performed concurrently with the meta-RL [Kamienny et al., 2020].

For example, the informed policy regularization algorithm (IMPORT) [Kamienny et al., 2020] follows the simultaneous-training paradigm. In this case, the auxiliary inference objective is given by

$$J_{\text{infer}}(\theta) = \mathbb{E}_\mathcal{M}[\mathbb{E}_{\mathcal{D}|\pi}[(g_\theta(c_\mathcal{M}) - \hat{g}_\theta(\mathcal{D}))^2]].$$

This inference objective forces the encoding of g and $\hat{g}$ to look similar. While optimizing this objective, the algorithm also optimizes the normal meta-learning objective (see Equation 2.3) but with the policy conditioned on the inferred task

representation: $\pi_\theta(a|s, \hat{g}_\theta(\mathcal{D}))$. Additionally, IMPORT simultaneously optimizes the meta-RL objective but with the multi-task policy conditioned on the learned task representation: $\pi_\theta^{\text{multi}}(a|s, g_\theta(c_\mathcal{M}))$.

Some task distributions even allow for significant shared behavior between the informed multi-task agent and the uninformed meta-RL agent. This sharing is generally possible when little exploration is needed for the meta-RL policy to identify the task. In this case instead of only inferring the privileged task information, the meta-RL agent may imitate the multi-task agent through distillation [Weihs et al., 2021, Nguyen et al., 2022], or by direct parameter sharing of policy layers [Kamienny et al., 2020, Peng et al., 2021]. For instance, in the IMPORT algorithm, π_θ and $\pi_\theta^{\text{multi}}$ share parameters by using the same components of the meta-parameters vector, θ. In this case, the entire policy is shared, such that $\pi = \pi^{\text{multi}}$. Moreover, in Chapter 5 we will demonstrate that when the multi-task and meta-RL policies are computed sequentially, the pre-trained multi-task agent may still be used as an initialization for the meta-RL agent.

In contrast, when task distributions require taking sufficiently many exploratory actions to identify the task, sharing policies becomes less feasible. For example, in the circle navigation task, the informed multi-task policy, with access to the goal, never needs to explore, whereas the primary behavior of the meta-RL policy is exploration around the circumference of the circle. If one were to reuse the multi-task policy conditioned on the inferred goal, $\pi_\theta = \pi_\theta^{\text{multi}}(a|s, \hat{x}_{\text{goal}}, \hat{y}_{\text{goal}})$, as the meta-RL policy, then the inference would not have sufficient data to be accurate. The initial inferred goal location, for example, may be $(0, 0)$, which is not in the training distribution of the multi-task agent. In this case, the behavior of the multi-task agent would be undefined. However, sharing some parameters, simultaneously training the shared policy on inferred representation [Kamienny et al., 2020], or fine-tuning the shared policy as an initialization for the meta-RL policy (Chapter 5), can solve this issue.

For example, if training the shared policy simultaneously with known and inferred goal locations, $(x_{\text{goal}}, y_{\text{goal}})$ and $(\hat{x}_{\text{goal}}, \hat{y}_{\text{goal}})$, the agent can learn to execute exploratory behavior whenever the inferred task is $(0, 0)$, or has high uncertainty in the posterior distribution.

3.2.3 Task inference without privileged information

Other task inference methods do not rely on privileged information in the form of the known task representation. If the algorithm is allowed to know whether two trajectories were generated by the same task, then one option is to use this label as a learning signal for the encoder [Fu et al., 2021, Mu et al., 2022, Choshen and Tamar, 2023, Guo et al., 2018, Luo et al., 2022]. More commonly, the representation can be (a sample from) a latent variable parameterizing a learned reward function or transition function [Zhou et al., 2019, Zintgraf et al., 2020, Zhang et al., 2021, Zintgraf et al., 2021b, He et al., 2022]. These methods use only information already observable and train $f_\theta(\mathcal{D})$ to represent a task distribution, given trajectories in $\mathcal{D}$.

For example, Zintgraf et al. [2020] propose to learn the task parameters, ϕ, as a latent variable parameterizing the reward and transition function. In this case, the latent is trained in a self-supervised manner by reconstructing trajectories. This is accomplished using only observable information by predicting rewards and next states for each transition, given each ϕ_t:

$$\begin{aligned} J_{\text{infer}}(\theta) &= \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\mathcal{D}|\pi}[\sum_{t=0}^{HT} \log p_\theta(\mathcal{D}|\phi_t)]], \\ &= \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\mathcal{D}|\pi}[\sum_{t=0}^{HT} \sum_{s,a,r,s' \in \mathcal{D}} [\log p_\theta(s'|s,a,\phi_t) + \log p_\theta(r|s,a,\phi_t)]]]. \end{aligned}$$

An extension of this algorithm has even been proposed with a hierarchical latent variable to accommodate the additional structure in distributions of POMDPs [Akuzawa et al., 2021].

3.2.4 Inner-loop representation

Generally, task inference is implemented by adding an additional loss function, and not by any particular meta-parameterization of f_θ. While task inference methods do not require a particular meta-parameterization, most implementations use a "black box," such as a recurrent neural network [Humplik et al., 2019, Zintgraf et al., 2020, Kamienny et al., 2020, Liu et al., 2021, Zintgraf et al., 2021c]. Since many task inference methods infer a latent variable, it is also common for f_θ to explicitly model this distribution using a variational information bottleneck [Humplik et al., 2019, Rakelly et al., 2019, Zintgraf et al., 2020, Liu et al., 2021, Zintgraf et al., 2021c].

In this case, the variational distribution defining the latent variable is conditioned on $\mathcal{D}$. We write this as $q_\theta(z|\mathcal{D})$. This variational distribution has its own parameters inferred from the dataset, such as a mean, $\mu_\theta(\mathcal{D})$, and a covariance matrix $diag(\sigma_\theta(\mathcal{D}))$. First, the task representation is reconstructed, e.g., with a linear projection of samples from q_θ: $\hat{c}_\mathcal{M} = L^c_\theta(z \sim q_\theta)$. Then, the inference is trained by maximizing J_{infer}, also called the reconstruction loss, over samples from the variational distribution:

$$J_{\text{infer}}(\theta) = \mathbb{E}_\mathcal{M}[\mathbb{E}_{\mathcal{D}|\pi}[p_\theta(c_\mathcal{M}|\hat{c}_\mathcal{M})]],$$

along with a KL-Divergence to a prior,

$$J_{\text{prior}}(\theta) = \mathbb{E}_\mathcal{M}[\mathbb{E}_{\mathcal{D}|\pi}[D(q_\theta(z)||p_\theta(z)]].$$

Once the task inference is trained, the parameters passed to the policy, ϕ, must be chosen. One option is to represent ϕ as a projection of the full distribution of the latent variable, e.g., its mean and variance, in order to capture uncertainty about the task [Zintgraf et al., 2020, Imagawa et al., 2022]. This can be written $\pi_\theta(a|s, \phi = L^\phi_\theta(\mu, \sigma))$. Combining this representation of the latent distribution with the J_{infer} for trajectory reconstruction above, we arrive at a canonical method, variational Bayes-Adaptive Deep RL (VariBAD) [Zintgraf et al., 2021b]. This method is depicted in Figure 3.4.

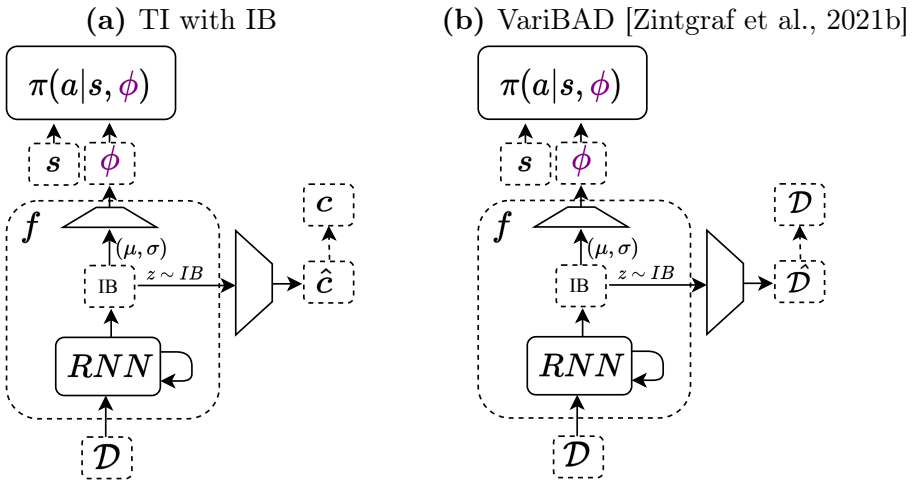


Figure 3.4: Task inference with an information bottleneck (IB) (a), task inference with an IB and without privileged information, as in Zintgraf et al. [2021b] (b). In either case, the parameters of a latent distribution are passed to the policy. When using privileged information, the task representation, c, is reconstructed from samples of this distribution; otherwise, the set of trajectories, $\mathcal{D}$, is reconstructed.

Alternatively, it is possible for the policy to condition on samples from the information bottleneck (i.e., $\pi_\theta(a|s, \phi \sim q_\theta)$), instead of conditioning on the parameters of the information bottleneck directly [Rakelly et al., 2019]. In this case, the task inference can be trained entirely from the actor and critic loss given by the meta-learning objective, rather than a distinct task inference objective. Rakelly et al. [2019], Wen et al. [2022] use such probabilistic embeddings for actor-critic RL, as introduced by the PEARL method [Rakelly et al., 2019].

Several methods also make use of the exchangeability, or permutation invariance, of the transitions implied by the Markov property in task inference [Rakelly et al., 2019, Korshunova et al., 2020, Raileanu et al., 2020, Imagawa et al., 2022]. PEARL specifically models the task distribution conditioned on $\mathcal{D}$ as a product of individual distributions conditioned on each transition in $\mathcal{D}$:

$$\phi \sim q_\theta(z|\mathcal{D}) \propto \prod_{s,a,r,s' \in \mathcal{D}} q_\theta(z|s,a,r,s').$$

While it is not necessary to model this permutation invariance, standard sequence

models are sensitive to the ordering of their inputs, which can can be detrimental to learning when the order does not matter [Beck et al., 2020]. For this reason, it may be useful to embed the structure of permutation invariance into the task inference model as an inductive bias. Specifically, PEARL uses a product of Gaussians, $q_\theta(z|s,a,r,s') = \mathcal{N}(z; \mu_\theta(s,a,r,s'), diag(\sigma_\theta(s,a,r,s')))$, which has a simple closed form for the joint mean and covariance. However, other methods forgo the product entirely, and use more general permutation invariant representations, such as neural processes [Garnelo et al., 2018b, Wang and van Hoof, 2022] and transformers [Vaswani et al., 2017].

Finally, the use of hypernetworks in the inner-loop is an open question in meta-RL, which we explore ine Chapter 4. It is possible to use a neural network to map an inferred task to the weights and biases of a base network: $\pi_{h_\theta(\hat{c}_\mathcal{M})}(a|s)$. In this case, the mapping is a hypernetwork, and it can be trained end-to-end with the meta-learning objective, although this has not been explored prior to this thesis. One method, FLAP does evaluate hypernetworks in a similar setting [Peng et al., 2021]. However, FLAP learns to explicitly infer a set of expert weights trained in the multi-task setting, but never trains in the meta-RL setting, which constrains the adaptation process. (For example, FLAP cannot learn to explore to reduce uncertainty.) Still, using expert weights from the multi-task setting as explicit targets for additional supervision is explored in Chapter 5.

3.2.5 Task inference trade-offs

Task inference methods present trade-offs in comparison to other methods, which motivate the work of this thesis. First, we consider the comparison to PPG methods then we consider the comparison to black box methods. In comparison to PPG methods, task inference methods can adapt faster, but are less capable of broad generalization. PPG methods are unlikely to recover an algorithm as efficient as task

inference, and so are not generally applicable to the zero-shot setting considered here, as discussed in Section 3.1. For distributions where task inference is possible, fitting such a method to the task distribution enables faster adaptation. However, PPG methods generalize well to novel tasks because of the additional structure of the policy gradient hard coded into their inner-loop. In the case where a novel task cannot be represented using the task representations learned during meta-training, task inference methods fail [Rimon et al., 2022]. This comparison underscores the need for methods to improve the generalization capabilities of task-inference methods, as we propose in Chapter 4.

In comparison to black box methods, task inference methods impose more structure. On the one hand, task inference methods often add additional supervision through the use of privileged information [Humplik et al., 2019, Kamienny et al., 2020, Liu et al., 2021] or the use of self-supervision [Zintgraf et al., 2020, 2021c], which may make meta-training more stable and efficient [Humplik et al., 2019, Zintgraf et al., 2020]. This is particularly useful since recurrent policies, often used in task inference and black box methods, are difficult to train in RL [Beck et al., 2020]. On the other hand, task inference methods tend to be more complicated to implement and training the inner-loop to accomplish any objective that is not Equation 2.3 may be suboptimal with respect to that meta-RL objective over the given task distribution. Thus, if we can find black box methods that train as stably and efficiently as task inference methods, this would represent a significant improvement to meta-RL methods. Surprisingly, in Chapter 5, we show precisely that this is achievable using novel methods presented in this thesis.

3.3 Exploration and Meta-Exploration

Exploration is the process by which an agent collects data for learning. In standard RL, exploration should work for any MDP and may consist of random on-policy

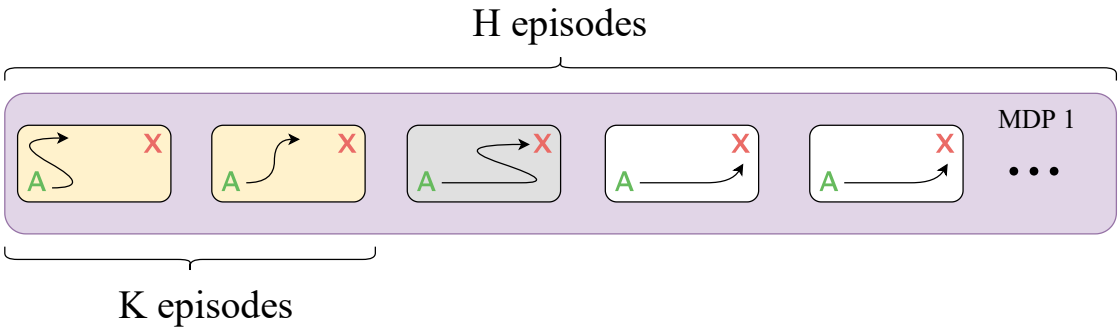


Figure 3.5: Illustration of "free" exploration in first K episodes (yellow), followed by not free exploration (gray), followed by exploitation (white). An agent (A) is trying to identify how to get to a goal location (X). The agent has K shots, or free episodes to explore. (In the zero-shot setting, $K = 0$ and there are no free episodes.) Since, K episodes may not be enough, in the $(K + 1)$th episode, the agent is still exploring the map to find the goal, and is penalized for this. In the remaining two episodes, the agent has learned to navigate to the goal optimally, once the goal has been found, and no longer needs to explore.

exploration, epsilon-greedy strategies, or methods to find novel states. In meta-RL, this type of exploration still occurs in the outer-loop, which is called *meta-exploration*. However, there additionally exists exploration in the inner-loop, referred to as just *exploration*, which is where we begin our discussion. This inner-loop exploration is specific to the distribution of MDPs, $p(\mathcal{M})$. To enable sample efficient adaptation, the meta-RL agent uses knowledge about the task distribution to explore efficiently. For instance, instead of taking random actions, the robot chef may open every cabinet to learn about the location of food items and utensils when first entering a new kitchen. This exploration is targeted and used to provide informative trajectories in $\mathcal{D}$ that enable zero-shot adaptation to the MDP within the task distribution.

Recall that in the few-shot adaptation setting, on each trial, the agent is placed into a new task and is allowed to interact with it for a few episodes (i.e., its few shots K), before being evaluated on solving the task in the next few episodes (i.e., over the $H - K$ episodes in Equation 2.3). An illustration can be seen in Figure 3.5. Intuitively, the agent must explore to gather information during the first few shots

Sub-topic	Papers
End-to-end components	Stadie et al. [2018], Garcia and Thomas [2019], Boutilier et al. [2020]
Posterior sampling	Gupta et al. [2018], Rakelly et al. [2019], Kveton et al. [2021], Simchowitz et al. [2021]
Task Inference	Zhou et al. [2019], Gurumurthy et al. [2020], Wang et al. [2020a], Liu et al. [2021], Fu et al. [2021], Zhang et al. [2021]
Meta-exploration	Zintgraf et al. [2021c], Grewal et al. [2021]

Table 3.2: zero-shot meta-RL research categorized by exploration method as described in Section 3.3. End-to-End methods learn to explore implicitly, by directly maximizing the meta-RL objective. Posterior sampling maintains a distribution over possible tasks and acts optimally with respect to samples from this distribution. Task inference guides exploration in order to enable better inference of the task. Meta-exploration concerns exploration in the outer-loop.

that enables it to best solve the task in later episodes. More generally, there is an exploration-exploitation trade-off, where the agent must balance taking exploratory actions to learn about the new task (potentially even beyond the initial few shots) with exploiting what it already knows to achieve high rewards. It is always optimal to explore in the first K episodes, since no reward is given to the agent. However, exploration is often still needed past the first K episodes. This is especially true in the zero-shot problem setting of this thesis, where $K = 0$. However, the optimal amount of exploration in the remaining episodes depends on the size of the evaluation period, $H - K$ in general, and H in the zero-shot setting: When the evaluation period is long, more exploration is optimal, as sacrificing short-term rewards to learn a better policy for higher later returns pays dividends, while when the evaluation period is small, the agent must exploit more to obtain any reward it can, before time runs out. In this section, we survey approaches that navigate this trade-off. Table 3.2 summarizes these categories.

End-to-end optimization Perhaps the simplest approach is to learn to explore and exploit *end-to-end* by directly maximizing the meta-RL objective (Equation 2.3) as done by black box meta-RL approaches [Duan et al., 2016, Wang et al., 2016, Mishra et al., 2018, Stadie et al., 2018, Boutilier et al., 2020]. Approaches in this category implicitly learn to explore, as they directly optimize the meta-RL objective whose maximization requires exploration. More specifically, the returns in the later episodes $\sum_{\tau \in \mathcal{D}_{K:H}} R(\tau)$ can only be maximized if the policy appropriately explores in the early episodes, so maximizing the meta-RL objective can yield optimal exploration in principle. However, when more complicated exploration strategies are required, learning exploration this way can be extremely sample inefficient. One issue is that learning to exploit in the later episodes requires already having explored in the early episodes, but exploration relies on good exploitation to provide reward signal [Liu et al., 2021]. For example, in the robot chef task, the robot can only learn to cook (i.e., exploit) when it has already found all of the ingredients, but it is only incentivized to find the ingredients (i.e., explore) if doing so results in a cooked meal. Hence, it is can be challenging to learn to exploit without already having learned to explore and vice-versa, and consequently, end-to-end methods may struggle to learn tasks requiring sophisticated exploration in a sample-efficient manner, compared to methods with more structure, discussed later in this section.

One method that learns exploration end-to-end also modifies the reward to encourage exploration. E-RL^2 [Stadie et al., 2018] is an end-to-end method that sets all rewards in the first K episodes to zero in the outer-loop. While ignoring these rewards does introduce sparsity, it may be helpful when myopically maximizing an immediate, dense reward prevents exploration needed for a longer-term reward. In general, many methods add more structure over end-to-end optimization of the meta-RL objective.

Algorithm 1 PEARL Inner-Loop

1: Sample task $\mathcal{M} \sim p(\mathcal{M})$
2: Initialize empty meta-trajectory, τ
3: **for** each shot $k = 0, ..., H-1$ **do**
4: Sample $\phi \sim q_\theta(z|\tau)$
5: Roll out $\pi_\theta(a|s, \phi)$ to collect trajectory more data
6: Update τ with new data
7: **end for**

Posterior sampling To circumvent the challenge of implicitly learning to explore, Rakelly et al. [2019] propose meta-RL methods that explore via posterior sampling [Strens, 2000, Osband et al., 2013], an extension of Thompson sampling [Thompson, 1933] to MDPs. The inner-loop of this method, PEARL, is described in Algorithm 1. When the agent is placed in a new task, the general idea is to maintain a distribution over what the identity of the task is, and then to iteratively refine this distribution by interacting with the task until it roughly becomes a point mass on the true identity. Posterior sampling achieves this by sampling an estimate of the task identity from the distribution on each episode, and acting as if the estimated task identity were the true task identity for the episode. Then, the observations from the episode are used to update the distribution either with black box methods [Rakelly et al., 2019] or directly via gradient descent [Gupta et al., 2018].

Posterior sampling does, however, also have a drawback. First, since the policy employed is always conditioned on a sampled task, all of the exploration in such a method is executed by a policy that assumes it knows the task it is in. This means that the same policy is used for both exploration and exploitation. This can lead to suboptimal exploration in terms of Equation 2.3. Consider a robot chef that has to find a stove along a curved kitchen counter. Optimally exploring, the chef walks along the perimeter of the counter until it finds the stove. If the chef must be reset to its initial position, e.g., to charge its battery at the end of an episode, then the robot resumes exploration where it last left off. In contrast, a chef using posterior sampling,

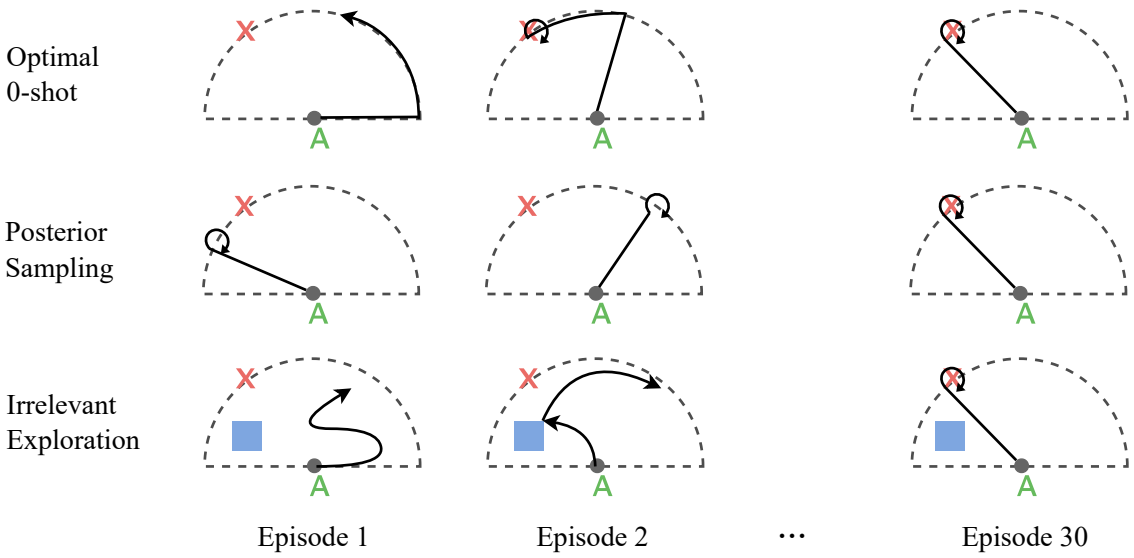


Figure 3.6: Exploration for a robot chef (green) finding a stove (red) along a circular counter. The first two rows compare optimal exploration and posterior sampling. The third row shows excessive exploration (blue) when irrelevant information is available in the sampled MDP.

in each episode, simply walks to a different point along the counter that it has not yet checked, repeating this process until it finds the stove. For this reason, it is generally not suitable to zero-shot exploration setting, and has been shown empirically to not be competitive in that setting [Zintgraf et al., 2021b].

This comparison is depicted in Figure 3.6. Other methods for exploration in meta-RL exist that add structure to exploration without this limitation.

Task inference There is often a tight relationship between task inference methods and exploration. Since task inference methods explicitly represent the inferred task, or the posterior over the task distribution, they can directly represent the exploration conducted so far, without having to learn this representation via its effect on downstream rewards. Such a representation can enable more effective learning of rapid exploration strategies [Zintgraf et al., 2020, Humplik et al., 2019].

Additionally, task inference methods can enable better meta-exploration strategies.

Specifically, these methods often add an intrinsic reward to gather information that removes uncertainty from the task distribution. In other words, these methods train the policy to explore states that facilitate task prediction. For example, these intrinsic rewards generally incentivize improvement in transition predictions (i.e., in adapting the dynamics and reward function) [Zhou et al., 2019] or incentivize information gain over the task distribution [Fu et al., 2021, Liu et al., 2021, Zhang et al., 2021]. The idea is that recovering the task is sufficient to learn the optimal policy, and hence achieve high returns in the later episodes. However, these rewards can incentivize more exploration than necessary, and these methods often require free episodes for a separate exploration policy [Zhou et al., 2019, Gurumurthy et al., 2020, Liu et al., 2021, Fu et al., 2021], limiting their applicability in the zero-shot setting. Thus, in this thesis we focus on the effect of task inference on representation. However, we briefly discuss connections to meta-exploration, which we summarize next.

Meta-exploration Finally, in meta-RL, there is still the process of acquiring data for the outer-loop learning, just as in standard RL. This is called *meta-exploration*, since it must explore the space of exploration strategies. While meta-exploration can be considered exploration in the outer-loop, both loops share data, and exploration methods may affect both loops, so the distinction may be blurry. Often, sufficient meta-exploration occurs simply as a result of the exploration of the standard RL algorithm in the outer-loop. However, a common method to specifically address meta-exploration is the addition of an intrinsic reward. In fact, the addition of any task inference reward, discussed in the previous paragraph, can be considered meta-exploration. This is particularly apparent when considering that this intrinsic reward can be used to train a policy exclusively for off-policy data collection during meta-training. In addition to task inference methods, other methods exist for including intrinsic rewards to improve meta-exploration.

Alternatively, intrinsic rewards for meta-exploration can be added that function similarly to those in standard RL. For example, using random network distillation (RND) [Burda et al., 2019], a reward may add an incentive for novelty [Zintgraf et al., 2021c]. In this case, the novelty is measured in the joint space of the state and task representation, instead of just in the state representation, as in standard RL. For example, HyperX [Zintgraf et al., 2021c], a method that build off of the VariBAD method discussed earlier, adds the reward,

$$r^{\text{hyper}}(\phi, s) = ||f(\phi, s) - h(\phi, s)||,$$

where $\phi = L_\theta(\mu, \sigma)$ represents a projection of the task distribution, as in VariBAD; f represents the predictor network in RND; and h represents the random prior network in RND. This reward can be seen as a more general exploration methods for POMDPs [Yin et al., 2021] applied to the meta-RL problem setting.

3.4 Statement of Authorship

Chapter 3 is based on content from **A Tutorial on Meta-Reinforcement Learning** [Beck et al., 2023a, 2025b].

Content from Beck et al. [2023a, 2025b] is additionally included in the thesis: Vuorio, Risto. "Advances in Meta-Reinforcement Learning and Imitation Learning." PhD diss., University of Oxford, 2024.

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the co-lead author on this manuscript. I was responsible for the writing, structure, and surveying, along with the other co-lead. I had a particular responsibility for content related to few-shot and zero-shot meta-RL, whereas the other co-lead focused on many-shot methods, not germane to this thesis. I additionally reviewed, edited, and re-wrote all content as needed.

Citation:

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. "A Tutorial on Meta-Reinforcement Learning." *Foundations and Trends in Machine Learning*, 2025.

4

Initialization in Hypernetworks

Contents

4.1 Introduction

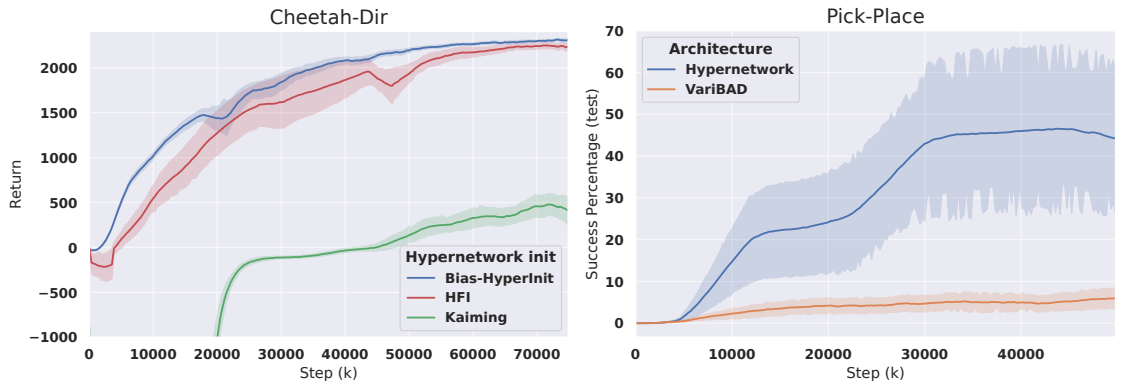
As we have seen in Chapters 2 and 3, meta-RL aims to improve sample efficiency of adaptation to novel tasks by generalizing over a distribution of related tasks. While

such an approach can enable rapid adaptation, such generalization over the task distribution has proven difficult in practice. In fact, even in the easier multi-task RL setting (see Chapter 2), methods often fail to outperform a degenerate solution that simply trains a separate policy for each task [Yu et al., 2020].

One promising way to improve generalization is with a *hypernetwork*, a neural network that produces the parameters for another network, called the *base network* [Ha et al., 2017]. In multi-task RL, using a hypernetwork that conditions on the task ID to generate task-specific parameters can replicate the separate policies of the degenerate solution, while also allowing generalization across tasks. Furthermore, unlike the degenerate solution, hypernetworks can also be applied to meta-RL, where task IDs are not provided and test tasks may be novel, by conditioning them on the output of a task encoder, using task inference methods, discussed in Chapter 3.

However, hypernetworks come with their own challenges. Since hypernetworks generate base network parameters, the initialization of parameters in the hypernetwork determines the initialization of the base network it produces. Evidence suggests hypernetwork performance is highly sensitive to the initialization scheme in supervised learning [Chang et al., 2020]. However, to our knowledge this question has not been considered in meta-RL. In this chapter, we show that hypernetwork initialization is also a critical factor in meta-RL, and that naive initializations yield poor performance.

Furthermore, we propose two novel initialization schemes: *Bias-HyperInit* and *Weight-HyperInit*. Both produce strong results, with the former matching or exceeding the performance of the state-of-the-art hypernetwork initialization method designed for supervised learning [Chang et al., 2020]. Moreover, both proposed methods are simpler and more general than this existing method, in that they may be applied to arbitrary base network architectures and target base network initializations without additional derivation. Using Bias-HyperInit, we present results that substantially improve a



(a) Hypernetwork initialization methods **(b)** Architectures

Figure 4.1: Naive initializations such as Kaiming [He et al., 2015] fail for hypernetworks, whereas our proposed Bias-HyperInit does not and matches the state of the art, HFI [Chang et al., 2020] (claims 1, 2). Adding hypernetworks with the proposed Bias-HyperInit significantly improves the state-of-the-art meta-RL method, VariBAD [Zintgraf et al., 2021b] (claim 3).

state-of-the-art method on a range of meta-RL benchmarks.

Applying hypernetworks to meta-RL, we make the following contributions (examples in Figure 4.1):

1. We empirically demonstrate that initialization is a critical factor in the performance of hypernetworks in meta-RL, and that naive initializations fail to learn reliably;

2. We propose a novel hypernetwork initialization scheme that matches or exceeds the performance of a state-of-the-art approach proposed for supervised settings, as well as being simpler and more general; and

3. We use this method to show that hypernetworks can improve a state-of-the-art method on a range of meta-RL benchmarks (grid-world [Zintgraf et al., 2021b], MuJoCo [Todorov et al., 2012], and Meta-World [Yu et al., 2020]).

4.2 Related Work

Meta-RL. Despite the advantages of hypernetworks [Ha et al., 2017], they remain relatively unexplored in meta-RL. We use hypernetworks to arbitrarily update a policy's parameters at every time-step, whereas all prior work we are aware of restrict this procedure in some way. Many procedures in few-shot meta-RL build off of MAML [Finn et al., 2017] to adapt the parameters of a policy network using a policy gradient [Finn et al., 2017, Vuorio et al., 2019, Li et al., 2017]. Such methods require the estimation of a policy gradient, which reduces sample-efficiency when faster adaptation is possible, as in our benchmarks [Zintgraf et al., 2021b]. Most meta-learning procedures capable of zero-shot adaption use a sequence model (such as an RNN or attention) that can represent an arbitrary update function [Zintgraf et al., 2021b, Mishra et al., 2018, Duan et al., 2016]. These methods generally update a set of activations on which a fixed policy is then conditioned, whereas hypernetworks update all policy parameters. We include a state-of-the-art method from this class in our evaluations [Zintgraf et al., 2021b]. There are also unsupervised methods in zero-shot meta-RL for weight updates [Najarro and Risi, 2020, Miconi et al., 2018] but none can produce a fully general learning procedure since they make use of local and unsupervised heuristics. Sarafian et al. [2021] use hypernetworks in the context of meta-RL, but the policy network, not the hypernetwork, is conditioned on the RNN used for adaptation, preventing the hypernetwork from representing a general learning procedure. Finally, FLAP [Peng et al., 2021] learns to infer a set of weights trained in the multi-task setting; however since the adaptation procedure is not trained on a meta-RL objective, it is constrained. For example, FLAP cannot learn to explore to reduce uncertainty. Finally, Xian et al. [2021] use hypernetworks to predict model dynamics then use model predictive control. However, this model still requires planning to make use of an uncertain model, whereas model-free RL learns a

policy that explores optimally in order to attain data for adaptation. Using a general procedure trained to arbitrarily modify the weights of a model-free policy has never been tried in RL, to the best of our knowledge.

Hypernetworks. Hypernetworks, or similar architectures, have been used in supervised learning (SL), multi-task RL, and meta-SL. Hypernetworks have been used in the supervised learning literature for sequence modelling [Ha et al., 2017], as well as in continual learning and image classification [Chang et al., 2020], where it was shown that the hypernetwork initialization scheme was crucial for performance. Similar models have also been used in multi-task RL and meta-SL, but not meta-RL. For instance, in multi-task RL, Yu et al. [2019] use a network conditioned on a task encoding to produce the weights and biases for every other layer in another network conditioned on state. In meta-SL, there have also been attempts to use one network to adapt weights of another, both as a general function of the dataset [Rusu et al., 2019, Munkhdalai and Yu, 2017, Przewiezlikowski et al., 2022], conditioned on an embedding adapted by gradient descent [Zhao et al., 2020], and by adding deltas in a way framed as learning to optimize [Ravi and Larochelle, 2017, Li and Malik, 2017]. The abundance of representations in meta-SL suggest there is a similarly large space of representation-based methods to explore in meta-RL. Our work – getting hypernetworks to work in practice for meta-RL – can be seen as a first step towards applying all of these methods in meta-RL.

4.3 Background

4.3.1 Architecture and Task Inference

We consider meta-RL agents capable of adaptation at every time-step, and adaptation within one episode is required to solve some of our benchmarks. In such methods [Zintgraf et al., 2021b, Mishra et al., 2018, Duan et al., 2016], the history is generally

summarized by a function, g, into an embedding that represents relevant task information. We write this embedding as $e = g(\tau)$, and call g the task encoder. The policy, represented as a multi-layer perceptron, then conditions on this task embedding as an input, instead of on the history directly, which we write as $\pi_\theta(a|s, \phi = e)$, or just $\pi_\theta(a|s, e)$. We call this the *standard architecture*, shown in Figure 4.2.

In this chapter, we primarily build upon VariBAD [Zintgraf et al., 2021b], which can be seen as an instance of the standard architecture where the task encoder is the mean and variance from a recurrent variational auto-encoder (VAE) [Kingma and Welling, 2013] trained using a self-supervised loss. In other words, the task is inferred as a latent variable optimized for reconstructing a meta-episode. See Chapter 3 or Zintgraf et al. [2021b] for details. Additionally, we evaluate the addition of hypernetworks to RL2 [Duan et al., 2016] on the most challenging benchmark. (See Section 4.5.2.) In RL2, the task encoder is a recurrent neural network trained end-to-end on Equation 2.3.

4.3.2 Hypernetwork Initialization

Chang et al. [2020] show that applying existing initialization methods for neural networks to hypernetworks produces unstable base network initializations with exploding or vanishing activations. Furthermore, they empirically demonstrate a reduction in training stability for Kaiming initialization [He et al., 2015]. We corroborate this failure for Kaiming initialization on meta-RL, as well as for Orthogonal initialization [Saxe et al., 2014] and Normc initialization [Dhariwal et al., 2017], which we collectively refer to as *default initializations.*

As a solution, Chang et al. [2020] propose the first initialization designed for hypernetworks and show it to be effective in supervised learning. Their approach is based on Kaiming initialization, which samples network parameters such that the activations of the network at each layer maintain the same variance as in the previous layer. Chang

et al. [2020] extend this variance analysis to hypernetworks [Ha et al., 2017]. They propose two methods: Hyperfan-in (HFI) sets the variance of the initial parameter distribution of the hypernetwork to maintain constant variance of activations in the base network in the forward pass, and hyperfan-out (HFO) does the same for the backward pass. Since these are equally competitive, and produce state-of-the-art results, we include HFI as a baseline for comparison. However, this variance analysis is involved and requires modification for specific use cases depending on the activation function and whether or not the network produces weights or biases in the base network. This motivates the need for a simpler and more general initialization method, which we propose in this chapter.

4.4 Methods

In this section we introduce our proposed architecture using hypernetworks and our proposed hypernetwork initialization. At a high level, the hypernetwork conditions on a task representation to generate all of the parameters for a base policy. The initialization provides a simple and general way to sample parameters for this hypernetwork at the start of training.

4.4.1 Hypernetworks and Task Inference

We propose the use of hypernetworks in meta-RL, instead of the standard architecture described earlier. In this setting, we use a hypernetwork to arbitrarily adapt the parameters of the base policy. We still use a task encoder, $e = g(\tau)$, but instead of conditioning a policy on these activations, we use a hypernetwork, h_θ, to generate policy parameters, $\phi = h_\theta(e)$. These parameters, i.e., weights and biases, are then used directly for the base network, which we write: $\pi_\phi(a|s)$. This is also depicted in Figure 4.2.

In this case, the hypernetwork can arbitrarily adapt all the the parameters of π based on

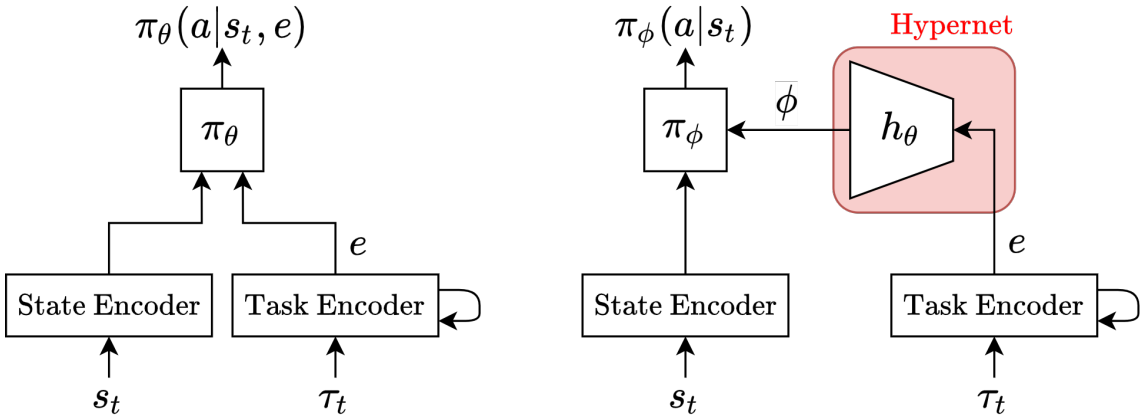


Figure 4.2: A standard architecture (left) and hypernetwork model (right). The standard architecture computes a task encoding, e, that is passed as an input to the policy network. The hypernetwork, in contrast, conditions on the task encoding to produce weights and biases, ϕ, for the base network.

history. In comparison, in the standard architecture, the shared fixed parameters of the base network (θ) are still required to generalize between all of the tasks. Since training separate policies for each task often performs better than state-of-the-art methods for generalizing across all tasks [Yu et al., 2020], this motivates the ability to produce base policies with no or few shared parameters. Hypernetworks allow for shared parameters when possible, but also provide the ability to have no shared parameters in the base policy when diverse policies are necessary and little generalization between tasks is possible.

In fact, hypernetworks are capable of replicating the training of separate policies for each task under certain conditions. To see this, consider the case where the hypernetwork is linear and has no bias. Then, the hypernetwork consists only of a weight matrix, W. (That is, $\theta = W$.) If this hypernetwork conditions on a one-hot task ID for task i: $e = \mathbb{1}_i$, then the parameters selected by this hypernetwork, ϕ are a separate set of parameters for each task: $\phi^i = h(e) = W\mathbb{1}_i$. In other words, training individual networks for each task is equivalent to training a hypernetwork when that hypernetwork is: 1) linear, 2) has no a bias, and 3) is conditioned on a one-hot task ID.

However, we can relax these assumptions and still retain the ability to produce distinct parameters, while also enabling generalization. If we add a bias, we reintroduce shared parameters in the hypernetwork, but they can still produce separate base networks for each task when little transfer between the policies is required. If we relax the one-hot assumption, the network is still capable of producing a one-hot encoding when the tasks are discrete and the task embedding is sufficiently large. Relaxing these restrictions allows for both generalization and the application of hypernetworks to meta-RL.

4.4.2 Hypernetwork Initialization

Default initialization methods fail for hypernetworks. However, given that hypernetworks generalize training separate networks for each task, it must be possible to initialize them as if each of the corresponding base networks were initialized independently, from some given initialization function, f, known to train reliably. Using this insight, we propose and evaluate two initialization schemes for the hypernetwork. We propose one where (under some assumptions) each base network is independently initialized from f. We also propose one where (under no assumptions) all base networks share an initialization sampled from f.

Our first method is *Weight-HyperInit*. Let W and b be the weights and bias in the last layer of our hypernetwork, $h(e)$, respectively. (These parameters are both contained in θ.) We define this weight-only initialization as follows:

$$W_{:,i} := \phi^i \sim f(\phi) \ \forall i, \qquad b := \mathbf{0},$$

where f is an arbitrary initialization scheme for the base network specifying a distribution over a vector of parameters, ϕ, and $W_{:,i}$ is the i-th column of W.

Weight-HyperInit reproduces any given initialization for each base network (π_ϕ), under the assumptions that $e = \mathbb{1}_i$ and the hypernetwork is linear. In this case, each column

of W is simply a sample from the base scheme, one of which is selected for each task via the one-hot encoding. For example, given the task embedding $e = \mathbb{1}_3$, the following base network initialization ϕ_{init} is produced:

$$\phi_{\text{init}} = We + b = \begin{pmatrix} \phi_1^1 & \phi_1^2 & \phi_1^3 & \dots \\ \phi_2^1 & \phi_2^2 & \phi_2^3 & \dots \\ \phi_3^1 & \phi_3^2 & \phi_3^3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix} + \mathbf{0} = \phi^3. \tag{4.1}$$

Moreover, in the case that there is also no bias, it is also equivalent to training separate networks for each task. Although these assumptions are not met for meta-RL, and so do not hold for our experiments, we find this is still an improvement over default neural network initialization schemes.

Additionally, we propose *Bias-HyperInit*. This bias-only initialization is defined as follows:

$$W_{i,j} := 0 \ \forall i, j, \qquad b := \phi_{\text{shared}} \sim f(\phi).$$

Bias-HyperInit achieves an arbitrary initialization for the base network without any assumptions, by setting the parameters for any task to be the same at initialization. This encourages parameter sharing between base networks at the beginning of training, where possible, but also allows for separate base network parameters to be learned, if necessary. Under any set of assumptions, the base network is initialised to the following:

$$\phi_{\text{init}} = Wx + b = \begin{pmatrix} 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \end{pmatrix} + \phi_{\text{shared}} = \phi_{\text{shared}}, \tag{4.2}$$

where x is the final hidden layer of the hypernetwork. ($X = e$ in the case of a linear hypernetwork.)

Both methods initialize only W and b, which define the head of the network. All other layers in h may be initialized by any default initialization scheme. All such choices are detailed in supplementary materials.

4.4.3 Baselines

VariBAD. VariBAD [Zintgraf et al., 2020, 2021b] is a task-inference method making use of variational inference to avoid the requirement of privileged information. See Section 4.3.1 and Chapter 2 for details.

RL2. RL2 [Duan et al., 2016, Wang et al., 2016] is a black-box method that structures that inner-loop as a recurrent neural network to arbitrarily condition on history. See Section 4.3.1 and Chapter 2 for details.

FiLM. FiLM [Perez et al., 2018] is a convenient baseline situated between hypernetworks and the standard architecture. In FiLM, the hypernetwork generates biases for each layer, but only point-wise scales the activations instead of generating weight matrices. In this case, the base network has its own weights. Bias-HyperInit can easily be adapted to FiLM; details presented in supplementary materials.

HFI. HFI [Chang et al., 2020] is a state-of-the-art initialization method developed in the supervised learning setting. While we do compare to HFI, our methods are simpler and more general. Specifically, our methods work with arbitrary base network target initialization (as opposed to being tied to Kaiming) and our methods work with arbitrary base network architectures (without additional variance analysis). Moreover, our approach is straightforward to apply to additional methods, which we show by applying it to FiLM. Finally, although both HFI and Weight-HyperInit do make assumptions about the input to the hypernetwork, our strongest method, Bias-HyperInit, does not.

4.5 Experiments

In this section, we compare our hypernetwork architecture and initialization methods to baselines on 2D navigation [Zintgraf et al., 2021b], MuJoCo [Yu et al., 2020], ML1 [Yu et al., 2020], and ML10 [Yu et al., 2020] benchmarks. MuJoCo is a common meta-RL benchmark [Finn et al., 2017], as are toy 2D navigation tasks [Finn et al., 2017, Zintgraf et al., 2021b, Humplik et al., 2019, Rakelly et al., 2019]. These two benchmarks allow us to demonstrate that hypernetworks with default initialization methods fail to learn, whereas our proposed methods learn reliably. ML1 and ML10 are benchmarks in Meta-World [Yu et al., 2020]. These two benchmarks have greater room for improvement with state-of-the-art methods [Zintgraf et al., 2021b], which allow us to demonstrate improvement over the baseline architectures. Finally, we use two MuJoCo environments to investigate the performance of hypernetworks against standard architectures in terms of the number of parameters in the model overall.

Throughout our evaluation, we use two-tailed t-tests with $p = 0.05$ to determine significance. Details on model tuning and implementation are presented in supplementary materials.

4.5.1 Navigation and MuJoCo

Here we evaluate on the grid-world variant from Zintgraf et al. [2021b] as our 2D navigation task and MuJoCo [Todorov et al., 2012]. Note Grid-World and Cheetah-Dir contain twenty-four and two non-parametric tasks respectively, while the other MuJoCo environments have parametric variation between the tasks.

In Table 4.1 we see that default initializations frequently fail to learn while Bias-HyperInit learns reliably. Specifically, Kaiming and Normc initializations used with hypernetworks achieve far lower returns than all other methods. Orthogonal initialization is more competitive, however it is still significantly outperformed by Bias-HyperInit

Table 4.1: Comparison of return on grid-world and MuJoCo tasks over five random seeds (mean ± standard error). Entries in bold have insignificant difference from the highest-performing result. HyperInit is abbreviated as *HI*.

Method	Grid-World	Cheetah-Dir	Walker	Ant-Dir	Humanoid
Standard	35.5± 0.4	**2104± 87**	**1828± 38**	1167± 16	**1842± 233**
Kaiming	32.1± 1.2	378± 169	331± 37	253± 86	266± 23
Normc	32.2± 0.5	356± 134	357± 60	264±106	249± 18
Orthogonal	34.9± 0.5	1379± 310	1687± 98	1127± 93	1126± 76
HFI	**36.8± 0.2**	**2218± 80**	1618±130	**1370± 9**	**1323± 57**
Weight-HI	**36.1± 0.5**	**2066± 119**	1748± 57	**1346± 7**	1048± 21
Bias-HI	**36.7± 0.2**	**2300± 32**	**1994± 67**	**1328± 23**	**1678± 162**

in every environment.

We also compare hypernetworks with Bias-HyperInit to the standard architecture and Bias-HyperInit to HFI. We see that Bias-HyperInit matches or exceeds the performance of HFI, with a significant improvement in Walker. Hypernetworks with Bias-HyperInit also significantly outperform the standard architecture on grid-world and Ant-Dir. In fact, Bias-HyperInit is not significantly outperformed by any other method. However, the standard architecture, HFI, and Bias-HyperInit all achieve near optimal performance, motivating an evaluation on Meta-World, on which the standard architecture has greater room for improvement [Zintgraf et al., 2021b].

4.5.2 ML1 and ML10

Here we evaluate on the more challenging Meta-World ML1 and ML10 benchmarks [Yu et al., 2020]. ML1 and ML10 have one and ten non-parametric training tasks respectively (e.g. pushing a ball or opening a window). ML10 additionally has five distinct test tasks. Within each task, there exists parametric variation, such variation in the goal location. Note that we only test on the Pick-Place task from ML1, since VariBAD already achieves a 100% success rate on all other tasks [Zintgraf et al., 2021b].

In Table 4.2 we see significant improvement from hypernetworks with Bias-HyperInit over the standard architecture, as well as the efficacy of Bias-HyperInit on FiLM. On Pick-Place, Bias-HyperInit outperforms the standard architecture with a 9-fold increase in test success percentage. Additionally, Bias-HyperInit improves the FiLM architecture and exceeds the performance of HFI. On ML10, hypernetworks with both Bias-HyperInit and HFI yield a 2-fold increase in test success percentage compared to the standard architecture. Finally, we evaluated Bias-HyperInit when applied to RL2 on ML10, finding a 2-fold increase over both HFI and the standard architecture. Taken together, these results demonstrate a clear improvement from the application of hypernetworks with Bias-HyperInit, regardless of the baseline method that they are applied to.

Table 4.2: Comparison of meta-test success percentage on the Pick-Place ML1 task (ten seeds) and ML10 (three seeds). On more challenging domains, Bias-HyperInit significantly outperforms the standard architecture, with up to a 9-fold increase in test success percentage.

Method		**Pick-Place**	**ML10**	
		VariBAD	**VariBAD**	**RL2**
Standard		4.4 ± 2.4[1]	10.2 ± 3.0	7.2 ± 5.0
FiLM	Normc	5.5 ± 4.8	—	—
	Bias-HyperInit	$\mathbf{34.2 \pm 15.9}$	—	—
Hypernetwork	HFI	$\mathbf{25.5 \pm 14.5}$	28.4 ± 6.0	7.1 ± 2.4
	Bias-HyperInit	$\mathbf{42.9 \pm 16.3}$	23.9 ± 6.2	14.2 ± 7.2

4.5.3 Scaling

Because hypernetworks learn a mapping from a task embedding to the parameters of a base network, they require significantly more parameters in the entire model than a standard architecture with the equivalent base network. For a fair comparison, we evaluate return over both a range of base network sizes and total number of

[1] Zintgraf et al. [2021b] report a success percentage of 29% for Pick-Place; however, we were not able to replicate this result.

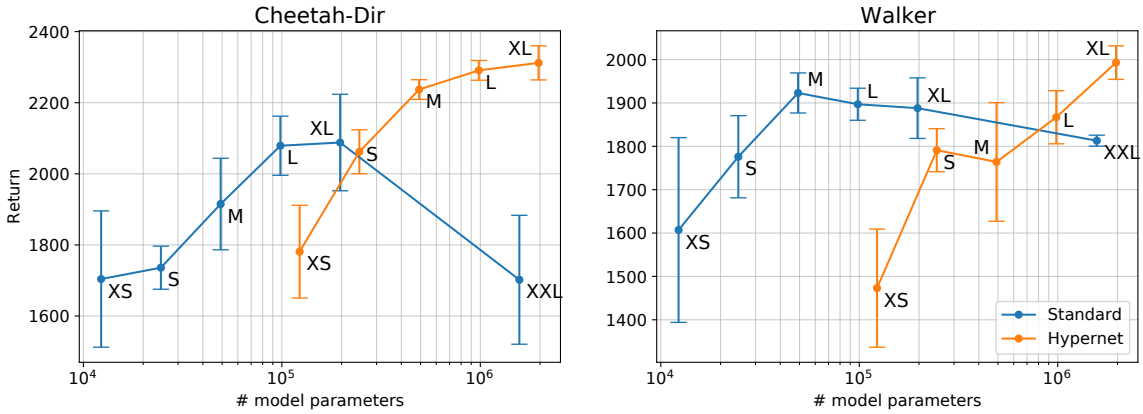


Figure 4.3: Performance of standard and hypernetwork models over a range of base policy architecture sizes on Cheetah-Dir and Walker. Architectures are presented in supplementary materials.

parameters in the model on the Cheetah-Dir and Walker tasks (Figure 4.3). We find that hypernetworks consistently match or outperform the standard architectures with the equivalent base network size. We also find that hypernetworks likewise outperform standard architectures for equivalent number of parameters in the entire model, i.e., for a given value on the x-axis, when the total number of parameters in the model is sufficiently large. Just as the performance of the standard architecture deteriorates when the number of parameters becomes excessively large, we anticipate that the performance of hypernetworks would likewise degrade beyond some threshold. While we were unable to validate this hypothesis due to computational limitations, we expect the performance curve to exhibit a similar shape, but shifted to the right along the x-axis. However, even if performance does eventually decline, the maximum achieved by hypernetworks already exceeds that of the standard architectures. Because the hypernetwork curve is both vertically higher and shifted to the right, improved returns can be achieved simply by increasing the total number of model parameters.

4.6 Conclusion

While we significantly improve the performance of meta-RL agents, we first consider some limitations before summarizing our key contributions. While our proposed methods are general, we cannot guarantee an improvement for all meta-RL methods. To mitigate this limitation, we build on top of VariBAD, which is state of the art, and additionally evaluate our method applied to RL2 on ML10. Furthermore, as in any empirical study, there is no guarantee that our results hold on real robots or other meta-RL benchmarks. However, we have tested on seven standard meta-RL environments in total, including Meta-World, which was proposed specifically for addressing robotic agents in the real world. As much as is possible from simulated meta-RL experiments, these results give us confidence in a significant improvement over previously established methods.

In summary, we used hypernetworks to improve a state-of-the-art method in meta-RL, evaluating over a range of benchmarks. In doing so, we demonstrated that hypernetworks are a promising path forward for meta-RL. Moreover, we showed that the initialization of the hypernetwork is crucial, as default initialization methods fail. To overcome this difficulty, we presented two novel initialization methods: Bias-HyperInit and Weight-HyperInit. Bias-HyperInit matched or exceeded the performance of existing methods from the supervised learning setting, while also being simpler and more general – applying to arbitrary base network initializations, base network architectures, and also improving FiLM. Using Bias-HyperInit, we showed that hypernetwork performance improves substantially over standard architectures. Finally, we demonstrated that hypernetworks outperform the standard architecture for equivalently sized base policies, and outperform it at any size given sufficiently many parameters in the entire model. Thus, hypernetworks not only outperform the standard architecture, but also scale preferentially in terms of the overall number of

parameters.

While the work in this chapter significantly improves the performance of meta-RL agents, it also enables future research on the application of hypernetworks in the field. For example, this work opens the path both for future research extending meta-SL methods using hypernetworks [Rusu et al., 2019, Ravi and Larochelle, 2017] and for applying multi-task RL methods with separate parameters for each task [Teh et al., 2017, Parisotto et al., 2016, Misra et al., 2016] to meta-RL.

4.7 Statement of Authorship

Chapter 4 is based on content from **Hypernetworks in Meta-Reinforcement Learning** [Beck et al., 2022].

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the lead author on this manuscript. I was primarily responsible for the ideation, implementation, experimentation, and writing. Co-authors helped in these areas, particularly with experimentation and editing.

Citation:

Jacob Beck, Matthew Jackson, Risto Vuorio, and Shimon Whiteson. "Hypernetworks in Meta-Reinforcement Learning." *6th Conference on Robot Learning*, 2022.

5

Supervision in Hypernetworks

Contents

5.1 Introduction

In Chapter 4, we investigated the use of initialization methods for hypernetworks in meta-RL, primarily in conjunction with *task-inference* methods. While we also

briefly investigated *black-box* methods, and saw that hypernetworks may improve performance for such methods, the use of black-box methods was not investigated extensively.

In this chapter, we compare task-inference methods and black-box methods when used with hypernetworks in meta-RL. As discussed in Chapter 3, any sequential model such as a recurrent neural network (RNN), can be deployed to learn this mapping end-to-end [Duan et al., 2016, Wang et al., 2016]. Such methods are also called *black-box*, and the recurrent version is a canonical example of a black box method. In contrast, much prior work as focused on task-inference methods that are specialized for meta-RL. A meta-RL algorithm learns to reinforcement learn over a distribution of MDPs, or tasks. By explicitly learning to infer the task, many methods have shown improved performance relative to the recurrent baseline [Humplik et al., 2019, Zintgraf et al., 2020, Kamienny et al., 2020, Liu et al., 2021, Beck et al., 2022].

Recent work has shown the simpler recurrent methods to be a competitive baseline relative to task-inference methods [Ni et al., 2022]. However, such claims are contentious, as the supporting experiments compare only to one task-inference method designed for meta-RL, the experiments provide additional compute to the recurrent baseline, and the results still show similar or inferior performance to more complicated methods on the majority of difficult domains. In particular, they consider two toy domains and four challenging domains, with RNNs significantly outperformed on two of the four challenging domains, and superior to the single task-inference baseline on only one.

In this chapter, we conduct a far more extensive empirical investigation with stronger and carefully designed baselines in meta-RL specifically. In addition, we afford equal computation in terms of number of samples for hyper-parameter tuning to all existing baselines. We present the key insight that the use of a hypernetwork architecture [Ha et al., 2017] is crucial to maximizing the potential of recurrent networks. For

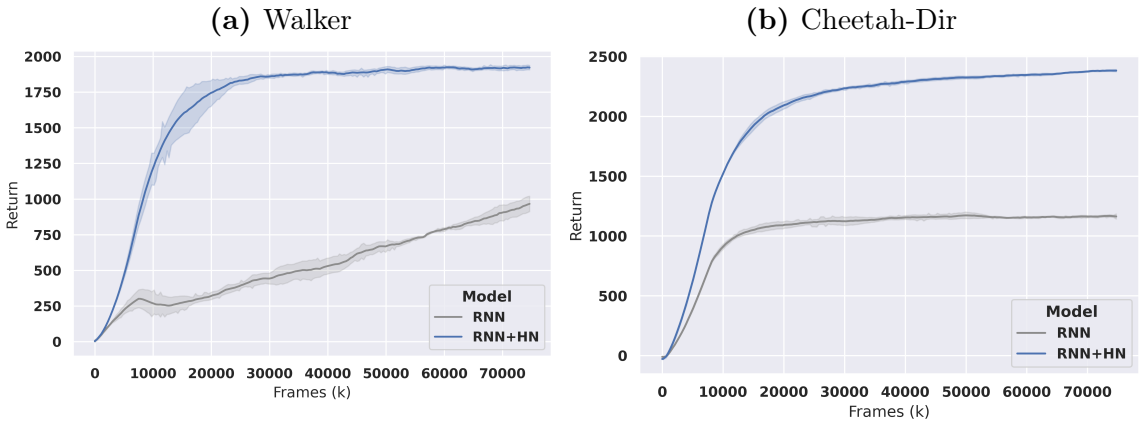


Figure 5.1: In some environments, recurrent neural networks fail to learn meta-RL tasks, whereas recurrent hypernetworks achieve strong performance.

an illustration of the potential magnitude of improvement, see Figure 5.1. Notably, recurrent hypernetworks have never been evaluated in meta-RL, let alone shown to outperform contemporary task-inference methods. We additionally provide preliminary evidence that the robust performance hypernetworks achieve such is in part due to how they condition on the current state and history. Finally, our results establish recurrent hypernetworks as an exceedingly strong method on meta-RL benchmarks that is also far simpler than alternatives, providing significant ease of use for practitioners in meta-RL.

5.2 Related Work

Recurrent Meta-RL. Many meta-RL methods structure the learned RL algorithm as a black box using a neural network as a general purpose sequence model [Duan et al., 2016, Wang et al., 2016, Mishra et al., 2018, Fortunato et al., 2019, Ritter et al., 2021, Wang et al., 2021, Ni et al., 2022]. While any sequence model can be used, often the model is structured as an RNN [Duan et al., 2016, Wang et al., 2016, Ni et al., 2022]. Such models [Duan et al., 2016, Wang et al., 2016] are commonly used as simple meta-RL baselines.

One study has shown RNNs to be a competitive baseline in meta-RL [Ni et al., 2022]; however, the scope of the study was broader than meta-RL and the evidence specific to meta-RL is inconclusive. First, the study evaluates only a single specialized meta-RL method [Zintgraf et al., 2020], which was, but is not currently, state-of-the-art [Beck et al., 2022]. Second, the experiments use results or hyperparameters from the original papers, while affording extra computation to tune the RNNs on each benchmark, including dimensions that were not tuned for the other baselines. This computation includes tuning architecture choices, the context length, the RL algorithm used, and the inputs [Ni et al., 2022]. And third, the study does not show particularly strong performance of recurrent methods relative to the chosen specialized baseline. On the MuJoCo domains, the recurrent baseline outperforms the specialized method on only one of these four domains, performs similarly on another, and is significantly outperformed on the other remaining two [Ni et al., 2022]. In contrast, our work compares against four specialized baselines; affords equal computation to all methods, defaulting to hyper-parameters that favor existing task-inference methods for parameters that are not tuned; and still establishes recurrent hypernetworks as the strongest method evaluated.

Task Inference Meta-RL. In addition to recurrent meta-RL methods, task-inference methods [Humplik et al., 2019, Zintgraf et al., 2020, Kamienny et al., 2020, Liu et al., 2021, Beck et al., 2022] and policy-gradient methods [Yoon et al., 2018, Finn et al., 2017, Vuorio et al., 2019, Zintgraf et al., 2019] constitute a significant bulk of existing work. We exclude the latter methods from comparison since the estimation of a policy gradient in policy-gradient approaches requires more data than in our benchmarks [Zintgraf et al., 2019, Beck et al., 2023a]. Task inference methods are a strong baseline for our benchmark, but are generally more complicated than recurrent meta-RL methods. For example, such methods typically add a task inference

objective [Humplik et al., 2019], and may also add a variational inference component [Zintgraf et al., 2021b], or pre-training of embeddings with privileged information [Liu et al., 2021]. In this chapter, we ablate each of these components to create the strongest task-inference baselines possible. In the end, we find the more complicated task inference methods are still inferior to the recurrent baseline with hypernetworks.

Hypernetworks. A hypernetwork [Ha et al., 2017] is a neural networks that produces the parameters (weights and biases) for another neural network, called the base network. Hypernetworks have been used in supervised learning (SL) [Ha et al., 2017, Chang et al., 2020], Meta-SL [Rusu et al., 2019, Munkhdalai and Yu, 2017, Przewiezlikowski et al., 2022], multi-task RL [Xiong et al., 2023, 2024], and meta-RL [Beck et al., 2022, Xian et al., 2021, Peng et al., 2021, Sarafian et al., 2021]. While these networks are complicated, and can fail to work out-of-the-box, simple initialization methods can be sufficient to enable stable learning, as we have seen [Beck et al., 2022, Chang et al., 2020]. In meta-RL, no prior work to this thesis has investigated training a hypernetwork end-to-end to arbitrarily modify the weights of a policy. However, this appears to be an oversight, since the results in Chapter 4 suggest that hypernetworks are particularly useful in preventing interference between different tasks and enable greater returns as the the number of parameters increases. Until this thesis, recurrent hypernetworks had never been evaluated in meta-RL, and until this chapter, recurrent hypernetworks had not been shown to outperform contemporary task-inference methods.

5.3 Methods

5.3.1 Recurrent Methods

Recurrent methods are perhaps the simplest and most common meta-RL baseline. Recurrent methods use an RNN to encode history and train all meta-parameters end-to-end on Equation 2.3. These methods are depicted in Figure 5.2. Note that the

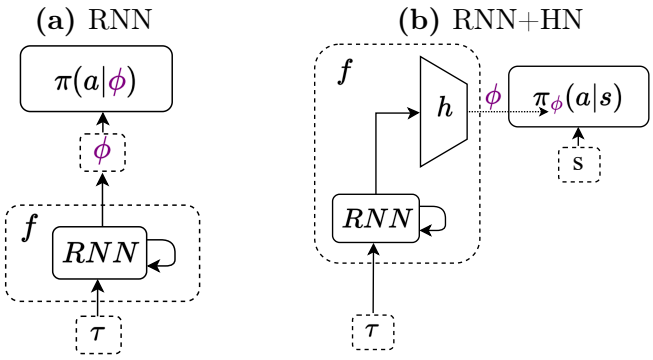


Figure 5.2: The standard RNN policy (a) and the RNN policy with a hypernetwork (b).

combination of a hypernetwork with recurrent networks (RNN+HN below) has never been evaluated in meta-RL prior to this thesis, but we will show that the combination actually achieves the strongest results.

RNN. Our first recurrent baseline is the simplest and is equivalent to RL2 [Duan et al., 2016] and L2RL [Wang et al., 2016]. In this case $\pi_\theta(a|\phi = f_\theta(\tau))$, where f is a recurrent network, π is a feed-forward network, and f and π each use distinct subsets of the meta-parameters, θ.

RNN+HN. Our second recurrent model is the recurrent hypernet and achieves the strongest results. Here, the recurrent network produces the weights and biases for the policy directly: $\pi_\phi(a|s)$. The state must be passed as input again to this policy for the feed-forward policy to condition on an input, and we follow the initialization method for hypernetworks, Bias-HyperInit, proposed in Chapter 4. In this initialization, the hypernetwork's final linear layer is initialized with a zero weight matrix and a non-zero bias, so that the hypernetwork produces the same base-network parameters for any trajectory at the start of training.

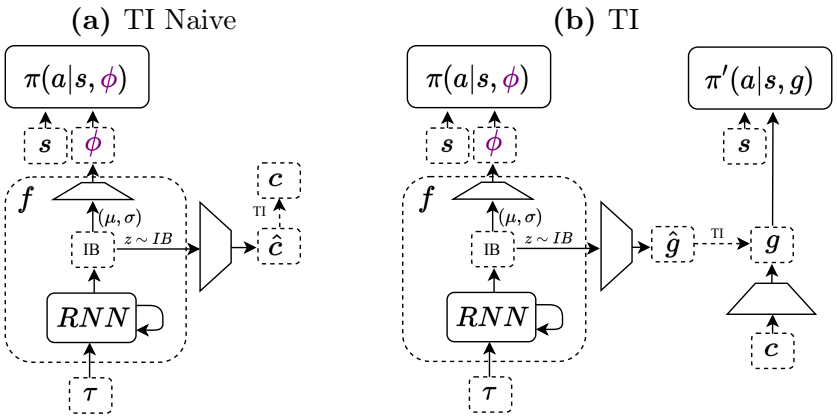


Figure 5.3: Task-inference baselines. Naive task-inference (a), additional multi-task pre-training (b).

5.3.2 Task-Inference Methods

Task-inference methods constitute the main category of meta-RL methods capable of adaptation as quickly as recurrent (i.e., black-box) methods [Humplik et al., 2019, Zintgraf et al., 2020, Kamienny et al., 2020, Liu et al., 2021, Beck et al., 2022]. These methods train the inner-loop not end-to-end but rather to identify the task, within the given task distribution. One perspective on these methods is that they attempt to shift the problem from the more difficult meta-RL setting to the easier multi-task setting by learning to explicitly infer the task. Here we define relevant task-inference methods used as baselines (Figures 5.3 and 5.4). For additional ablations, summary, and details on the method selection process, see the appendix. Results in the appendix include combinations of end-to-end supervision with task-inference supervision and task-inference methods that use experts trained in the multi-task setting as explicit targets for inference, similar to [Peng et al., 2021].

TI Naive. The inner-loop of a meta-RL method must take in the trajectory, τ, and produce the policy parameters, ϕ. In task inference methods, the inner-loop additionally produces an estimate of the current task, $\hat{c}_{\mathcal{M}}$, given a known task representation, $c_{\mathcal{M}}$. While it is possible to represent ϕ as $\hat{c}_{\mathcal{M}}$ directly, i.e., pass $\hat{c}_{\mathcal{M}}$ to the policy,

it is common to compute ϕ from an intermediate layer of the network that predicts $\hat{c}_{\mathcal{M}}$, which contains more information about the belief state [Humplik et al., 2019, Zintgraf et al., 2020]. It is also common to use an information bottleneck to remove information not useful for task inference from the trajectory [Humplik et al., 2019, Zintgraf et al., 2020]. Following Zintgraf et al. [2020], we condition ϕ on the mean and variance of the bottleneck layer in order to explicitly condition the policy on task uncertainty for more efficient exploration (Figure 5.3). Putting these together, we can write a task inference method as follows:

$$\begin{aligned}
\mu &= P^{\mu}(RNN(\tau)) \\
\sigma &= P^{\sigma}(RNN(\tau)) \\
IB &= \mathcal{N}(z; \mu, \sigma) \\
\hat{c}_{\mathcal{M}} &= P^{c}(z \sim IB) \\
\phi &= ReLU(P^{\phi}\perp(\mu, \sigma)) \\
J_{infer}(\theta) &= \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\tau|\pi}[-||c_{\mathcal{M}} - \hat{c}_{\mathcal{M}}||_2^2]] \\
J_{prior}(\theta) &= \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\tau|\pi}[D(IB||\mathcal{N}(z; \mu = 0, \sigma = I))]],
\end{aligned}$$

where P^c, P^{ϕ}, P^{μ}, and P^{σ} are all linear projections, $\perp$ represents a stop-gradient, $P^{\phi}\perp(\mu, \sigma)$ is the matrix multiplication of P^{ϕ} with stop-gradient of a concatenation of (μ, σ), and D is the KL-divergence. Here, $J_{infer} + J_{prior}$ constitutes the evidence lower bound from Zintgraf et al. [2020] and is used to train IB, whereas all other parameters (P^{ϕ} and $\pi(\cdot|\phi)$) are trained via Equation 2.3.

TI. As presented, the TI Naive baseline may suffer from a known issue where the given task representation contains too little or too much information. When too little information is present, the policy may miss information crucial to the task. When too much information is present, the policy may be presented with the difficult problem of separating the useful information from irrelevant task features. Toward this end, it

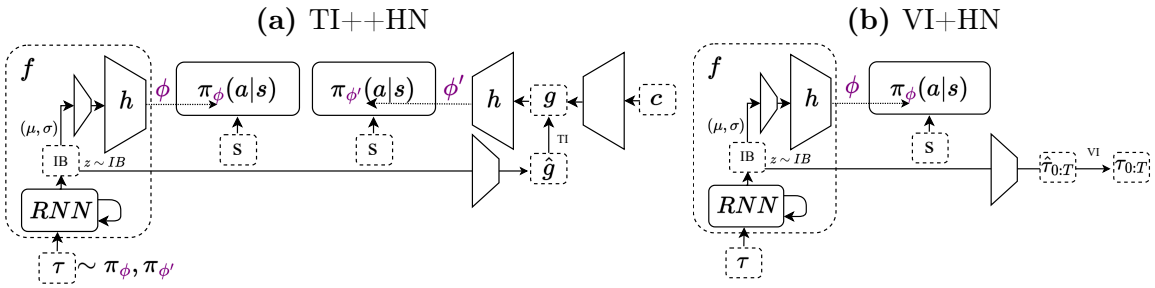


Figure 5.4: Task-inference baselines. Task inference with additional parameter reuse and hypernetwork (a), the competitive task-inference algorithm with hypernetworks from Chapter 4 (b).

is possible to pre-train the task representation end-to-end using an additional policy [Humplik et al., 2019, Kamienny et al., 2020, Liu et al., 2021]. Our TI baseline (Figure 5.3) is the same as TI Naive, except that an additional multi-task policy, π', is pre-trained to learn a representation of the task, $g_\theta(c_\mathcal{M})$. Given a linear projection, P^g:

$$\hat{g} = P^g(z \sim IB)$$

$$J_{multi}(\theta) = \mathbb{E}_{\mathcal{M}\sim p(\mathcal{M})}[\mathbb{E}_\tau[R(\tau)|\pi'_\theta(\cdot|g_\theta(c_\mathcal{M})), \mathcal{M}]$$

$$J_{infer}(\theta) = \mathbb{E}_\mathcal{M}[\mathbb{E}_{\tau|\pi(\cdot|\phi)}[-||g_\theta(c_\mathcal{M}) - \hat{g}||_2^2]].$$

For a fair comparison, training of the multi-task policy, π', occurs at the expense of training the meta-learned policy π, with the total number of samples remaining constant. Instead of fully training the multi-task policy, we experiment with different amounts of pre-training in the appendix, finding significant benefits already from less than 5% of total training allocation for the pre-training.

TI++HN. The TI++HN baseline is the same as TI, with three additions that we found to strengthen task inference (Figure 5.4). The first two additions (++) are novel and are 1) initializing the parameters of the meta-policy, π, to that of the pre-trained multi-task policy, π', to encourage transfer and 2) training of the task-inference

(J_{infer}) over trajectories from the initial multi-task training phase in addition to the meta-learning phase, since the former tend to be more informative and simply provide extra data. The third addition (HN) uses a hypernetwork to condition the policy on ϕ. We write this as $\pi_\phi(\cdot|s)$ to show that ϕ represents the weights and biases of π, just as in the recurrent baseline. The output of the hypernetwork is ϕ, and the input to the hypernetwork is a the projection of μ and σ, $\phi = h(ReLU(P^\phi \perp(\mu, \sigma)))$. When using a hypernetwork with the first two additions (++), the parameters of the hypernetwork for the meta-learned policy are initialized to the parameters of hypernetwork for the multi-task policy, instead of sharing policy parameters directly.

VI+HN. While task-inference methods rely on known task representations, it is also possible to design methods that can infer the MDP more directly. This can be done by inferring transitions and rewards in full trajectories, since the transition function and reward function collectively define the MDP. In particular, such a method, called VariBAD, is proposed by Zintgraf et al. [2020], and extended with the use of hypernetworks, as in Chapter 4. Here, we call this method VI+HN, and it is a contemporary task-inference method (Figure 5.4). Precisely, this model reconstructs full trajectories including future transitions for meta-episodes, $\tau_{0:T}$, instead of task embeddings, from τ, the current trajectory:

$$J_{infer}(\theta) = \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\tau|\pi(\cdot|\phi)}[-||\tau_{0:T} - \hat{\tau}_{0:T}||_2^2]].$$

See Appendix B.1 for ablations with VI alone.

5.4 Experiments

In this section we compare recurrent hypernetworks (RNN+HN) to task inference baselines. We evaluate over three simple navigation domains [Zintgraf et al., 2020, Humplik et al., 2019, Rakelly et al., 2019], designed to test learning of exploration and memory, in addition to four more difficult tasks using MuJoCo [Todorov et al., 2012],

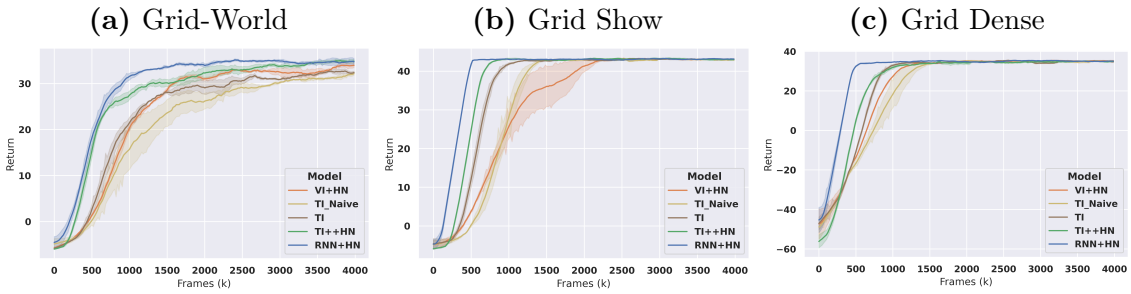


Figure 5.5: Evaluation on grid-world benchmarks. RNN+HN and TI++HN improve return. RNN+HN achieves the greatest asymptotic return and sample efficiency.

and one task testing long-term memory from visual observations in MineCraft [Beck et al., 2020]. Results show meta-episode return, optimized over five learning rates and averaged over three seeds (four in MineCraft), with a 68% confidence interval using bootstrapping. Additional details and results on Meta-World [Yu et al., 2020] are available in the appendix. Our experiments demonstrate that while our task inference methods are strong baselines, RNN+HN is able to outperform them and achieve the highest returns.

5.4.1 Grid-Worlds

Navigation tasks are a common benchmark in meta-RL [Zintgraf et al., 2020, Humplik et al., 2019, Rakelly et al., 2019]. Here we evaluate on the grid-world variant from Zintgraf et al. [2020], in addition to two of our own variants. The first environment, Grid-World, consists of a five by five grid with a goal location in one of the grid cells. The agent starts in the bottom left corner of the grid, and then must navigate to a goal location, which is held constant throughout the meta-episode. (Details are in the appendix.) This environment is useful for testing how well a meta-learning algorithm learns to efficiently explore in the gridworld as it searches for the goal. Additionally, our Grid-World Show environment was designed to be relatively harder for end-to-end methods, in order to provide a challenge for our proposed method. In this environment, the goal position is visible to the agent at the first timestep

of each episode. Task inference methods will directly encourage the storage of this information in memory, whereas the end-to-end recurrent methods must learn to store this information through its effect on the policy. In contrast, our Grid-World Dense environment provides dense rewards and may be easier for end-to-end methods. In this environment, the agent receives and observes a reward equal to the Manhattan distance to the goal location. Instead of inferring the task explicitly, the agent can simply move up or to the right until the reward stops increasing.

Surprisingly, on all three grid-worlds, RNN+HN achieves both the greatest asymptotic return and greatest sample efficiency (Figure 5.5). The recurrent hypernetwork achieves the fastest learning on Gridworld Show, despite the environment being specifically designed to be harder for end-to-end methods. TI++HN dominates all other task-inference baselines on these grid-worlds, suggesting that it is a relatively strong task-inference method. Collectively, these grid-worlds demonstrate that end-to-end learning with hypernetworks can learn to store the task in memory and to explore optimally with this information directly from return, just as well as task-inference methods.

5.4.2 MuJoCo

Here we evaluate baselines on more challenging domains. We evaluate on all four MuJoCo variants proposed by Zintgraf et al. [2020], which is known to be a common and more challenging meta-RL benchmark involving distributions of tasks requiring legged locomotion [Zintgraf et al., 2020, Humplik et al., 2019, Rakelly et al., 2019, Beck et al., 2023a]. Ant-Dir and Cheetah-Dir both involve non-parametric task variation, whereas Cheetah-Vel and Walker include parametric variation of the target velocity and physics coefficients respectively. For environment details, see the appendix. We expect Walker in particular to be difficult for end-to-end methods since it has the largest space of tasks with the dynamics defined by 65 different parameters. Assuming

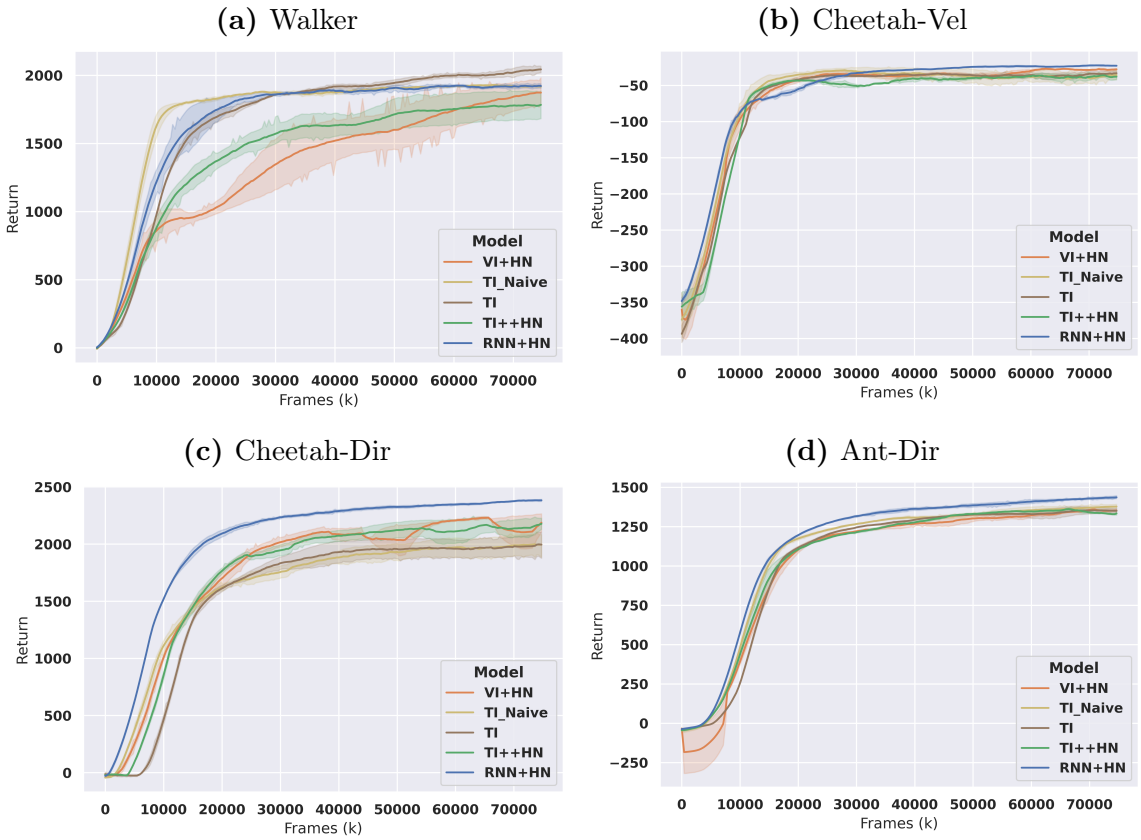


Figure 5.6: Models evaluated on MuJoCo benchmark. RNN+HN matches returns on Walker and Cheetah-Vel, and exceeds returns on Cheetah-Dir and Ant-Dir.

the values of these parameters are important for the optimal policy, task inference methods may learn the optimal policy faster.

On Cheetah-Dir and Ant-Dir, RNN+HN achieves greater returns than all other baselines by a wide margin (Figure 5.6). On Cheetah-Vel, all methods achieve fairly similar results, with RNN+HN still achieving the greatest asymptotic return by a small margin. As expected, RNN+HN does not outperform task inference on Walker. For Walker, only TI outperforms RNN+HN in terms of efficiency, and only TI Naive outperforms RNN+HN in terms of asymptotic return; however, the effect size is small and both TI and TI Naive have among the worst performance on Cheetah-Dir and the grid-worlds. Still, RNN+HN achieves similar performance on Walker, which is notable in a high dimensional task space. And, RNN+HN achieves greater returns overall.

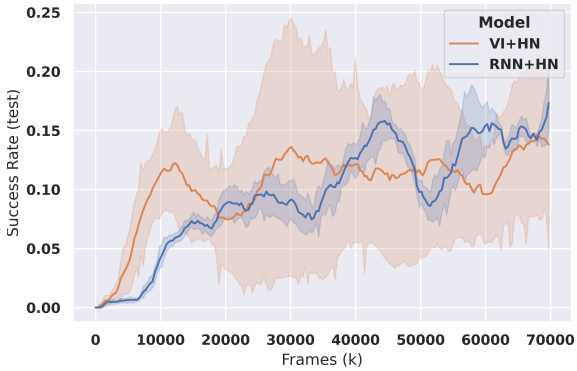


Figure 5.7: Results on ML10 show that RNN+HN is able to match the performance of the more complicated VI+HN method, though results are not definitive on this environment.

5.4.3 Meta-World

We additionally evaluate on the Meta-World (ML10) [Yu et al., 2020] environment and show RNN+HN is able to match the performance of the more complicated VI+HN method (Figure 5.7). Meta-World is a meta-RL benchmark for robotic manipulation. There are 10 non-parametric training tasks and 5 distinct non-parametric test tasks, ranging from pushing a ball to opening a window. Additionally, there is non-parametric variation, e.g. of the goal location, within each task. In this environment, the variance of returns and the computation requirements for training are dramatically greater, so the evidence is not strong is any direction. However, we include the results here since ML10 is a standard benchmark, and to elaborate on the results from Chapter 4, since those results were not statistically significant.

Here, we find that the final return of RNN+HN is higher than VI+HN. RNN+HN has a final average test success percent of 17.22, compared to 13.87 for VI+HN. However, the variance is still too large to draw firm conclusions. Additionally, we find that the learning curves for RNN+HN and VI+HN overlap consistently throughout training, and the confidence interval for RNN+HN lies almost entirely within that of VI+HN.

While these results are not sufficient to conclusively show superiority on ML10, we do

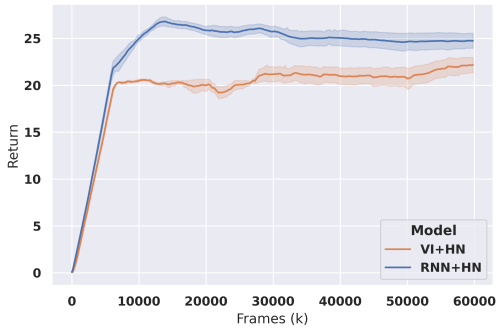


Figure 5.8: RNN+HN outperforms VI+HN on MC-LS (MineCraft) environment.

show superiority on many other environments. Additionally, these results do show RNN+HN to match the performance of VI+HN. Matching existing task-inference baselines on ML10 is significant, since it demonstrates that complicated task-inference methods are not necessary to be competitive.

5.4.4 MineCraft

We additionally evaluate on the MC-LS environment from Beck et al. [2020], designed to test long-term memory from visual observations in MineCraft. Here, the agent navigates through a series of 16 rooms. In each room, the agent navigates left or right around a column, depending on whether the column is made of diamond or iron. Discrete actions allow for a finite set of observations. Correct behavior can be computed from the observation and receives a reward of 0.1. At the end, the agent moves right or left depending on a signal (red or green) that defines the task and is shown before the first room. Correct and incorrect behavior receives a reward of 4 and -3, respectively. We allow the agent to adapt over two consecutive episodes, forming a single meta-episode.

On MC-LS we compare RNN+HN to VI+HN alone, given a limited compute budget and since VI+HN is a a competitive task-inference baseline, as seen in Chapter 4. Additionally, we add an extra seed (four in total) and a linear learning rate decay due to high variance in the environment. In Figure 5.8, we see that RNN+HN significantly

outperforms VI+HN. While VI+HN learns to navigate through all rooms, it does not reliably learn the correct long-term memory behavior. In contrast, RNN+HN is able to adapt reliably within two episodes, and one seed even learns to adapt reliably within a single episode. While further work is needed to learn the optimal policy, these experiments demonstrate that RNN+HN outperforms VI+HN, even on more challenging domains.

5.5 Discussion

Here we investigate why the recurrent hypernetworks have such robust performance on meta-RL benchmarks. First, we observe that the state processing differs between the RNN and RNN+HN baselines. In particular, RNN conditions on the current state only through its dependence on ϕ, whereas hypernetworks pass in the state again to the policy. Thus, we investigate whether the difference in inputs alone could be the cause of the improvement in performance. To this end, we introduce a new ablation to test the effect of just passing in the state again. Details are below. Second, we inspect how sensitivity to latent variables encoding the trajectory affects performance.

5.5.1 State Conditioning

Hypernetworks condition on the current state both through ϕ, which contains information about trajectory, including the current state, and by directly conditioning

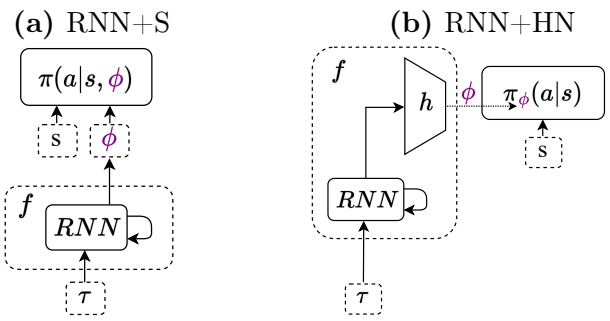


Figure 5.9: The RNN+S policy (a) and the RNN policy with a hypernetwork (b).

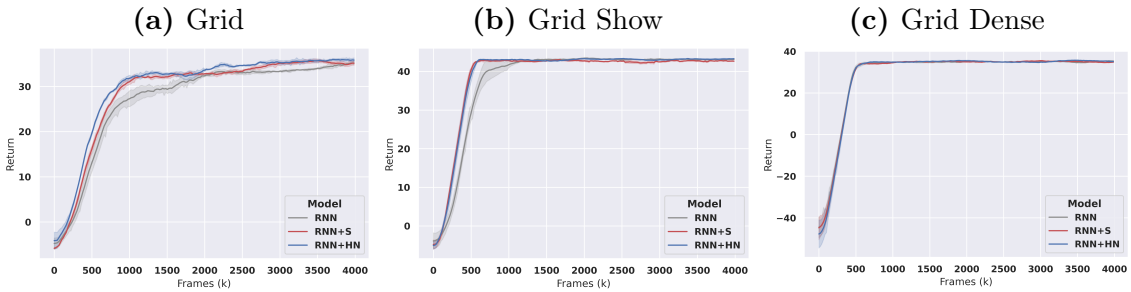


Figure 5.10: RNN+HN matches or exceeds RNN+S and RNN, but RNN+S is also strong on grid-worlds.

on the state. Since the hypernetwork conditions on state twice, we test to see the effect of conditioning on the state twice without hypernetworks. We call this ablation RNN+S, which we write as $\pi_\theta(a|s,\phi)$ (Figure 5.9). In an empirical evaluation, we see that while RNN+S does perform favorably relative to RNN alone, RNN+HN still outperforms RNN+S (Figures 5.10 and 5.11). In particular, RNN+HN achieves similar returns to RNN+S on Ant-Dir and Cheetah-Vel, and outperforms RNN+S on all other environments, in terms of asymptotic return and sample efficiency. Taken together, we see that the advantage of RNN+HN comes both from the ability to re-condition on state directly and from the hypernetwork architecture. These results confirm that to achieve the strongest performance, re-conditioning on state directly is not sufficient, and that the hypernetwork architecture itself is still critical.

5.5.2 Trajectory Conditioning

We also investigate how the hypernetworks condition on the trajectory. In particular, we investigate the sensitivity of the output of the network to an intermediate latent representation of the trajectory. For this purpose, we chose to measure the gradient norm of the first hidden layer of the hypernetwork on the Walker environment. We perform this investigation both with the initialization method designed for hypernetworks that we used throughout our experiments, Bias-HyperInit, and with Kaiming initialization [He et al., 2015], not designed for hypernetworks. We add Kaiming

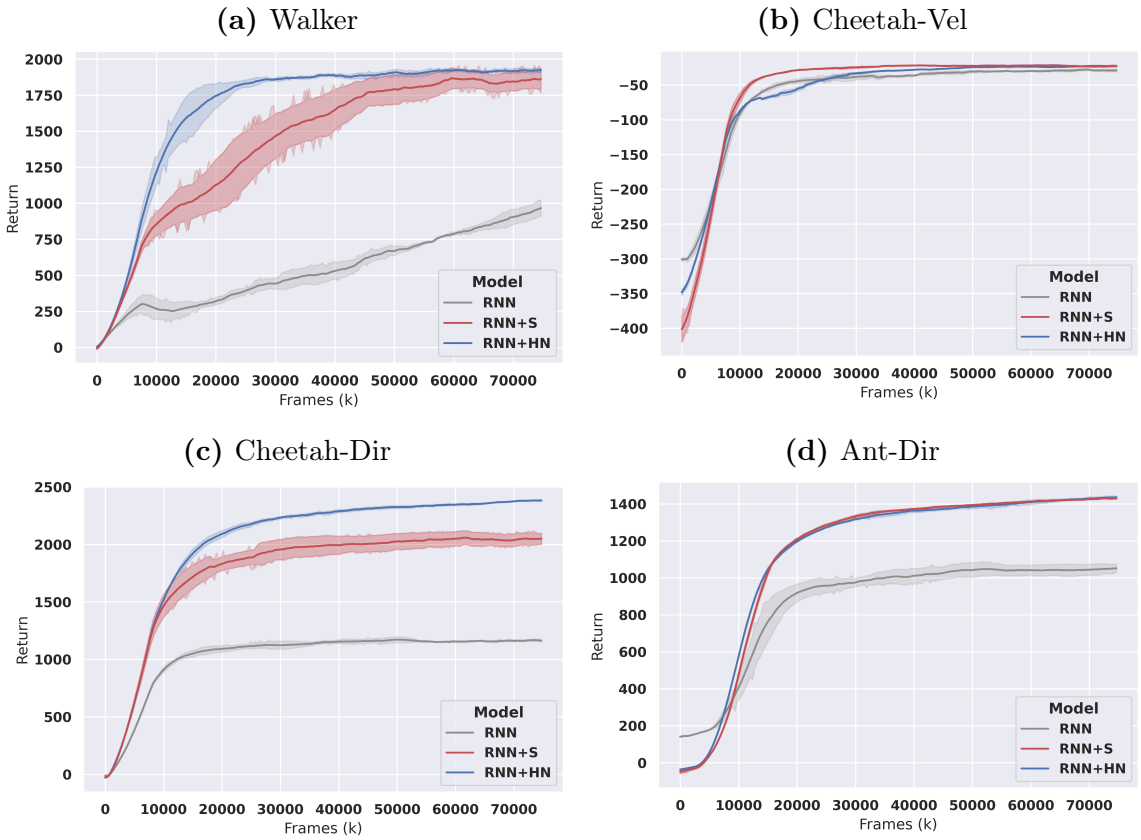


Figure 5.11: RNN+HN matches or exceeds RNN+S on MuJoCo tasks. RNN alone is a weak baseline.

initialization, since Bias-HyperInit ignores trajectories at the start of training. First, we confirm the finding that Bias-HyperInit is crucial for performance (Figure 5.12). Second, we see the two models that performs worst, RNN and RNN+HN Kaiming, also have the greatest norm. Moreover, we find that both RNN+HN and RNN+S start with a low gradient norm and then further decrease this norm throughout training, whereas the RNN model increases this norm. We hypothesize that a low norm, i.e., low sensitivity to the latent variable, is crucial for stable training and that the RNN model increases this norm to remain sensitive to the state, since the state is only encoded in the latent for this model.

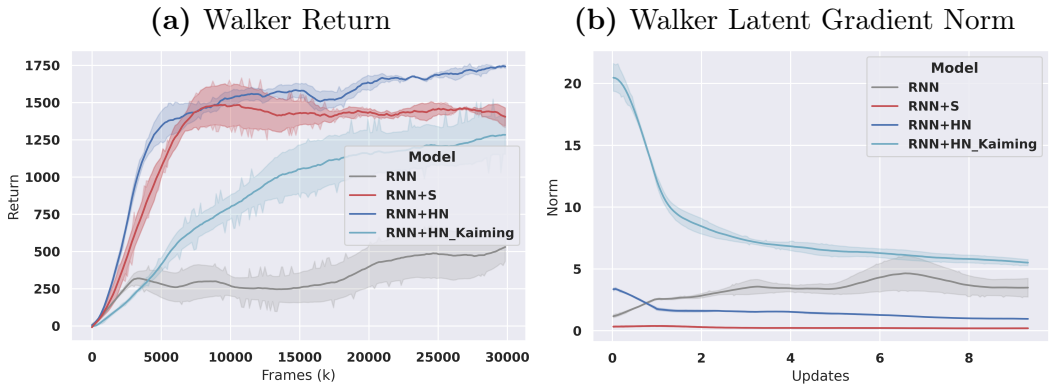


Figure 5.12: Returns (a) and Gradient norms on Walker (b). The RNN method must increase this norm to condition on state, whereas others do not. Lower gradient norms seem important to performance.

5.6 Conclusion

Here, we consider limitations and then conclude the chapter. As an empirical study of meta-RL, we cannot guarantee that recurrent hypernetworks will improve over every baseline nor on every environment. However, we mitigate this issue by comparing to many baselines and performing many ablations. In particular, we compare to a contemporary task-inference method (VI+HN), design our own baseline which we show to be stronger than others on all grid-worlds (TI++HN), and also include standard methods (TI and TI Naive), in addition to further ablations in the appendix. In as much as an empirical study can, we believe our study demonstrates a significant improvement of the RNN+HN method over existing baselines.

In this chapter, we establish recurrent hypernetworks as a surprisingly strong method in meta-RL. While much effort has gone into designing specialized task-inference methods for meta-RL, we present the surprising result that the simpler recurrent methods can be easily adapted to outperform the task-inference methods. By combining recurrent methods with the hypernetwork architecture, we achieve a new strong baseline in meta-RL that is both robust and easy to implement. In comparison to existing evidence, we provide much stronger empirical results, afford equivalent computation for tuning to all

baselines, and establish recurrent hypernetworks as a strong method. We additionally show that passing the state variable to the policy is a crucial component of this method. Finally, we presented gradient analysis suggesting lower latent gradient norms to play an important role in the performance of meta-RL methods. Since the gradient analysis is preliminary and investigates state and latent variables in isolation, future work could investigate the interaction between these variables. Future work could also analyze the interaction between hypernetworks and other sequence models, such as transformers. We hope these insights, along with a simple and robust method, open the way for the broader use of sample-efficient learning in meta-RL and beyond.

5.7 Statement of Authorship

Chapter 5 is based on content from **Recurrent Hypernetworks are Surprisingly Strong in Meta-RL** [Beck et al., 2023b].

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the lead author on this manuscript. I was primarily responsible for the ideation, implementation, experimentation, and writing. Co-authors assisted, particularly with ideation and editing.

Citation:

Jacob Beck, Risto Vuorio, Zheng Xiong, and Shimon Whiteson. "Recurrent Hypernetworks are Surprisingly Strong in Meta-RL." *37th Conference on Neural Information Processing Systems*, 2023.

6

Sequence Modeling in Meta-Reinforcement Learning

Contents

6.1 Introduction

In Chapter 4 we saw that hypernetworks can be leveraged to drastically improve returns in zero-shot meta-RL, and in Chapter 5 we saw that, when using hypernetworks, end-to-end supervision is superior. In this chapter, we investigate inductive biases built into the sequence model in meta-RL that summarizes data input to the hypernetwork.

Recall that in order to condition on data from new tasks, end-to-end methods can use a generic sequence model, such as a recurrent neural network (RNN) [Duan et al., 2016, Wang et al., 2016]. These methods are referred to as *black-box* methods, as discussed in Chapter 2.

By contrast, *task-inference* methods, surveyed in Chapter 2, typically infer an explicit posterior over tasks, given data collected from a new task. To do so, they generally use distinct objectives and distinct sequence models, designed to enable inference of the unknown task. In particular, it is common to use sequence models that are invariant to the order of their inputs, which we refer to as permutation invariant aggregation. Due to the Markov property, the true posterior over tasks does not depend on this order.

While many task inference methods have been developed for meta-RL, recent work has shown black-box methods to be more effective in practice [Ni et al., 2022], which is corroborated in Chapter 5. However, these results focus primarily on demonstrating the superiority of the end-to-end objective used in black-box methods over the task-inference objective, and do not investigate the effect of the particular sequence model. This leaves an open question: *When using an end-to-end objective, is it still worth using permutation invariant sequence models?*

In this chapter, we answer in the affirmative. We show that permutation invariant sequence models still confer an advantage in a number of domains, even when trained

end-to-end. However, we also find, surprisingly, that there are domains where dependence on the permutation remains useful. Specifically, we find sequence models with a permutation variant component to be less sensitive to choices in the permutation invariant component, and we find permutation variance useful when there exist permutation variant suboptimal policies. We extensively investigate the conditions under which each type of sequence model is useful, conduct analysis to support our conclusions, and propose a simple sequence model, called Split Aggregator, or SplAgger, adapted and simplified from the literature on partial observability [Beck et al., 2020]. SplAgger, depicted in Figure 6.1c, uses both permutation invariant and permutation variant components to achieve the best of both worlds and high returns in all domains evaluated.

6.2 Related Work

End-to-End Meta-RL The problem setting defined by meta-RL can be viewed as a particular type of partially observable Markov decision process (POMDP) [Beck et al., 2023a]. From the theory of POMDPs, we know that the optimal policy for POMDPs, and thus meta-RL, can be represented as an arbitrary function of history [Subramanian et al., 2022]. Inspired by this, one category of meta-RL methods, called *black-box* methods, train general purpose sequence models end-to-end on the meta-RL objective [Duan et al., 2016, Wang et al., 2016, Ni et al., 2022, Team et al., 2023, Beck et al., 2023b]. Recently, it has been shown that these methods are a strong baseline in meta-RL [Ni et al., 2022]. Moreover, if hypernetworks [Ha et al., 2017] are used, these methods have superior performance to task inference methods, as seen in Chapter 5. We build off of these results, using end-to-end trained hypernetworks, following the methods developed in Chapters 4 and 5. However, in contrast to the prior chapters, we provide strong evidence that specialized sequence models, still trained end-to-end, can provide a strong advantage.

Sequence Models in Meta-RL Task-inference methods explicitly attempt to infer a posterior distribution over the identify of the task. Following directly from the Markov property, it can be shown that this posterior does not depend on the order of the data on which the agent conditions. While generic sequence models, such as RNNs, may model permutation invariance, they must learn to do so. In order to incorporate this inductive bias directly, methods generally modify the sequence model to be permutation invariant [Rakelly et al., 2019, Galashov et al., 2019, Raileanu et al., 2020, Wang and van Hoof, 2022, Imagawa et al., 2022]. One popular method, called probabilistic embeddings for actor-critic RL, or *PEARL* [Rakelly et al., 2019], incorporates permutation invariance into the probability density function of a stochastic latent variable summarizing history. Specifically, the density function, modelled as a product over individual transitions in the data, is permutation invariant. We compare to this style of aggregation in our experiments. Another approach uses commutative operators applied across the data [Imagawa et al., 2022, Wang and van Hoof, 2022, Galashov et al., 2019]. Generally, these can be viewed as (conditional) Neural Processes (CNP) [Garnelo et al., 2018b,a], and so we compare to this style of aggregation as well. Yet another approach uses attention, self-attention, or transformers [Mishra et al., 2018, Fortunato et al., 2019, Nguyen and Grover, 2022]. Attention is inherently permutation invariant. Still, attention is computationally expensive: whereas both commutative aggregation and recurrent networks use $O(1)$ memory and compute per timestep, attention generally requires $O(t^2)$ memory and compute per timestep t, for autoregressive inference. While fast approximations of attention exist [Katharopoulos et al., 2020], the computational requirements are significantly larger. Additionally, our limited experimentation shows they have difficulty learning on our domains (see the appendix for details). Thus, we limit our solutions to constant memory and compute, in line with sequence models designed to quickly handle long contexts [Garnelo et al., 2018b, Beck et al., 2020]. Finally, we compare to Aggregated Memory for RL, or

AMRL [Beck et al., 2020]. While AMRL was originally proposed as a method for POMDPs, it was evaluated in meta-RL problem settings. Our method proposes a simplification to AMRL that is vital in practice. Details of AMRL are covered in Section 6.4.

In-Context Learning The methods we investigate in this chapter can be seen as performing *in-context* learning. Learning that occurs after training and within the activations of a sequence model is called *in-context* learning [Brown et al., 2020]. Black box and task inference methods both perform in-context learning. In part due to the popularity of large language models [Devlin et al., 2019, Brown et al., 2020, Chowdhery et al., 2022], in-context learning has gained significant traction recently, including in decision-making applications [Raparthy et al., 2023, Lee et al., 2024] and reinforcement learning (RL) [Kirsch et al., 2022]. While other meta-RL methods exist that do not use sequence models for in-context learning, such as parameterized policy gradient methods, they generally require more samples to adapt to novel tasks, including those used in our benchmarks [Zintgraf et al., 2021b, Beck et al., 2023a]. In-context learning promises to address the sample inefficiency still impeding progress in reinforcement learning. Learning to perform in-context RL is the problem studied in meta-RL.

6.3 Background

6.3.1 Notation

Recall that τ is a sequence forming a trajectory of states, actions, rewards, and next states. Here, τ may span multiple episodes within a single MDP, since multiple episodes of interaction may be necessary for learning. Collectively, we refer to these episodes as a *meta-episode*, and use the same symbol, τ, to refer to it. We can see

τ as a trajectory of transitions, $\tau_0, \tau_1, ..., \tau_t$, where τ_t is shorthand for the transition, (s_t, a_t, r_t, s_{t+1}). We use this notation for transitions throughout this chapter.

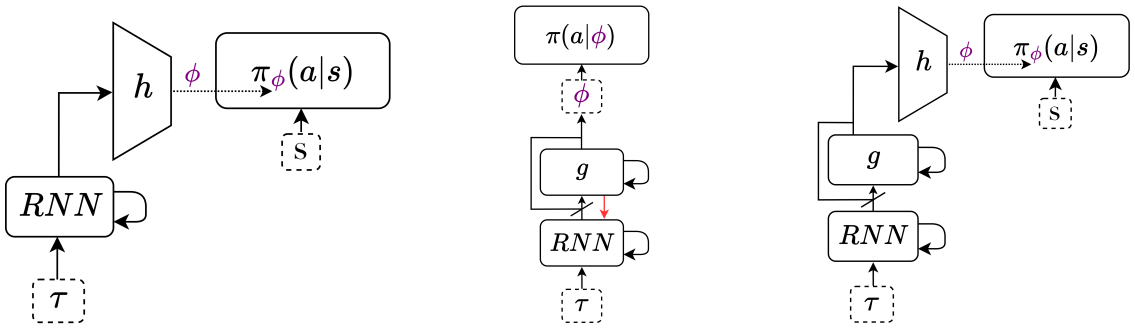
6.3.2 Permutation Invariance

Task inference methods in meta-RL explicitly infer a posterior over tasks. In meta-RL, the optimal policy can be computed from both the current state, s_t, and this posterior, $P(\mathcal{M}|\tau)$. Following Bayes's rule and the Markov property, this posterior distribution, for a trajectory of length T, can be written in a form that is independent of the order of historical transitions.

Proposition 1 (Permutation Invariance of the Posterior) *Let $\mathcal{M}$ denote a task (MDP), drawn from a prior distribution $P(\mathcal{M})$, and let τ be a trajectory, defined as above, generated under a task $\mathcal{M}$ and policy π.*

$$
\begin{aligned}
& P(\mathcal{M}|\tau) && && (6.1)\\
&= \frac{P(\tau|\mathcal{M})P(\mathcal{M})}{P(\tau)} && \textit{// Bayes's Rule} && (6.2)\\
&\propto P(\tau|\mathcal{M})P(\mathcal{M}) && && (6.3)\\
&= P(\tau_1, \tau_2, ..., \tau_T|\mathcal{M})P(\mathcal{M}) && && (6.4)\\
&= P(\mathcal{M}) \prod_{t=1}^{t=T} P(\tau_t|\tau_1, \tau_2, ..., \tau_{t-1}, \mathcal{M}) && && (6.5)\\
&= P(\mathcal{M}) \prod_{t=1}^{t=T} P(\tau_t|s_t, \mathcal{M}) && \textit{// Markov Property} && (6.6)\\
&= P(\mathcal{M}) \prod_{t=1}^{t=T} P(s_t, a_t, r_t, s_{t+1}|s_t, \mathcal{M}) && && (6.7)\\
&= P(\mathcal{M}) \prod_{t=1}^{t=T} P(a_t, r_t, s_{t+1}|s_t, \mathcal{M}) && && (6.8)
\end{aligned}
$$

In this expression, the posterior, $P(\mathcal{M}|\tau)$, does not depend on the order of the transitions, $\tau_1, .., \tau_t$. While it is possible to learn each factor, $P(a_t, r_t, s_{t+1}|s_t, \mathcal{M})$, as a function of each individual transition, τ_t, this form is not particularly amenable to



(a) Hypernetwork **(b)** AMRL **(c)** SplAgger

Figure 6.1: The hypernetwork from Chapter 5 is depicted in 6.1a, the AMRL model from Beck et al. [2020] is depicted in 6.1b, and SplAgger is depicted in 6.1c. The angled line indicates a split connection that divides the neurons in half. The red arrow indicates a modified gradient computation in the backward pass. A hypernetwork is indicated by h. SplAgger makes use of the hypernetwork architecture combined with the AMRL sequence model. The hypernetwork architecture is necessary for performant end-to-end training. Critically, SplAgger also removes the gradient modification from AMRL which we show to be deleterious to performance.

inference, since it requires marginalizing over all MDPs at test time. Still, it is possible to incorporate the permutation invariant structure into the sequence model directly.

Generally this is done through the use of a permutation invariant binary operator, $\bigoplus$, such as a mean or a sum [Garnelo et al., 2018b,a, Galashov et al., 2019, Imagawa et al., 2022, Wang and van Hoof, 2022], applied over the sequence of inputs. For a given sequence of encoded inputs, $e_1 ..., e_t$, we write the resulting aggregated representation, $g(e_1, ..., e_t)$:

$$g(e_1, ..., e_t) = e_1 \bigoplus e_2 \bigoplus ... \bigoplus e_t. \tag{6.9}$$

The benefit of using these operators, in addition to permutation invariance, is that the sequence can be computed recursively, since $g(e_1, ..., e_t) = g(e_1, ..., e_{t-1}) \bigoplus e_t$. This means that the sequence at all points in time can be compressed into $O(1)$ memory and each new timestep can be computed in $O(1)$ time. Still, as we will demonstrate in practice, permutation invariance can be beneficial in some environments and detrimental in others.

6.3.3 AMRL

AMRL, depicted in Figure 6.1, is a method for POMDPs that combines both permutation variant and permutation invariant components [Beck et al., 2020]. While AMRL uses permutation invariant aggregators, the encoded inputs to the aggregators themselves are a function of history:

$$e_t = RNN_\theta(\tau_1, ..., \tau_t). \tag{6.10}$$

Additionally, the neurons of each encoded input are split in half before aggregation, into $e_{t,1/2}$ and $e_{t,2/2}$. The first half of the neurons are aggregated, while the second half skip the aggregation. The complete sequence model is defined as

$$f_\theta(\tau) = \text{concatenate}(e_{t,1/2}; g_\theta(e_{1,2/2}, ..., e_{t,2/2})). \tag{6.11}$$

Here, the RNN is able to handle short permutation variant sequence, which can then be integrated without respect to order by the permutation invariant aggregation over longer periods of time. Since we split the neurons before aggregation, we call this process *split aggregation*.

Additionally, AMRL modifies each Jacobian, $\frac{dg}{de_{i,2/2}} \forall_i$, when computing the chain rule in the backward pass. Specifically, it overwrites the true Jacobian with the identity matrix, I. This is called passing the gradient *straight through*, or an *ST gradient modification*. Since the sum aggregator actually has I as the true Jacobian, this can be seen as replacing the gradient with that of a sum. For the average and max aggregators, the Jacobian is already similar to I. Specifically, the average has a Jacobian that is I/t, and the max has an expected Jacobian that is I/t, under mild assumptions. Thus, the modification can be seen as a rescaling of the gradient that does not diminish with time. Beck et al. [2020] hypothesize that the ST modification has no negative impacts while also preventing gradient decay.

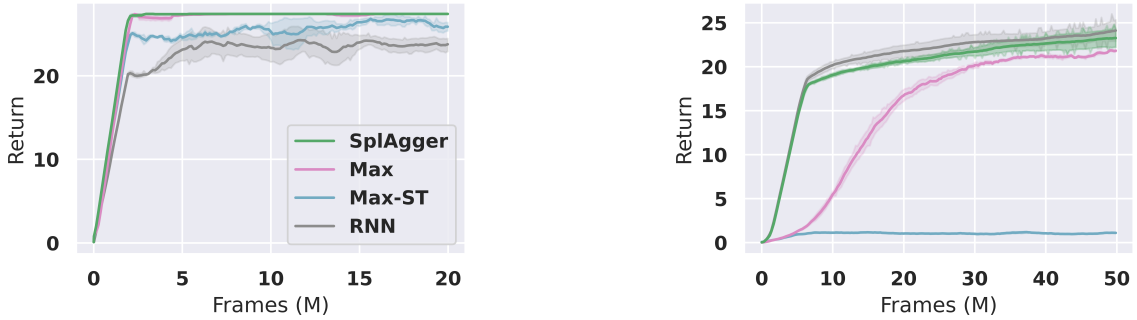
6.4 Methods: SplAgger

Here we present our model, Split Aggregator, or SplAgger. As motivation, we first present a preview of our experimental results. In Figure 6.2, we experiment with the permutation invariant model using a point-wise maximum aggregator, as suggested in AMRL, and a permutation variant model, the RNN. Permutation variance improves performance on the some domains (Figure 6.2a), but decreases performance on others (Figure 6.2b). Moreover, when we add the ST gradient modification, performance decreases severely. These result motivate the need for our model, SplAgger, which achieves the highest returns in both domains.

SplAgger uses the same split aggregation as in AMRL, but without the ST gradient modification. Simplifying AMRL by removing this gradient modification is key to its performance. While the ST modification does prevent one type of gradient decay, permutation invariant aggregators already address the relevant type of gradient decay. Moreover, the ST aggregator causes both the explosion of other gradients and a severe decrease in performance. We show that this is the case in Sections 6.5 and 6.6, motivating the need for SplAgger. Additionally, SplAgger uses a different architecture for the policy than in AMRL. Our previous results show that the hypernetwork [Ha et al., 2017] architecture is critical in unlocking the performance of end-to-end objectives and enabling black box methods to outperform task inference methods. Thus, the main idea behind SplAgger is to add AMRL to hypernetworks trained end-to-end but remove the ST gradient modification. SplAgger is the combination of these components and is depicted in Figure 6.1c.

6.5 Experiments

In this section we evaluate SplAgger on several domains. First, we evaluate on two standard meta-RL benchmarks in Section 6.5.1, to make sure that the aggregation



(a) MC-LS **(b)** Planning Game

Figure 6.2: A preview of later results. The permutation invariance of the max aggregator improves returns relative the RNN on the MC-LS environment [Beck et al., 2020], but decreases returns on the Planning Game [Ritter et al., 2021]. Additionally, the ST gradient decreases the returns of the max aggregation. These results motivate SplAgger, which achieves the highest returns. (Results are reported with a 68% confidence interval, computed through bootstrapping with 1,000 iterations across three seeds, consistent with all plots presented.)

method does not harm performance on environments without large demands on the sequence models. Second, we evaluate on two prior meta-RL benchmarks designed to test sequence models in mazes in Section 6.5.2. We additionally evaluate on three environments design to systematically test different components of SplAgger in Section 6.5.3.

On the four primary benchmark environments, we compare to four baselines. Hyperparamter tuning is detailed in the appendix. Since prior results demonstrate the need for hypernetworks when training end-to-end (Chapter 5), all baselines have been evaluated using hypernetworks, with design choices detailed in the appendix. We additionally present negative results on a novel initialization method in the appendix. The baselines evaluated primarily differ in their choice of aggregation function, g, and encoding of inputs, e_t. The baselines are described below.

RNN. The RNN baseline can be written $f(\tau) = e_t = RNN(\tau_1, ..., \tau_t)$. Here there is no aggregation function, g, and the baseline uses a standard gated recurrent unit [Cho et al., 2014], as in Zintgraf et al. [2021b] and Beck et al. [2023b].

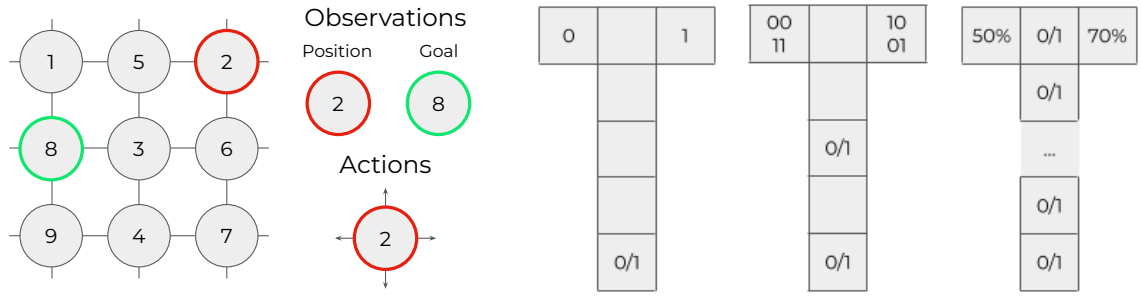
CNP. The conditional neural process (CNP) consists of permutation invariant aggregation without any additional components [Garnelo et al., 2018a]. Specifically, $f(\tau) = g(e_1, ..., e_t)$. Here, e_t is a linear encoding of τ_t. We use the mean operator for g, as suggested by Garnelo et al. [2018a].

AMRL. AMRL [Beck et al., 2020] uses an RNN to encode e_t in addition to permutation invariant aggregation, and is described in Section 6.3. For AMRL, we use the pointwise maximum aggregator in our experiments, both to match the aggregator used in SplAgger, and because that aggregator was found to be strongest by Beck et al. [2020].

PEARL. The PEARL baseline uses the aggregation method from the PEARL algorithm [Rakelly et al., 2019], which incorporates permutation invariance into the probability density function of a stochastic latent variable summarizing history. The density function is modelled as a product over individual transitions in the data. We can write this as $f(\tau) = g(e_1, ..., e_t) = z \sim \alpha \prod_{t=1}^{t=T} \mathcal{N}(z; \mu_t = e_{t,1/2}, \sigma_t^2 = diag(e_{t,2/2}))$, where e_t is a linear encoding of τ_t and α is a normalizing constant. To compare the effects of aggregation in isolation, our PEARL baseline only implements the aggregation method used in PEARL, and leaves the rest of the algorithmic choices the same as in SplAgger. Additional design choices and hyperparameters for PEARL are presented in the appendix.

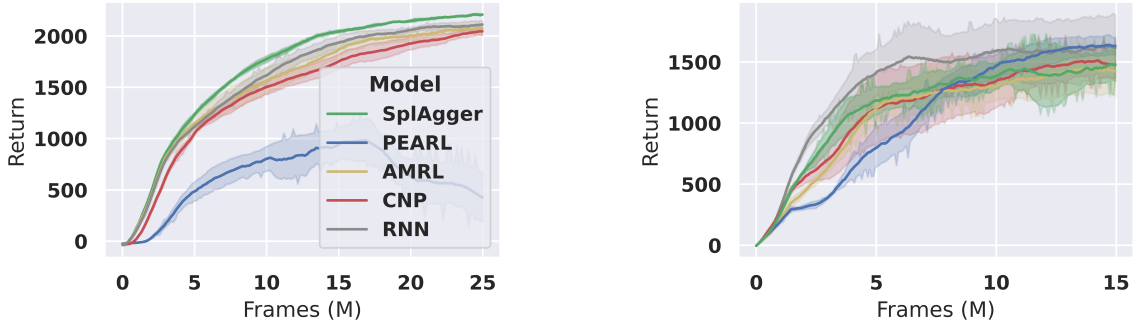
6.5.1 MuJoCo Benchmarks

The first two environments for benchmarking are variants of MuJoCo proposed by Zintgraf et al. [2021b], and both involve legged locomotion. While these environments have no great demands on memory, they are common meta-RL benchmarks [Humplik et al., 2019, Rakelly et al., 2019, Zintgraf et al., 2021b, Beck et al., 2022, 2023b]



(a) Planning Game **(b)** T-LS (T-Maze) **(c)** T-Maze Agreement **(d)** T-Maze Latent

Figure 6.3: Depictions of the Planning Game, T-LS, T-Maze Agreement, and T-Maze Latent environments used in Sections 6.5.2 and 6.5.3.

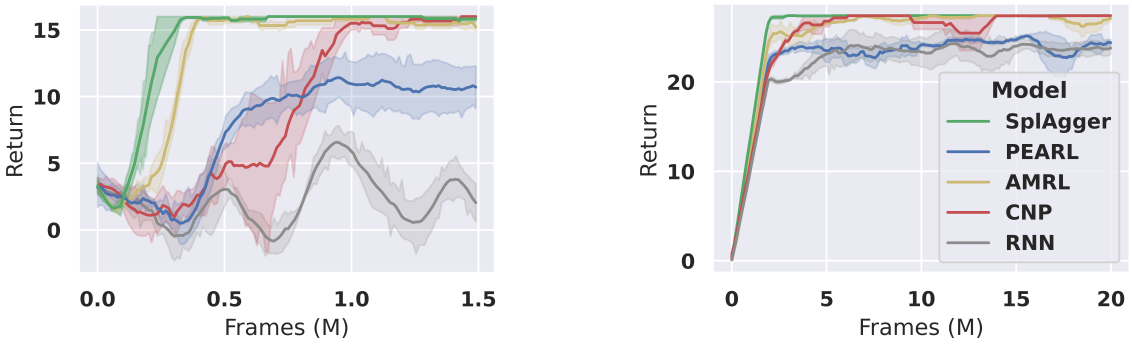


(a) Cheetah-Dir **(b)** Walker

Figure 6.4: Results on MuJoCo benchmarks. SplAgger achieves the same or better results on both domains. PEARL achieves significantly lower return on Cheetah-Dir.

that enable us to evaluate what effect SplAgger has on standard RL tasks. See the appendix for details on the environments.

Results are shown in Figure 6.4. SplAgger achieves the greatest return, though the improvement is modest. Training the PEARL baseline on this domain is unstable and PEARL receives significantly lower returns. On Walker, we see similar performance across all methods. Overall, SplAgger achieves similar or greater returns compared to other baselines. This demonstrates that our method, designed to improve environments with difficult demands on memory, also does not decrease performance on domains with limited memory requirements.



(a) T-LS **(b)** MC-LS

Figure 6.5: Results on memory benchmarks. SplAgger achieves the highest returns on both domains, indicating the fastest learning. The standard RNN is not able to learn on either domain within the allotted number of frames.

6.5.2 Memory Benchmarks

We additionally conduct tests on two environments, T-LS and MC-LS, proposed by Beck et al. [2020]. Both of these environments were designed to test long-term memory, and we use the latter in Chapter 5. The T-LS environment is depicted in Figure 6.3b. The MC-LS environment is designed to challenge an agent's long-term memory based on visual cues from Minecraft. Environment details can be found in the appendix.

Results in Figure 6.5 show that SplAgger achieves the highest sample efficiency on both environments. The RNN and PEARL are not able to learn the optimal policy within the allotted number of frames. The reasons for the failure of the RNN are discussed in Section 6.6, while potential reasons for the failure of PEARL are analyzed in the appendix. While CNP is able to learn optimally, it requires significantly more frames on T-LS. AMRL achieves similar performance. Both AMRL and CNP are similar to our method, SplAgger. However, AMRL differs in its gradient estimation and CNP differs in its use of mean aggregator, instead of max, and its lack of an RNN. To fully understand the contribution of each, we systematically modify these components in isolation, in the next section.

6.5.3 Alternative Aggregation

In this section we introduce modifications of SplAgger, and three new environments designed to test these modifications. Since the existing baselines differ in their choice of aggregation function, g, encoding of inputs, e_t, and use of the ST gradient modification, we systematically test these differences here. The most relevant additional baselines are described below, with the rest detailed in the appendix.

SplAgger-noSplit. SplAgger-noSplit removes the split aggregation from SplAgger, but still uses max aggregation and an RNN to encode e_t. Comparing to this method allows us to validate the use of the split aggregation in SplAgger.

SplAgger-noRNN. SplAgger-noRNN removes the RNN from SplAgger in order to test the effects of removing permutation variant components. Since this obviates the need for the split connection, that component is removed as well. Without these components, this method is equivalent to just computing a maximum over linear encodings of each transition, τ_t.

AMRL-noRNN. AMRL-noRNN removes the RNN and split connection from AMRL. Without these components, this method is equivalent computing a maximum over linear encodings of each transition, τ_t, along with the ST gradient modification.

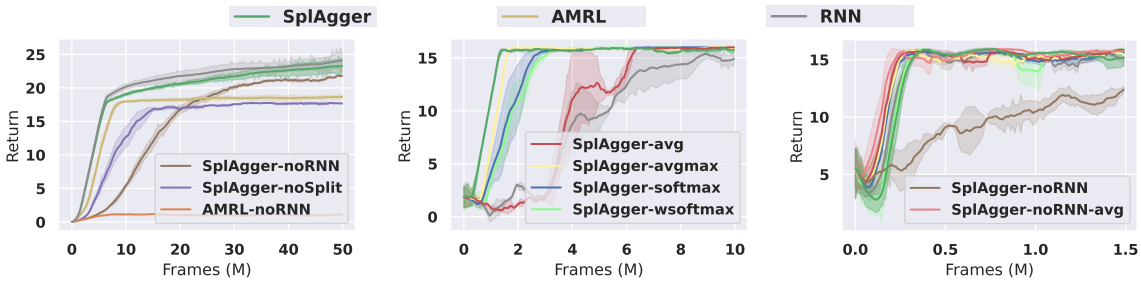
SplAgger-avg. SplAgger-avg replaces the max operator in SplAgger with an average, in order to test the effects of alternative permutation invariant operators. SplAgger with other operators (avgmax, softmax, wsoftmax) that interpolate between the average and max are evaluated as well, with details in the appendix.

Planning Game. First, we evaluate on the Planning Game Ritter et al. [2021], in order to evaluate the need for permutation variant components and how to combine

them with permutation invariant components. This environment tests an agents ability to discover and remember multiple pieces of information required for subsequent navigation. The Planning Game is useful for evaluation here due to the existence of both a permutation invariant optimal policy and permutation invariant suboptimal policy. The environment is depicted in Figure 6.3a and detailed in the appendix.

On this domain, we evaluate methods that modify how the RNN is combined with the permutation invariant aggregation. Results in Figure 6.6a show that SplAgger and RNN learn the fastest in this domain. Both AMRL and SplAgger without the split connection learn a suboptimal policy. This demonstrates the detrimental effects of the AMRL gradient modification and the benefit of the split aggregation, which motivates SplAgger. We also see that AMRL without an RNN fails to learn any reasonable policy, achieving near zero reward. Since the only difference between this and SplAgger without an RNN is the use of the ST gradient modification, this shows strong evidence of the detrimental effects of the gradient modification in AMRL. We analyze the causes in Section 6.6. While both SplAgger with an RNN and SplAgger without an RNN outperform the sub-optimal exploration policy eventually, and achieve similar returns ultimately, SplAgger with an RNN learns faster initially. We discuss this further in Section 6.6.

T-Maze Agreement. Second, we evaluate on T-Maze Agreement in order to investigate the specific permutation invariant operator, $\bigoplus$. In this environment, the agent receives two binary signals: one at the beginning of the maze and one in the middle. The agent must open a door depending on whether the signals agree or disagree. This environment is depicted in Figure 6.3c. While the max aggregation can easily identify the support of the state distribution in the data [Beck et al., 2020], and thus easily identify which signals have been seen in each state, the average aggregation must learn to adjust the representation of all states to interpret the average. Thus, we



(a) Planning Game **(b)** T-Maze Agreement **(c)** T-Maze Latent

Figure 6.6: SplAgger achieves returns that are equal to or higher than other methods. The Planning Game shows the importance of incorporating RNNs, either in isolation or with split aggregation, and the failure of gradient modification as in AMRL. Combining the RNN and permutation invariant aggregation, without the split connection (SplAgger-noSplit), decreases performance of each. The T-Maze Agreement domain shows the max operator to be beneficial, enabling performance even greater than the RNN when used with SplAgger. The T-Maze Latent environment shows that SplAgger is able to make the max aggregator performant, even in environments where the computing the average alone is superior. Note that each legend shows the additional methods introduced for that experiment, while the legend at the top shows methods from prior experiments.

hypothesize that this environment is easier for max aggregation and harder for mean aggregation.

On this domain, we evaluate methods that modify the specific permutation invariant operators. Results in Figure 6.6b show that SplAgger achieves the highest returns, demonstrating the superiority of max aggregation in this environment. While the RNN and average variant of SplAgger are able to learn, they require more frames, as expected. The avgmax, softmax, and wsoftmax all learn almost as quickly as SplAgger. Thus, we see that SplAgger is fairly robust to the choice of operator in this domain, as long as it computes some information about the maximum.

T-Maze Latent. Finally, we evaluate on T-Maze Latent, which also modifies the T-Maze environment in order to investigate the specific permutation invariant operator. In this environment, the agent receives an indicator at every timestep. This indicator is either drawn from $\{0, 1\}$ with a 50% chance of 1 or a 70% chance of 1, depending on

the task. This environment is depicted in Figure 6.3d. The average aggregator should quickly reveal the latent variable, as the variance of the mean decreases, whereas the max aggregator should have a more difficult time counting the occurrence of indicators in different states. Thus, we hypothesize that this environment should be harder for max aggregation methods and easier for average aggregation.

On this domain, as on T-Maze Agreement, we evaluate methods that modify the specific permutation invariant operators. Results in Figure 6.6c show that, surprisingly, all operators used with SplAgger learn at approximately the same rate. We hypothesize that here, SplAgger is able to fall back upon leveraging the RNN. Since the environment was designed to be more difficult for the max operator, we predict that there may be a difference between the operators when the RNN and skip connection of SplAgger are not used. To test this hypothesis, we conduct two additional experiments. We test both SplAgger without an RNN, but still with the default max operator (SplAgger-noRNN), and SplAgger without an RNN but with the average operator (SplAgger-noRNN-avg). Since the removal of the RNN obviates the need for the split aggregation, the split aggregation is removed as well. We see that, as predicted, the method with the average operator performs better than the method with the max operator, when the RNN and split connection are removed. This demonstrates that the combination of the RNN and split aggregation make SplAgger remarkably robust to different environments, and justifies both our aggregation method and the max operator.

6.6 Analysis

In this section we analyze different sequential models to gain insights into their performance. We investigate why RNNs remain useful in some cases, even when permutation invariance should be sufficient, and why AMRL performs poorly in our experiments. Specifically, we find permutation variance to be useful when there exist permutation variant suboptimal policies that form a useful stepping stone for learning

optimal policies later. Additionally, we find that both AMRL and SplAgger prevent certain types of gradient decay, but AMRL also causes other gradients to explode.

6.6.1 Learning Suboptimal Policies

While sensitivity to permutation is not required to learn optimal policies in meta-RL, we find that the RNN surprisingly improves sample efficiency on the Planning Game. As discussed in Section 6.5, the Planning Game has a permutation variant suboptimal policy. This policy re-explores all states after every goal is reached. To do this, the agent must remember where it is in a sequence of exploratory actions, and then restart the sequence when a new goal is found. We examine rollouts to confirm that SplAgger first learns this suboptimal policy. While SplAgger without an RNN can surpass the suboptimal policy eventually, it also achieves achieves lower returns throughout training compared to an RNN, since the RNN learns the suboptimal policy faster. Thus, RNNs can achieve higher returns sooner, when there is a permutation variant suboptimal policy that can act as a stepping stone for learning the optimal policy. We hypothesize that the reason SplAgger without an RNN cannot learn the suboptimal policy is that the max aggregation can record that a goal has been reached, but cannot identify when that goal was reached, or that it was *just* reached.

To confirm this, we perform an additional experiment, in which the agent receives no observation of the state, creating partial observability. All it observes is whether it is currently at the goal state. Hence, there is no way for it to distinguish which MDP it is in, and the problem can no longer be modelled as a distribution of MDPs. In this case, the observations are not Markov and permutation invariant aggregation no longer suffices for decision making. Still, the optimal policy in this environment requires the same strategy as the suboptimal policy in the Planning Game: explore every state until the goal is reached. When the agent reaches a goal, the goal location is reset, and the agent must restart exploration. Figure 6.7 shows that the RNN is able

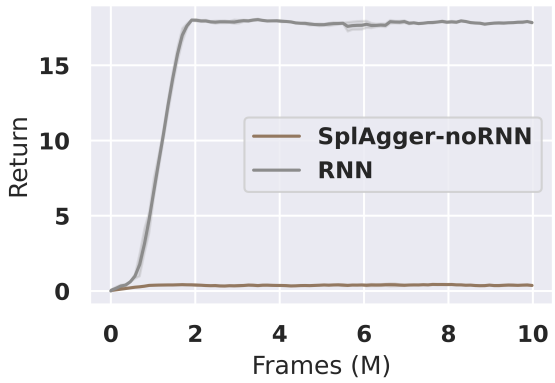


Figure 6.7: Here we show results on a modified Planning Game. If no observation is given to the agent, an RNN is required to learn the proper exploration strategy. This is the same exploration as required by the easier sub-optimal policy in the original Planning Game.

to learn this policy, while SplAgger without an RNN is not. This shows how, even in a distribution of MDPs, where the Markov property holds in each MDP, permutation variance can improve sample efficiency, due to the presence of non-Markov policies that are suboptimal but faster to learn.

6.6.2 Gradient Decay

Finally, we investigate the gradients of our sequence models to explain why, in some domains, SplAgger works, while AMRL [Beck et al., 2020] does not. AMRL demonstrates the benefit of its gradient modification by measuring the average gradient of the memory with respect to the encoding of the initial transition, $||\frac{df_t}{d\tau_0}||_2$. AMRL shows that this quantity decays over time for normal sequence models, but not for AMRL, due to the gradient modification. AMRL methods overwrite this gradient to set it equal to the identity.[1] We depict these gradients at initialization in Figure 6.8a, evaluating over three model initializations.

While AMRL prevents gradient decay, it also causes gradient explosion, with respect

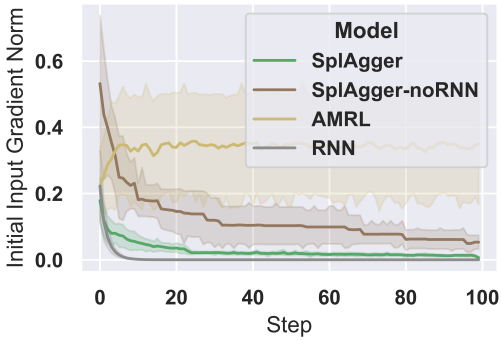
[1]Beck et al. [2020] also note that the signal-to-noise ratio can also affect performance. However we can only recreate this result by setting the bias in all models to zero, and find it less predictive than the gradients regardless.

to the model parameters, $||\frac{df_t}{d\theta}||_2$. Since the number of inputs grows over time, and the norm of the gradients for each input does not shrink, the gradients with respect to the parameters grows. We depict this phenomenon in Figure 6.8b. For an input, we sample noise uniformly over $[-1, 1]$, and replicate this sample for every dimension in the input to the sequence model.

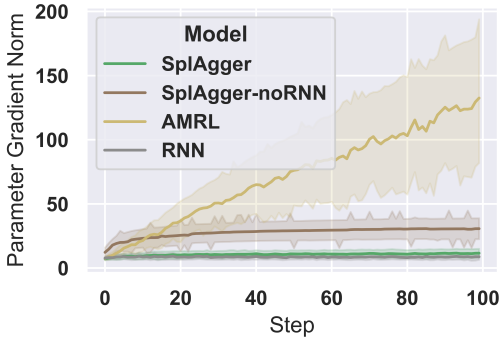
From these two gradients, it is not clear why SplAgger performs better than an RNN, so we propose two alternative metrics for evaluation. In Figure 6.8c, we plot the gradient of the inputs, over time (t), holding the output time (T) fixed: $||\frac{df_T}{d\tau_t}||_2$. This value is roughly constant for all models, except for the RNN. For an RNN, it grows as t approaches T, implying that, for a fixed output, the inputs become less sensitive backward in time. In other words, by privileging recent transitions, the gradients are not permutation invariant. The gradients are not equal for all inputs, for a given output. In addition to this metric, we can measure the permutation variance directly. In Figure 6.8d, we compute the mean difference between encodings of different permutations of inputs at initialization. We also normalize the encodings, to have unit magnitude first. We see that the RNN is the most permutation variant, and sequence models without any RNN, such as SplAgger without an RNN, are the least. Critically, models like SplAgger that perform best are not the most or least permutation variant, but rather have some components of each.

6.7 Conclusion

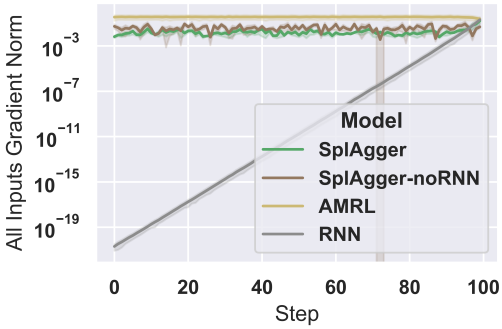
To conclude, we first consider some limitations of this chapter and then summarize our key findings. While our proposed method, SplAgger, improves the performance in all domains evaluated, we cannot guarantee improvement in all environments. In contrast to our work in Chapter 5, which moved our methods away from the meta-RL specific objectives and in the direction of more general methodology, the work in this chapter proposes a sequence model with a highly specific structure. To mitigate this limitation,



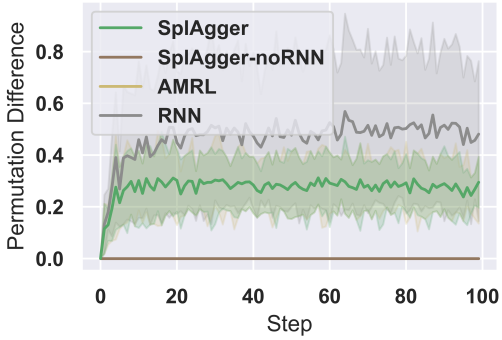
(a) Gradient w.r.t. Initial Transition



(b) Gradient w.r.t. Parameters



(c) Final Gradient w.r.t. All Transitions



(d) Encoding Permutation Difference

Figure 6.8: An empirical analysis of gradients in SplAgger, AMRL, and RNNs. In 6.8a we see that the gradient of the output of the sequence model with respect to the initial input decreases over time. AMRL modifies the gradients to prevent this, but at the cost of exploding gradients with respect to the parameters, depicted in 6.8b. We find that poor performance of the RNN is rather due to earlier inputs having a smaller gradient, when considering the final output of the sequence model. depicted in 6.8c. Finally, models that are the least permutation variant do not necessarily perform better; the highest performing model, SplAgger, has an intermediate difference, but has components that are both permutation variant and permutation invariant. This is shown in 6.8d.

we evaluate in standard control domains, more complex memory domains, and custom domains designed to test the specific structure to ensure robustness to architectural choices. Indeed, we find that precisely by combining permutation variant and invariant components, we are able to achieve dominant performance. That said, even if the specific sequence model proposed, in the form of SplAgger, is surpassed in the future, we believe two key contributions in this chapter remain: We show previously accepted gradient approximations to be detrimental in practice, and we show that, despite

permutation invariance being sufficient in theory for the optimal policy, permutation variance is useful for suboptimal policies on the path to the optimal policy. For the latter reason, whether or not in the specific form of SplAgger, we have demonstrated that any solution to the meta-RL problem can benefit from permutation variant and invariant components. We hope and believe that this insight will pave the way for future improvements in the sequence modeling for meta-reinforcement learning.

In this chapter we have shown how permutation invariance can be critical when learning to reinforcement learn. We have, for the first time, confirmed this advantage even without the use of task inference objectives. Surprisingly, we also demonstrate that permutation variance can still be useful, both to learn sub-optimal non-Markovian policies early on, and to make the sequence model more robust to the choice of specific aggregation function. Using these insights, we presented SplAgger, making use of split aggregation to achieve the best of both methods. Moreover, we have shown that in several domains, popular existing methods fail, and discussed reasons for the failure of each. We analyzed how the gradient modification in AMRL causes gradients with respect to the parameters to explode, and measured different types of gradient decay and permutation variance in RNNs. We hope that these insights, in addition to our methodological contributions, pave the way for representations in future meta-RL agents.

6.8 Statement of Authorship

Chapter 6 is based on content from **SplAgger: Split Aggregation for Meta-Reinforcement Learning** [Beck et al., 2024a].

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the lead author on this manuscript. I was primarily responsible for the ideation, implementation, experimentation, and writing. Co-authors assisted, particularly with code optimization and editing.

Citation:

Jacob Beck, Matthew Jackson, Risto Vuorio, Zheng Xiong, and Shimon Whiteson. "SplAgger: Split Aggregation for Meta-Reinforcement Learning." *The first annual proceedings of the Reinforcement Learning Journal*, 2024.

7

Meta-Learning in Practice: Protein Fitness Prediction

Contents

7.1 Introduction

In the previous chapters we have seen methods improving the representations of meta-RL algorithms. In this chapter, we apply meta-learning to zero-shot and few-shot protein fitness prediction, across a broad distribution of both proteins and associated properties, in order to examine how meta-learning methods function in practice. The application to protein design was required by the constraints of a research internship, which provided the necessary collaboration and computation. The problem is highly suitable as an application due to the existence of a recognized public benchmark, the ability to train without simulation and the associated simulation-to-reality gap, the lack of existing meta-learning solutions, and the significant implications for the physical sciences of any advancements. However, notably, this chapter departs from the previous chapters in that it examines meta-supervised learning, as opposed to meta-reinforcement learning. This shift itself is a practical limitation imposed by the application. While RL has been applied as a framework for protein design [Angermueller et al., 2020], doing so required on the order of hundreds or thousands of samples, rather than the limited data (0-shot, 16-shot, and 128-shot) setting considered here. While meta-RL could be used to address this sample inefficiency in principle, it is not practically applicable in our setting. Since each task in our task distribution can be uniquely identified from any data, the optimal design for each protein can be inferred immediately. With only (up to) 121 tasks, a policy could easily memorize the solutions rather than learn to explore the design space of a protein. Thus, this chapter focuses on a *sub-problem* of protein design, known as *protein fitness prediction.*

The accurate prediction of the functional and biophysical properties of proteins, such as stability or binding affinity, is a critical challenge in the physical sciences with far-reaching implications for drug discovery, medical research and agriculture. Protein design seeks to optimize one, or a collection of these properties, referred to as fitness.

While protein fitness can be measured *in vitro*, the process is laborious and time-consuming. Consequently, machine learning models have emerged as a powerful tool to predict fitness directly from amino acid sequences *in silico*. However, due to the complex, high-dimensional relationship between protein sequences and fitness, and the limited availability of high-quality data, accurate fitness prediction is a challenge.

Proteins can be encoded as sequences of characters representing amino acids, making protein language models (PLMs) effective for modeling them [Madani et al., 2020, Rives et al., 2021, Rao et al., 2021, Lin et al., 2022, Notin et al., 2023, Truong Jr and Bepler, 2024]. By predicting masked amino acids, or subsequent amino acids, over known proteins at scale, PLMs can capture much of the structure and resultant properties deriving from the amino acid sequence. While PLMs are not directly trained to predict fitness, they are trained to model the likelihood of naturally occurring proteins, which has been found to correlate strongly with their fitness [Truong Jr and Bepler, 2024]. In practice, large PLMs are fine-tuned on downstream protein fitness data for regression. While highly effective, PLMs provide limited utility given severely limited data, or no data. With limited data, learning the correct regression is difficult, even with informative pre-trained representations. With no data, it must be assumed that protein fitness is solely a function of protein likelihood [Truong Jr and Bepler, 2024, Hawkins-Hooker et al., 2024], and for masked language models, it is also assumed that each amino acid contributes independently to fitness [Meier et al., 2021, Notin et al., 2024].

While limited or no data may be available for specific protein fitness prediction tasks, there are often other tasks that can serve as a valuable source of guidance. For example, rather than assuming that fitness is solely a function of protein likelihood, when no data is available for a given task, we can use data from other tasks to learn the relationship between PLM embeddings and fitness. Due to advances in high-

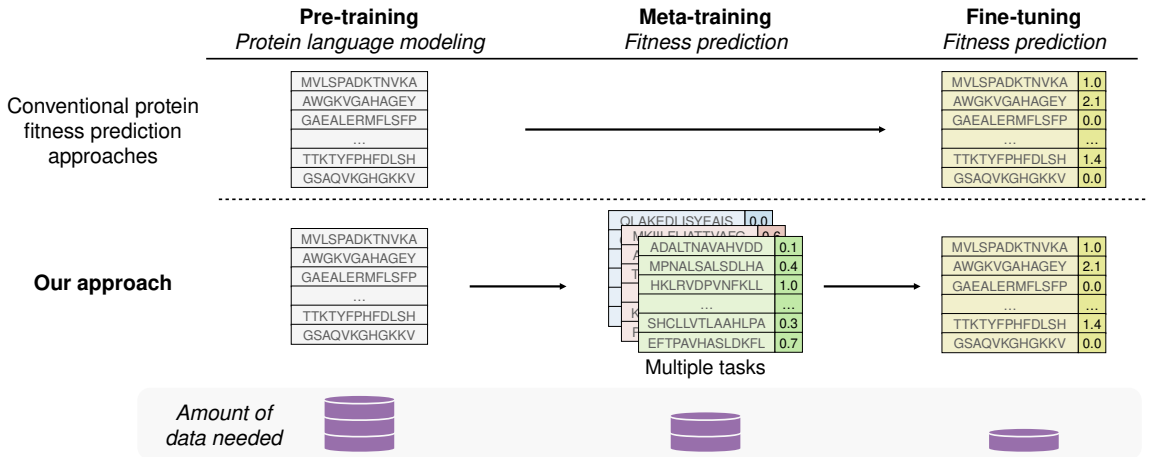


Figure 7.1: An overview of meta-learning for protein fitness prediction. PLMs are trained over massive quantities of unlabeled data. Using meta-learning, we also train over a smaller quantity of labelled fitness data. Using this extra data is critical given limited data for fine-tuning at test time.

throughput assays, such as deep mutational scanning [Fowler and Fields, 2014], data from other tasks is available to learn this relationship. One example is ProteinGym [Notin et al., 2024], which compiles over one hundred distinct fitness prediction tasks. Training over such a distribution of related tasks, to reason about new tasks, is a type of *meta-learning* [Huisman et al., 2021, Hospedales et al., 2021, Beck et al., 2023a].

To address the challenge of limited data in protein fitness prediction, we propose *Metalic*, integrating in-context meta-learning, protein language models, and fine-tuning. *Metalic* builds on top of PLM embeddings and fine-tuning methods, but additionally, unlike baseline methods, adds a meta-learning phase over other protein prediction tasks. While some PLMs use in-context data [Rao et al., 2021, Notin et al., 2022, 2023, Truong Jr and Bepler, 2024], these methods do not *meta-learn* how to use their limited context for protein fitness prediction. The meta-learning phase is depicted in Figure 7.1, and is critical to learning the relationship between PLM embeddings and fitness, given the limited fitness data available in each task at test time. While some meta-learning methods combine in-context learning and fine-tuning [Rusu et al., 2018, Vuorio et al., 2019], they do so by using computationally expensive higher-order gradients to account

for fine-tuning during meta-learning. Crucially, *Metalic* leverages in-context meta-learning, and then subsequent fine-tuning, without accounting for fine-tuning during meta-learning. This novel combination is particularly computationally efficient, and enables *Metalic* to outperform more complicated methods for meta-learning.

We present *Metalic*, an in-context meta-learning approach to tackle the problem of protein fitness prediction in low-data settings, and make the following contributions:

- We introduce a method that efficiently combines in-context meta-learning with PLMs and fine-tuning for protein fitness prediction.
- We advance state-of-the-art (SOTA) for zero-shot protein fitness prediction on the ProteinGym benchmark [Notin et al., 2024].
- We attain strong performance for few-shot fitness prediction with 18 times fewer parameters.
- We ablate each component of our method to demonstrate the contributions of each part and underscore their necessity.
- We empirically validate the superiority of our method to alternative forms of meta-learning.

7.2 Related Work

Meta-Supervised Learning Meta-learning aims to create a sample-efficient learning algorithm by training over a distribution of tasks. The goal is to learn algorithms such that they can rapidly adapt to new tasks during inference. In the supervised setting, each task is defined by a dataset, rather than by an MDP. This inference-time adaptation is often called the *inner loop*, for which there are two primary forms found in the literature: gradient-based meta-learning [Finn et al., 2017, Zintgraf et al., 2019]

and in-context (or black-box) meta-learning [Mishra et al., 2017, Nguyen and Grover, 2022].

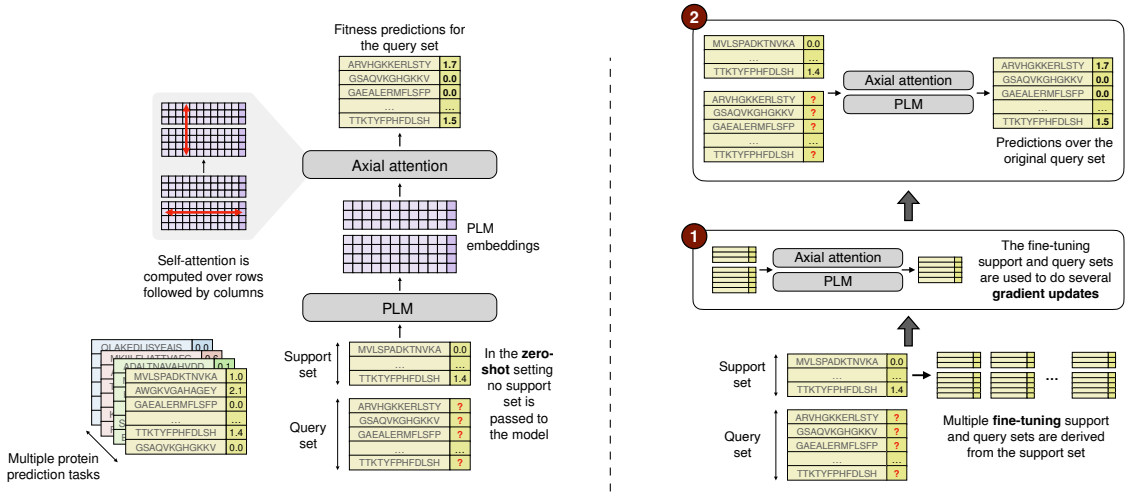
Gradient-based and in-context algorithms differ both in their computational efficiency and capacity for out-of-distribution generalization. Gradient-based approaches explicitly adapt model parameters within in the inner loop using standard gradient-based learning. Commonly, a parameter initialization is learned that can be adapted to new tasks with only a few gradient steps [Finn et al., 2017, Zintgraf et al., 2019, Vuorio et al., 2019]. However, this comes with considerable computational overhead – due to meta-gradients when differentiating the inner loop – which makes gradient-based meta-learning less suitable for large models. Alternatively, in-context meta-learning adapts by conditioning a sequence model on a task-specific dataset in context. These methods condition on the data points over which gradient-based approaches would train [Santoro et al., 2016, Mishra et al., 2017, Beck et al., 2024a, 2023b]. Such methods are typically more sample- and compute-efficient than gradient-based methods, but perform worse on out-of-distribution tasks given the lack of explicit gradient-based learning in the inner loop [Beck et al., 2023a]. Rather than training for in-context learning, the in-context learning of pre-trained large language models can also be used to perform meta-learning [Coda-Forno et al., 2023]; however, this lacks generalization guarantees and requires fitting all tasks in context simultaneously, which is not possible for the size of our data (Section 7.4.1).

In *Metalic*, we only train for in-context meta-learning, but find this is still compatible with task-specific fine-tuning. In the meta-reinforcement learning setting, this combination has been shown to be possible by increasing task-specific data for fine-tuning [Xiong et al., 2021]. In contrast, we evaluate in the supervised setting and do not give increased data at inference time. While prior work has combined gradient-based and in-context meta-learning [Rusu et al., 2018, Vuorio et al., 2019], these works

compute expensive meta-gradients to learn how to account for fine-tuning. Despite not explicitly meta-learning gradient-based adaptation, we find in-context meta-learning alone provides a strong foundation for subsequent fine-tuning and that both aspects are critical for high performance.

Likelihood-Based Fitness Prediction with PLMs Leveraging pre-trained PLMs is standard practice in protein fitness prediction [Rives et al., 2021, Notin et al., 2023, Rao et al., 2021, Truong Jr and Bepler, 2024]. In the few-shot setting, PLMs intended for sequence generation are repurposed by fine-tuning for protein fitness prediction [Rives et al., 2021]. In the zero-shot setting, it is assumed that the fitness correlates with the likelihood of the proteins associated sequence of amino acids, as predicted by a PLM [Meier et al., 2021, Truong Jr and Bepler, 2024]. Furthermore, if using a masked PLM, it is often assumed that each amino acid contributes independently to the fitness [Meier et al., 2021]. In this work, we likewise leverage PLMs for protein fitness prediction. However, in contrast, we make use of additional data in the form of additional fitness prediction tasks on other proteins. Specifically, we meta-learn how to use a PLM for protein fitness prediction, rather than relying on assumptions. Only after meta-learning, do we fine-tune our model, as depicted in Figure 7.1. Meta-learning across tasks lets us avoid restrictive model constraints and achieve SOTA performance. We will show that in-context meta-learning is necessary to achieve strong results in low-data settings.

In-Context PLMs We build upon existing PLMs that make use of in-context data for protein fitness prediction [Notin et al., 2022, Truong Jr and Bepler, 2024, Notin et al., 2023, Rao et al., 2021]. However, these methods do not *meta-learn* how to make use of their context. These methods either learn to use the context only for protein language modelling, and then assume that the likelihood from the generative model correlates with fitness [Truong Jr and Bepler, 2024, Notin et al., 2022, Rao et al.,



(a) Meta-training **(b)** Fine-tuning

Figure 7.2: An overview of *Metalic*. Meta-training our model over many protein prediction tasks enables in-context learning (a). Fine-tuning the in-context learning on the support set requires sub-sampling smaller support and query sets, and enables generalization at test time (b).

2021], or they use the context for protein fitness prediction, but not by meta-learning over protein tasks [Notin et al., 2023]. Of these ProteinNPT [Notin et al., 2023] is the most related to our method, since we use the same attention architecture to condition on fitness information about related proteins in-context, and it uses gradient steps to fine-tune to the target task. In comparison, our method meta-learns over many tasks how to make use of the fitness information, which we find to be critical (Section 7.4). Additionally, our method is the first to use the aforementioned procedure to allow fine-tuning and in-context conditioning on the very same context at inference time.

7.3 Methods

7.3.1 Problem Setting

We consider a fitness prediction task, $\mathcal{T}$, to be defined by a dataset of the form $\mathcal{D}_{\mathcal{T}} = \{(x_i, y_i \equiv f_{\mathcal{T}}(x_i))\}_{i=1}^{N}$, where x_i is a sequence of amino acids, and $y_i \in \mathbb{R}$ is the associated scalar fitness value assigned by the (unknown) underlying fitness function

$f_{\mathcal{T}}$. In the few-shot setting, the task-specific data is typically split into non-overlapping support and query sets: $\mathcal{D}_{\mathcal{T}}^{(\mathrm{S})} = \{(x_i^{(\mathrm{S})}, y_i^{(\mathrm{S})})\}_{i=1}^{N^{(\mathrm{S})}}$ and $\mathcal{D}_{\mathcal{T}}^{(\mathrm{Q})} = \{(x_i^{(\mathrm{Q})}, y_i^{(\mathrm{Q})})\}_{i=1}^{N^{(\mathrm{Q})}}$ such that $\mathcal{D}_{\mathcal{T}}^{(\mathrm{S})} \cap \mathcal{D}_{\mathcal{T}}^{(\mathrm{Q})} = \emptyset$, with support and query set sizes, $N^{(\mathrm{S})}$ and $N^{(\mathrm{Q})}$, respectively. The support set provides data for task-specific adaptation, and the query set provides data for evaluating the adapted performance. The size of the support set is called the *shot*. Note, even with an empty support set (*zero-shot*), the model can still adapt to the task using information from the query set, i.e., related sequences without fitness scores.

Meta-learning for protein fitness prediction requires not just a single task, but multiple tasks, $\mathcal{D} = \mathcal{D}_1, \ldots \mathcal{D}_{\mathrm{T}}$ over which to learn. The full dataset of tasks, $\mathcal{D}$, can be seen as defining a distribution of tasks which can be split into training and test tasks in the usual way. Concretely, for meta-learning in-context, the goal is to learn a function with parameters θ conditioned on the full support set and unlabelled inputs from query set, $f_\theta(\{x_i^{(\mathrm{Q})}\}_{i=1}^{N^{(\mathrm{Q})}}, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})})$. In our case, rather than directly predicting fitness values, we instead follow prior works that use a preference-based objective to rank the query set in order of fitness [Krause et al., 2022, Brookes et al., 2023, Hawkins-Hooker et al., 2024].

7.3.2 Metalic

Architecture Our work leverages the ProteinNPT architecture proposed by [Notin et al., 2023] as an in-context PLM for fitness prediction. Here we briefly summarize the key elements, but defer the reader to the appendix and the original paper for full details. The architecture, along with the data we use for meta-leaning, are illustrated in Figure 7.2a. A detailed illustration is give in the appendix.

Protein sequences in both the support and query set are converted to per-residue embeddings using a pre-trained PLM; in our case we take the third layer of ESM2-8M [Lin et al., 2022]. Fitness scores in the support set are projected to match the

dimensionality of the residue embeddings using a linear layer, while the query set fitness embeddings share a single learned embedding. The protein sequence embeddings and fitness embeddings are then concatenated along the sequence dimension. Optionally, zero-shot fitness predictions from an auxiliary PLM can be embedded and concatenated in the same way, which we explore in Section 7.4.3. This tensor is then processed via axial attention blocks [Ho et al., 2019], each of which applies self-attention separately along and across the sequences. Axial attention reduces the computational complexity of self-attention from $O(K^2L^2)$ to $O(K^2 + L^2)$, where K is the shot and L is the length of a protein. Finally, a multi-layer perceptron conditions on the fitness embedding and mean-pooled sequence embedding to predict each query value, i.e., $\{v_i^{(\mathrm{Q})}\}_{i=1}^{N^{(\mathrm{Q})}} = f_\theta(\{x_i^{(\mathrm{Q})}\}_{i=1}^{N^{(\mathrm{Q})}}, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})})$, which is then used to rank the proteins by fitness.

Meta-Training Following prior works that use a preference-based objective [Krause et al., 2022, Brookes et al., 2023, Hawkins-Hooker et al., 2024], we reframe the relative score prediction of two sequences as binary classification, predicting whether sequence $x_i^{(\mathrm{Q})}$ has a higher fitness than $x_j^{(\mathrm{Q})}$:

$$p\left(y_i^{(\mathrm{Q})} > y_j^{(\mathrm{Q})}\right) = \sigma\left(v_i^{(\mathrm{Q})} - v_j^{(\mathrm{Q})}\right), \tag{7.1}$$

where σ is a sigmoid function and the dependency of the query values on θ has been dropped for brevity. This classifier is optimized with respect to every pairwise comparison between sequences in the query set corresponding to the loss function:

$$\mathcal{L}(\theta, \mathcal{D}_{\mathcal{T}}^{(\mathrm{Q})}, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})}) = -\sum_{i=1}^{N^{(\mathrm{Q})}} \sum_{\substack{j=1 \\ j \neq i}}^{N^{(\mathrm{Q})}} \mathbb{I}\left(y_i^{(\mathrm{Q})} > y_j^{(\mathrm{Q})}\right) \log \sigma\left(v_i^{(\mathrm{Q})} - v_j^{(\mathrm{Q})}\right), \tag{7.2}$$

where $\mathbb{I}$ is an indicator function. Intuitively, this is optimising $N^{(\mathrm{Q})} \times (N^{(\mathrm{Q})} - 1)$ binary classification problems. Note that we only compute the loss over the query set to avoid encouraging memorization of the support set. Adapting this to meta-learning (Figure 7.2a), the objective becomes to find the parameterization that minimizes the

loss across the task distribution,

$$\mathcal{J}(\theta, \mathcal{D}) = -\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\mathbb{E}_{(\mathcal{D}_{\mathcal{T}}^{(S)}, \mathcal{D}_{\mathcal{T}}^{(Q)})\sim p(\mathcal{D}_{\mathcal{T}}^{(S)}, \mathcal{D}_{\mathcal{T}}^{(Q)}|\mathcal{D}_{\mathcal{T}})}\mathcal{L}(\theta, \mathcal{D}_{\mathcal{T}}^{(\mathrm{Q})}, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})}), \tag{7.3}$$

where $p(\mathcal{T})$ is a uniform distribution over tasks, and $(\mathcal{D}_{\mathcal{T}}^{(S)}, \mathcal{D}_{\mathcal{T}}^{(Q)})$ are sampled uniformly and without replacement as disjoint subsets of $\mathcal{D}_{\mathcal{T}}$.

Fine-Tuning *Metalic* uses fine-tuning, during inference, in order to enable generalization, without having to account for fine-tuning during meta-training. This process is depicted in Figure 7.2b.

The combination of in-context learning and fine-tuning creates a unique problem. Since fine-tuning occurs at inference time, labels for the query set, $\{y_i^{(\mathrm{Q})}\}_{i=1}^{N^{(\mathrm{Q})}}$, are not available for training. While labels for the support set, $\{y_i^{(\mathrm{S})}\}_{i=1}^{N^{(\mathrm{S})}}$, are available, propagating gradients from the support set would encourage memorization of the support set, since the labels are also passed as input in-context. While prior methods that combine in-context and gradient-based meta-learning [Rusu et al., 2018, Vuorio et al., 2019] would encounter this issue, this problem is exacerbated for *Metalic*. Whereas prior methods compress inputs to a representation with a constant number of dimensions, *Metalic* uses self-attention, which scales with the number of inputs, allowing them to be stored without compression. Moreover, whereas prior methods use meta-gradients that could adjust the gradient update procedure so as to be useful for generalization and not memorization, *Metalic* does not take into account the fine-tuning process during meta-training.

We address the issue of memorization by sub-sampling from the support set. The fine-tuning procedure is the same as during meta-training, with the exception that the support set is sub-sampled. In order to compute updates on a single support set, the support set is sub-sampled into multiple smaller support and query sets, $\mathcal{D}_{\mathcal{T}}^{(\mathrm{S}')} \subseteq \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})}$ and $\mathcal{D}_{\mathcal{T}}^{(\mathrm{Q}')} \subseteq \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})}$. Concretely, this corresponds to fine-tuning on unseen data using

the objective:

$$\mathcal{J}(\theta, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})}) = -\mathbb{E}_{(\mathcal{D}_{\mathcal{T}}^{(S')}, \mathcal{D}_{\mathcal{T}}^{(Q')}) \sim p(\mathcal{D}_{\mathcal{T}}^{(S')}, \mathcal{D}_{\mathcal{T}}^{(Q')} | \mathcal{D}_{\mathcal{T}}^{(\mathrm{S})})} \mathcal{L}(\theta, \mathcal{D}_{\mathcal{T}}^{(\mathrm{Q}')}, \mathcal{D}_{\mathcal{T}}^{(\mathrm{S}')}), \tag{7.4}$$

where $(\mathcal{D}_{\mathcal{T}}^{(S')}, \mathcal{D}_{\mathcal{T}}^{(Q')})$ are sampled uniformly and without replacement as disjoint subsets of $\mathcal{D}_{\mathcal{T}}^{(S)}$. After fine-tuning, *Metalic* conditions on the complete support set, allowing no data to go to waste.

Using *Metalic*'s unique combination of in-context meta-learning followed by fine-tuning, we enable the generalization of extensive fine-tuning, while also precluding expensive computation. If a typical gradient-based meta-learning method requires $O(m)$ meta-gradients and $O(mn)$ regular gradients for meta-training, *Metalic* requires no meta-gradients and $O(m)$ regular gradients, constituting a linear reduction with superior performance to efficient alternatives, as demonstrated in Section 7.4.6.

7.4 Experiments

In this section, we evaluate *Metalic* on fitness prediction tasks from the ProteinGym benchmark [Notin et al., 2024]. We evaluate in the zero-shot setting with no support data, and the few-shot setting with limited support data. To establish SOTA results in the zero-shot setting, we first compare to the predictions provided by ProteinGym for the baseline models. To establish strong performance in the few-shot setting, since predictions are not provided, we train baselines from Hawkins-Hooker et al. [2024]. While *Metalic* does not achieve SOTA results in evaluations on proteins that have multiple mutations (multi-mutant proteins), we demonstrate that the performance grows as we add more meta-training tasks, providing a path forward for *Metalic* in the multi-mutant setting in the future. We perform ablations of *Metalic*, to show the benefits of meta-learning, in-context learning, and fine-tuning. Finally, we compare to the gradient-based method, Reptile [Nichol et al., 2018], to show that taking account of gradients during training is an unnecessary computational burden.

7.4.1 Experimental Setup

In our experiments, we focus on ProteinGym deep mutational scans. Each task in ProteinGym each measures one property on a set of proteins that all differ by one amino acid, or multiple amino acids, from a reference wild-type protein. We have 121 single-mutant tasks and 68 multi-mutant tasks from ProteinGym. From these, we evaluate over eight held-out single-mutant tasks, and five held-out multi-mutant tasks, following Notin et al. [2023], Hawkins-Hooker et al. [2024]. This leaves 113 single-mutant and 68 multi-mutant tasks for training when evaluating single mutants (Sections 7.4.2 and 7.4.3), and 121 single-mutant and 63 multi-mutant tasks for training when evaluating multiple mutants (Section 7.4.4). All fitness values are standardized by subtracting the mean and dividing by the standard deviation by task. Note that there are additional tasks in ProteinGym we do not consider. Specifically, we do not consider multi-mutant tasks that have overlapping proteins with single-mutant tasks, to make evaluation more difficult. We also ignore additional tasks in which the maximum protein length is > 750, to fit the backward pass on an Nvidia A100-80Gb device.

We use a query set size of $N^{(\mathrm{Q})} = 100$, and the size of the support set is determined by our evaluation setting and is one of three sizes: $N^{(\mathrm{S})} = 0$, 16, or 128. We also use an additional set of 128 points just for early stopping of the fine-tuning process, for all models, following the implementation of Hawkins-Hooker et al. [2024]. We then evaluate remaining points in the task, with a maximum of 2,000 points total, by dividing the data into multiple query sets. If the model can fit a larger query size, as in non-meta-learning baselines, then we pass the remaining data as a single query set. If the data is not divisible by the query set size, it is left out from evaluation. However, we sample the support data over three independent samples, avoiding systemic exclusion.

All evaluation uses the Spearman rank correlation, in line with prior work [Notin et al.,

2023, Truong Jr and Bepler, 2024, Hawkins-Hooker et al., 2024]. We compute the Spearman correlation per task, and then average over tasks. For all evaluations of our models, we compare over three seeds for training and report the mean and standard deviation. Each context, consisting of a support and query set, consists of $\leq$ 171,000 tokens. We meta-train for 50,000 updates and fine-tune for 100. Fine-tuning uses the same procedure, including an Adam optimizer and cosine learning rate scheduler, but fine-tuning skips the learning rate warm-up, so the scheduling has little effect. Using a single Nvidia A100-80Gb, training our model takes roughly 2 to 8 days per seed, depending on support size and frequency of fine-tuned evaluation.

7.4.2 Zero-Shot

The first setting we evaluate is the zero-shot performance of our model, with no support set for fine-tuning. In Table 7.1 we compare against predictions provided by ProteinGym for each baseline to compute the zero-shot Spearman correlation (ρ). We compare to provided predictions, on our data splits, to enable a fair comparison to the strongest models available without retraining each baseline from scratch ourselves. We include the best performing model, and notable models, as baselines. We find that *Metalic* outperforms every reported baseline and is SOTA at zero-shot prediction.

Our method significantly outperforms strong baselines with many more parameters, such as ESM1-v-650M. The 8 million parameter PLM used by our method, ESM2-8M, without *Metalic*, achieves a score of only .121 ρ, demonstrating the large contribution of our meta-learning procedure. The next strongest method after ours is VESPA [Marquet et al., 2022]. VESPA optimizes PLM embeddings to predict a distinct set of binary annotations from 9,594 proteins, and uses these features to predict protein fitness by comparing to a reference wild-type embedding. Note, unlike our method, VESPA relies on strong assumptions to generalize and is specific to the zero-shot setting.

Table 7.1: Spearman correlation in the zero-shot setting. Results are computed using predictions provided by ProteinGym. Using a single PLM, in the zero-shot setting, renders the baselines deterministic. For comparison, we report our best model, and the mean and standard deviation, for quantifying the variation in meta-training. *Metalic* achieves SOTA performance in either case.

Model Name	Spearman Correlation
Metalic (max)	**.484**
Metalic (mean)	**.482** ± .002
VESPA	.464
TranceptEVE-Medium	.457
ESM1-v-650M	.437
Tranception-Medium	.427
Progen2-Medium	.419
ESM2-650M	.399
MSA Transformer	.398
ESM-IF1	.365
ESM2-8M	.121

The strong performance of our method in the zero-shot setting can be attributed to meta-learning. Since there is no data for fine-tuning, the zero-shot performance increase over ESM2-8M derives entirely from our meta-training procedure. In this case, other methods generally assume that the PLM likelihood of a mutation correlates with fitness, whereas our model learns to make use of the information contained in PLM embeddings to make predictions zero-shot. Moreover, our method still conditions on an unlabeled query set, and the protein embeddings in that query set, which allow for meta-learning a form of in-context unsupervised adaptation.

7.4.3 Fine-Tuning Results

In Table 7.2 we report Spearman correlation with a support set of size $N^{(\mathrm{S})} = 0$, 16, and 128, and we compare to baselines that we train and evaluate ourselves over three random seeds. We re-train these methods using the models, following Hawkins-Hooker et al. [2024], to provide a comparison over multiple seeds between these methods in a range of practical low-data settings. All models use the same preference-based loss function as *Metalic*, for a fair comparison, and none use ensembling.

Table 7.2: Spearman correlation for the 0, 16, and 128-shot setting. Baseline results are re-computed. Standard deviation is provided over three seeds. *Metalic* matches or exceeds all baselines.

Model Name	$n = 0$	$n = 16$	$n = 128$
Metalic	**.482** ± .002	**.484** ± .001	.552 ± .009
Metalic-AuxIF	**.498** ± .008	**.500** ± .002	.556 ± .005
ESM1-v-650M	.384 ± .000	.452 ± .000	.553 ± .000
ESM2-8M	.105 ± .000	.226 ± .000	.406 ± .000
PoET	.416 ± .003	.475 ± .026	**.588** ± .006
ProteinNPT	N/A	.192 ± .003	.443 ± .003

We also train *Metalic*-AuxIF, which allows *Metalic* to condition on zero-shot predictions from an additional PLM, as introduced in Section 7.3.2. We choose ESM-IF1 [Hsu et al., 2022] as the auxiliary input, since it contains embeddings derived from inverse folding that summarize protein structure. Note that *Metalic*-IF1 uses 170 million parameters, whereas *Metalic* uses 36 million (including the ESM2-8M embedding), so the auxiliary predictions significantly increases the total parameters.

Again, we find that *Metalic* has the strongest performance in the 0-shot and 16-shot settings, and has comparably strong performance in the 128-shot setting, with 18 times fewer parameters. Moreover, *Metalic* also outperforms contemporary models, such as PoET [Truong Jr and Bepler, 2024], that make use of additional evolutionary information, in the form of multi-sequence alignment, and in-context learning. Note that ESM2-8M and ProteinNPT are the worst performing methods in the few-shot setting. This results suggests that the effectiveness of *Metalic* does not come from the ESM2-8M embedding, nor the ProteinNPT architecture, but rather from the meta-learning itself.

Additionally, we see that *Metalic*-AuxIF, improves results. This result is significant because ESM2-8M and ESM-IF1, the two PLMs used by *Metalic*-AuxIF, are the worst performing methods in the zero-shot setting (Table 7.1), with ESM2-8M also weak in the few-shot setting. Thus, the effectiveness of *Metalic*-AuxIF comes entirely from

meta-learning: We can learn how and when to rely on features from each of these weak predictors of protein fitness, to combine them into a strong predictor.

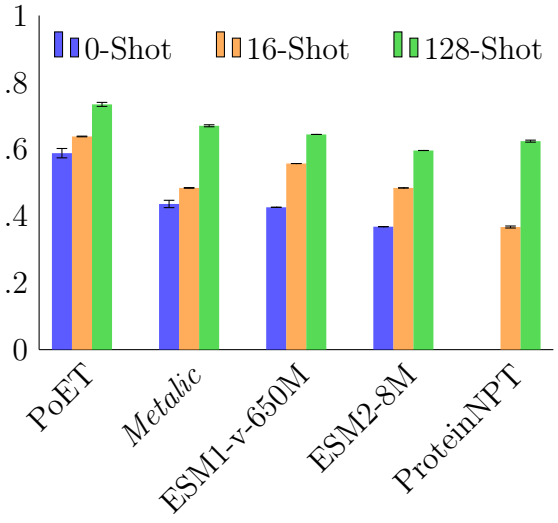
Consistent with the motivation of meta-learning, results are strongest when the data is most limited. Meta-learning adds an additional training stage to learn prior beliefs and inductive biases from related data. The more limited, the more relying on prior data is useful.

7.4.4 Multiple Mutants

Table 7.3a: Multi-mutant Spearman correlation in the zero-shot setting. Results are computed using predictions provided by ProteinGym on tasks with multiple mutations. *Metalic* is competitive, but not better than all baselines, due to lack of sufficient datasets with multiple mutants.

	Multiples
Model Name	0-Shot
Metalic (max)	.450
Metalic (mean)	.436 ± .011
Metalic-AuxIF (max)	.548
Metalic-AuxIF (mean)	.533 ± .012
ESM-IF1	**.590**
TranceptEVE-Medium	.529
Tranception-Medium	.513
MSA Transformer	.503
VESPA	.408
ESM2-650M	.345
Progen2-Medium	.305
ESM2-8M	.289
ESM1-v-650M	.279

Figure 7.3b: Multi-mutant Spearman Correlation by Shot. Spearman correlation for the 0, 16, and 128-shot, over three seeds, sorted by zero-shot performance. For complete results, including for *Metalic*-AuxIF, see the appendix.

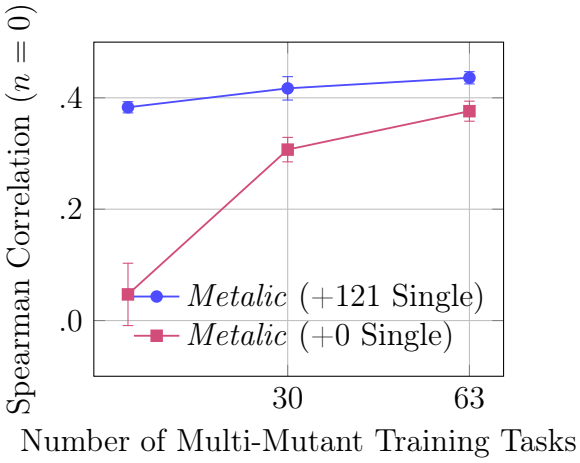


In this section we evaluate *Metalic* on tasks where proteins have multiple mutations. In Table 7.3a, we see that *Metalic* has strong performance, but does not outperform all baselines. However, with auxiliary inverse folding predictions (*Metalic*-AuxIF), *Metalic* is only outperformed by ESM-IF1 itself. These results indicate that ESM-IF1

Table 7.4a: Multi-mutant Spearman correlation by training data. Results are computed on eight tasks with multiple mutations with different amounts of single- and multi-mutant training tasks. We see performance of *Metalic* increases with more data, giving a path to improve results with future data collection. This trends holds even when we add single-mutant data, which is significantly different from the multi-mutant test data.

Single- Mutant	Multi- Mutant	Multiples $n = 0$
121	63	**.436 ± .011**
121	30	.417 ± .021
121	1	.383 ± .010
0	63	.376 ± .018
0	30	.307 ± .022
0	1	.047 ± .056

Figure 7.4b: Multi-mutant spearman correlation by number of tasks. *Metalic* with and without 121 additional single-mutant training tasks. (With 121 single-mutant is the default.) The performance of *Metalic* increases with more data, giving a path to improve results.



is particularly strong on the eight held-out evaluation tasks, and that we can recover most of its performance by incorporating its predictions into our model. Note that while ESM-IF1 is strong in the multi-mutant setting, it is among the worst performing models in the single-mutant setting. In contrast, our method is strong in both settings, since it can learn to leverage ESM-IF1 predictions when they are helpful, and ignore them when they are not.

By comparing across all shot settings, we can see that *Metalic* is competitive in the multi-mutant setting (Figure 7.3b), but no longer SOTA, which we hypothesize is due to limited multi-mutant training data. In Table 7.4a and Figure 7.4b, we explore this hypothesis. Specifically, we see the performance of *Metalic* increases as the amount of training data, measured in tasks, increases. This trend holds true even when adding single mutant tasks, which are significantly different from the testing data. Providing more data is a path forward for strengthening the multi-mutant results.

Table 7.5: Ablations in the 0 and 128-shot setting. Results show the importance of fine-tuning, in-context learning, meta-training, and additional training tasks as an augmentation.

Model Name	$n = 0$	$n = 128$
Metalic	**.482** ± .002	**.552** ± .009
Metalic-NoFT	**.482** ± .002	.488 ± .012
Metalic-NoPref	.465 ± .011	.520 ± .006
Metalic-NoICL	.441 ± .002	.529 ± .002
Metalic-NoMetaTrain	-.046 ± .018	.346 ± .003

7.4.5 Ablations

Here we report ablations of *Metalic*, evaluating on single-mutants (Table 7.5). Spearman correlation is reported in the zero-shot and 128-shot settings. The results justify all components of our method.

Most detrimental to performance is removing meta-training from *Metalic* (NoMeta-Train). This ablation is identical to *Metalic*, with the exception that there is no additional meta-learning stage over multiple protein landscapes. We see that without meta-learning, the initial zero-shot predictions have near zero correlation with the fitness and that the 128-shot predictions are critically impaired.

We also ablate the ability to attend to the rest of the proteins in context by turning off the column attention in the axial attention layers (NoICL), we ablate the fine-tuning stage of training (NoFT), and we ablate the preference loss by using mean squared error instead (NoPref). Note that when we remove the fine-tuning, and have a non-zero support size, we allow the early stopping data to be passed in-context, to not unfairly advantage fine-tuning with additional data. Ablating in-context learning decreases performance in both settings, indicating an ability to adapt in an unsupervised fashion even from the query set alone. Ablating fine-tuning decreased performance in the 128-shot setting, indicating the need for fine-tuning for generalization to held-out data. Ablating the preference-based loss decreased performance in both settings, in line

Table 7.6: Comparison to Reptile. Reptile [Nichol et al., 2018] is a gradient-based meta-learning method. Results are reported after 50,000 steps, with *Metalic* additionally reported after 150,000 steps, to allow for an equal number of gradient computations as the methods using Reptile. Results show that accounting for fine-tuning during meta-training using Reptile is unnecessary.

Model Name	Meta-Training Steps	Total Gradient Computations	$n = 128$
Metalic (150k)	150,000	150,000	**.562** ± .004
Metalic-Reptile	50,000	150,000	**.562** ± .004
Metalic (50k)	50,000	50,000	.552 ± .009
Reptile-3-100	50,000	150,000	.539 ± .001
Reptile-3-3	50,000	150,000	.499 ± .008

with recent literature [Krause et al., 2022, Brookes et al., 2023, Hawkins-Hooker et al., 2024]. An additional ablation is given in the appendix.

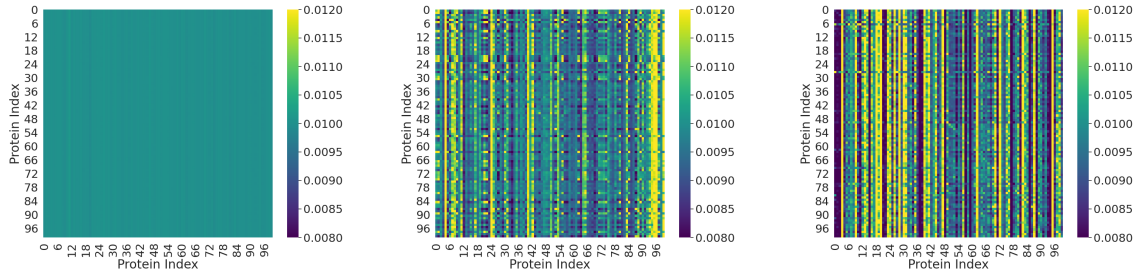
7.4.6 Gradient-Based Meta-Learning

In this section, we compare *Metalic* to the same architecture but trained with Reptile [Nichol et al., 2018], an efficient method for gradient-based meta-learning. Unlike *Metalic*, Reptile does not use in-context learning and modifies the outer-loop during meta-training to take into account the subsequent fine-tuning. While accounting for the fine-tuning during meta-training comes with increased computational costs and can increase bias and variance [Vuorio et al., 2021], Reptile provides a simplified algorithm that can be run more efficiently. The primary differences between Reptile and *Metalic* are that Reptile adjusts the meta-learning loss to account for fine-tuning during meta-training, while *Metalic* performs in-context learning. Reptile details are provided in the appendix.

Although Reptile is more compute-efficient than other gradient-based methods, it still performs gradient updates in the inner loop during meta-training, making it computationally expensive relative to *Metalic*. Our method uses 100 updates on the support data for fine-tuning. Reptile uses inner-loop gradient updates during both meta-training and fine-tuning. Due to compute limitations, we cannot use 100 steps

for each forward pass of meta-training. Thus, we evaluate Reptile with 3 inner-loop gradient steps during meta-training and 3 during fine-tuning, so that train and test match (Reptile-3-3), and we evaluate Reptile with 3 inner-loop gradient steps during meta-training and 100 during fine-tuning, so that test time matches our method (Reptile-3-100). Note that even Reptile-3-3 uses three times more compute than *Metalic*. Finally, we evaluate Reptile-3-100 with *Metalic*, to see whether they can be used in conjunction (*Metalic*-Reptile). We train for 50,000 steps, and also report *Metalic* after 150,000 steps, to match the number of gradient computations as the Reptile models.

Results are reported in Table 7.6. We evaluate in the 128-shot setting. Note that Reptile is not applicable in the zero-shot setting, as it requires some data for fine-tuning. We observe that without controlling for the amount of computation, *Metalic* (50k) outperforms both Reptile variants. We also observe that *Metalic*, in conjunction with Reptile (*Metalic*-Reptile), outperforms Reptile and *Metalic* independently. However, the improvement over *Metalic* is marginal, and the increased computation (by a factor of 3) is large. Controlling for the total number of gradient computations, *Metalic* (150k) achieves a Spearman correlation comparable to the combined *Metalic*-Reptile. Thus, Reptile performs poorly in isolation, and, while Reptile can be used with *Metalic*, the combination increases implementation complexity with no clear advantage. This result is in line with previous work showing that accounting for gradient updates during meta-training (in their case, without in-context learning) can be detrimental when there are a limited number of tasks for meta-training or a limited amount of data for the inner-loop [Gao and Sener, 2020, Triantafillou et al., 2020]. Collectively, this evidence further justifies fine-tuning only after meta-training.



(a) Untrained Attention **(b)** Half-Trained Attention **(c)** Trained Attention

Figure 7.5: Attention maps. We present axial attention maps over the query set in the zero-shot setting. Each row shows attention to other proteins in context, normalized to one by row, averaged over layers and mutation location. Attention at step 1 (a) 25,0000 (b) and 50,000 (c). Each protein learns to pay attention to itself, while still attending to other significant proteins in context.

7.5 Analysis

Here, we briefly investigate the in-context learning abilities of *Metalic* via attention maps. In Section 7.4, we show that in-context learning is vital to *Metalic* by ablating the attention between proteins and showing decreased performance. Notably, the in-context learning was beneficial not only in the few-shot setting, but also in the zero-shot setting. This suggests an interesting phenomenon: The emergence of unsupervised in-context learning from the query set alone. To confirm this phenomenon, in Figure 7.5 we show the attention maps in the axial attention layers between proteins in the query set. Over the course of training, we observe the emergence of bright vertical lines, indicating some significantly informative proteins to which all others proteins attend. Moreover, we observe no rows that are entirely dark off the diagonal entries. Thus, no protein attends only to itself. Together, these results further corroborate the necessity of in-context meta-learning.

7.6 Conclusion

In this chapter, we have demonstrated state-of-the-art results on a standard protein fitness prediction benchmark in low-data settings. To do so, we proposed *Metalic*, which

makes use of both in-context meta-learning and subsequent fine-tuning. Critically, we have demonstrated the ability of meta-learning to take advantage of additional data from other proteins fitness prediction tasks, while remaining computationally tractable by deferring fine-tuning to test time alone. Unique within the meta-learning literature, we show that in-context meta-learning provides a useful initialization for further fine-tuning, and can make use of test time data for both fine-tuning and in-context learning. *Metalic* additionally demonstrates the ability to learn from the query set alone (zero-shot), performing unsupervised in-context learning.

While this chapter only addresses supervised protein fitness prediction as a sub-problem of protein design, future work could leverage our pre-trained model to transfer the problem setting back to that of meta-RL. While beyond the scope of this work, *Metalic* could be leveraged as a world model in the meta-reinforcement learning setting, for training a policy to optimally trade off exploration and exploitation when designing novel proteins. Given its efficacy at leveraging additional data, we believe that meta-learning in-context will play a crucial role in advancing protein fitness prediction, with *Metalic* serving as a foundational step in that direction.

7.7 Statement of Authorship

Chapter 7 is based on content from **Metalic: Meta-Learning In-Context with Protein Language Models** [Beck et al., 2024b, 2025a].

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the lead author on this manuscript. I was primarily responsible for the ideation, implementation, experimentation, and writing. Co-authors assisted, particularly with experimentation, figures, and editing.

Citation:

Jacob Beck, Shikha Surana, Manus McAuliffe, Oliver Bent, Thomas D. Barrett, Juan Jose Garau Luis, and Paul Duckworth. "Metalic: Meta-Learning In-Context with Protein Language Models." *The Thirteenth International Conference on Learning Representations*, 2025.

8

Conclusion

Contents

8.1 Summary

While standard reinforcement learning algorithms have shown remarkable success at solving a wide array of problems, these data-hungry algorithms are a far cry from the rapid learning and adaptation humans are able to perform when encountering unknown environments. In this thesis, we have explored the field of meta-reinforcement learning, as a solution to enable rapid learning in our artificial agents. In meta-RL, an agent directly learns how to learn, over a distribution of different MDPs, each of which constitute a reinforcement learning problem. To ensure that our resultant learning algorithms are capable of rapid adaptation, we have specifically focused on the zero-shot problem setting, requiring the greatest speed of learning.

Still, we have found that learning such an agent, over the task distribution required for meta-training, presents its own challenges. Specifically, we discussed how training a single policy over all tasks is generally more difficult than training individual networks for each task independently. Inspired by the field of multi-task RL, in Chapter 4, we propose the use of hypernetworks, so that individual networks can be produced for each task when needed, but parameters can also be shared between the tasks when possible. Hypernetworks not only extended the idea of training individual networks, but also enabled that solution to be applicable to the field of meta-RL, where the task is unknown.

While evaluating the use of hypernetworks, we found that they presented their own difficulties. Specifically, we saw that hypernetworks were highly sensitive to their initialization scheme. While standard initialization methods maintain statistical properties of the layers of the hypernetwork, the properties do not hold at initialization in the base networks produced by the hypernetwork. To address this challenge, we propose a novel initialization method. The novel initialization method adjusts only the final layer in the hypernetwork, in order to re-produce any given initialization for the base network at the start of meta-training.

Using hypernetworks with proper initialization, we saw drastic improvements. We saw that our initialization matched or exceeded the performance of existing initialization methods for hypernetworks, while also being easier to extend and implement. We also evaluated hypernetworks, regardless of initialization method, in meta-RL for the first time, demonstrating their significant advantages. Moreover, we noted that when used in meta-RL, the hypernetworks with proper initialization not only performed better than conventional networks, but also enabled better scaling in terms of the overall number of parameters, when evaluated with contemporary task-inference methods for meta-RL.

In Chapter 5, we moved beyond task inference methods and evaluated hypernetworks with an end-to-end objective. This thesis is the first to evaluate hypernetworks that can flexibly modify base networks in meta-RL as an arbitrary function of history. As a first step, we improved the task-inference methods, by initializing the hypernetwork with parameters trained in the multi-task setting, and training the task-inference module itself over the multi-task data as well. Enabled by the use of hypernetworks, we additionally improved task-inference methods by training the task-inference module to infer the hypernetwork weights directly (see the appendix). Still, after all of these improvements, we found hypernetworks trained on the simpler end-to-end objective to match or exceed all alternatives.

In Chapter 6, we investigated the sequence models used to summarize history in meta-RL. While we found end-to-end objectives to achieve better performance relative to task-inference objectives, it still remained unknown whether end-to-end hypernetworks could benefit from the inductive biases built into the meta-parameterizations of standard task-inference methods. Specifically, we investigated the use of sequence models that are permutation-invariant with respect to historical transitions, since this inductive bias is theoretically sufficient for representing the optimal policy and can ameliorate gradient decay. Surprisingly, despite theoretical guarantees, we find that introducing controlled permutation variance improves architectural robustness and allows for the representation of sub-optimal policies, which can serve as stepping stones to the optimal solution.

Finally, in Chapter 7, we examined the applicable concerns that arise from meta-learning in practice. Specifically, we investigated meta-learning in the context of protein fitness prediction and validated an approach combining pre-trained language models, end-to-end in-context learning, and fine-tuning. Our method demonstrated significant improvements from meta-learning, while remaining computationally tractable by

deferring fine-tuning to test time alone. Moreover, we showed that in-context meta-learning provides a useful initialization for further fine-tuning, and can make use of test-time data for both fine-tuning and in-context learning.

From application, to hypernetworks and initialization, to supervision, to sequence modeling and permutation invariance, we covered a vast array of representations in meta-RL and contributed a significant number of methodological insights. By integrating these insights, we offer a path forward for building RL agents capable of both rapid adaptation and broad generalization, advancing a long-standing goal in the field of AI.

8.2 Open Problem: The *Problem* Problem

The meta-training task distribution plays an important role in meta-RL, as it determines *what* inductive bias we can learn from data, while the meta-RL algorithms determine *how well* we can learn this inductive bias. Meta-RL can learn to collect informative data in the inner-loop, solving the exploration-exploitation trade-off discussed in 2.3. However, the same dilemma still exists in the outer-loop for meta-exploration, and exploration is constrained by the task distribution available for meta-training. The task distribution should be both diverse enough so that the learned inductive bias can generalize to a wide range of new tasks, and dense enough in the task space so as to provide sufficient data to support learning an adaptation algorithm. However, existing few-shot meta-RL methods are frequently meta-trained on narrow task distributions, where different tasks are simply defined by varying a few parameters that specify the reward function or environment dynamics [Duan et al., 2016, Finn et al., 2017, Rakelly et al., 2019, Zintgraf et al., 2020], or they are meta-trained on fewer than 100 (highly diverse) tasks [Yu et al., 2020].

This problem is largely due to insufficiencies with the benchmarks themselves. For

example, some of benchmarks with the widest task distributions are private [Team et al., 2023], or not designed to test adaptation in meta-RL [Küttler et al., 2020]. Moreover, some of these benchmarks simply need to be more densely populated with tasks in order for few-shot adaptation to be learnable. For example, Meta-World [Yu et al., 2020], used in this thesis, has benchmarks with 1, 10, or 45 discrete tasks; however, meta-learning an algorithm that can reliably adapt to unseen meta-test tasks may require tens of thousands of meta-training tasks [Kirsch et al., 2022]. Consequently, it is clear that we need to design benchmarks that are diverse, in addition to having a densely populated task distribution, to support learning agents capable of such broad generalization.

One path forward for future benchmarks may be the use of automated environment design. Notably, methods for automatic environment design have begun to be used in meta-RL [Team et al., 2023], and large pre-trained models have begun to automatically both suggest new tasks [Wang et al., 2023] and provide measures of success for given tasks [Wang et al., 2024, Beck, 2025]. Whether manually or automatically designed, new benchmarks are needed to better reflect the complex task distribution in real-world problems, and promote the design of new methods that can generalize over these broad and challenging tasks.

8.3 Closing Remarks

Meta-learning is an extremely general and powerful paradigm. We have already seen meta-learning applied to protein fitness prediction in this thesis. The broad applicability of meta-RL, specifically, can be seen in sub-fields as disparate robotic locomotion to multi-agent RL. In robotics, the rapid adaptation learned by meta-RL agents in simulation can enable them to quickly overcome novel terrains when deployed in the real world [Song et al., 2020, Kumar et al., 2021]. In multi-agent RL, by viewing other agents as (part of) the task, meta-RL is directly applicable [Zintgraf et al.,

2021a, ab Abebe Tessera et al., 2024], with meta-RL even being used to provide best-response policies for computing Stackelberg equilibria [Gerstgrasser and Parkes, 2022], a concept itself applied in areas as significant as autonomous vehicles [Cooper et al., 2019].

Beyond broad application, meta-RL offers a promising path for future AI systems. Since meta-RL scales with data and computation, it is a method likely to survive and continue to scale with ever increasing computational capabilities. By trading engineering effort and computation at deployment for computation during meta-training, meta-RL is making a calculated bet on the future of hardware for automation. This bet is in line with Rich Sutton's "bitter lesson" [Sutton, 2019], and is one that has historically paid off.

Finally, no matter the methodological path, learning is a necessary feature for, and a hallmark of, capable autonomous agents. Thus, learning to learn, i.e., meta-learning, will be a phenomenon of any future capable and general agent. For this reason, insights from meta-RL should assist practitioners in developing and reasoning about such systems. We hope that this thesis, by its improvements to the field of meta-RL, contributes to such agents, enabling AI systems truly capable of both broad generalization and rapid adaptation.

8.4 Statement of Authorship

Some content in Chapter 8 is based on **A Tutorial on Meta-Reinforcement Learning** [Beck et al., 2023a, 2025b].

Content from Beck et al. [2023a, 2025b] is additionally included in the thesis: Vuorio, Risto. "Advances in Meta-Reinforcement Learning and Imitation Learning." PhD diss., University of Oxford, 2024.

Publication Status:

- Not Submitted
- Accepted for Publication
- Published ✓

Contribution:

I was the co-lead author on this manuscript. I was responsible for the writing, structure, and surveying, along with the other co-lead. I had a particular responsibility for content related to few-shot and zero-shot meta-RL, whereas the other co-lead focused on many-shot methods, not germane to this thesis. I additionally reviewed, edited, and re-wrote all content as needed.

Citation:

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. "A Tutorial on Meta-Reinforcement Learning." *Foundations and Trends in Machine Learning*, 2025.

Appendices

A

Appendix for Chapter 4, Initialization

Contents

A.1 Benchmark Details

We evaluate on grid-world [Zintgraf et al., 2021b], MuJoCo [Todorov et al., 2012], and Meta-World [Yu et al., 2020]. For Meta-World[1], we use version two of ML10 and version one of Pick-Place (ML1). We use gridworld and MuJoCo environments from the reference VariBAD implementation[2]. The number of episodes per meta-episode are the same as in this implementation. Returns reported are summed across episodes in the meta-episode.

[1] `https://github.com/rlworkgroup/metaworld`
[2] `https://github.com/lmzintgraf/varibad`

A.2 Model Implementation

Network Architecture. For all experiments, we use a three-layer MLP network to represent the base network, either learning the parameters directly or generating them with a hypernetwork. Both of these networks are conditioned on a task embedding of size 10 in all experiments. A range of architecture sizes were explored on Cheetah-Dir and Walker MuJoCo tasks, which are detailed in Table A.1.

Table A.1: Policy (base) network architectures.

Size	**Layer widths**
XS	(64, 64, 32)
S	(128, 64, 64)
M	(128, 128, 64)
L	(256, 128, 128)
XL (*Default*)	(256, 256, 128)
XXL	(1024, 512, 512)

Actor and Critic Networks. In the standard architecture, we use separate actor and critic networks. In the hypernetwork model, there are separate heads on the hypernetwork for the actor and critic parameters. Since all reported results use linear hypernetworks, this is equivalent to using separate hypernetworks for the actor and critic, conditioned on the same task embedding.

Choice of Base Network Initialization. Our proposed methods, Bias-HyperInit and Weight-HyperInit, require a given initialization method for the base network, which we denote f. We chose normc initialization [Dhariwal et al., 2017] for f, as it is the default initialization in the reference VariBAD implementation. Note that, as in VariBAD, the state and task encoders use Kaiming initialization with a uniform distribution [He et al., 2015] (or orthogonal initialization in the RNN), and the head of the critic (produced by the hypernetwork) also uses Kaiming. Additionally, the head of the actor has a gain of 0.01 for categorical distributions (i.e., in gridworld), and 1

for continuous distributions (as is standard for linear layers), as is default in VariBAD. However, as this default was noticed after experiments, the actor and critic heads of both HyperInit methods have a gain of $\sqrt{2}$, which is the default for ReLU activations.

A.3 Hyperparameter Tuning

Hyperparameters are identical to those in the reference VariBAD implementation, other than the learning rate and network architecture.

For Cheetah-Dir and Walker, the models were tuned over the following set of learning rates on three seeds: $\{3e{-}3, 1e{-}3, 3e{-}4, 1e{-}4, 3e{-}5\}$. Learning rates of $1e{-}3$ and $1e{-}4$ yielded the best performance for standard and hypernetwork models respectively, over both domains. For the remainder of the MuJoCo tasks and grid-world, the learning rate was tuned over a narrower range of learning rates. These ranges were $\{3e{-}3, 1e{-}3, 3e{-}4\}$ for the standard model and $\{3e{-}4, 1e{-}4, 3e{-}5\}$ for the hypernetwork. Again, we found that $1e{-}3$ and $1e{-}4$ yielded the best performance across domains for the standard and hypernetwork models respectively. Given this, we used these two tuned learning rates for further evaluation on Pick-Place. On ML10, we again evaluate over the full range: $\{3e{-}3, 1e{-}3, 3e{-}4, 1e{-}4, 3e{-}5\}$.

The network sizes were tuned on Cheetah-Dir and Walker as well. The standard and hypernetwork models were turned over all base network sizes in Table A.1, with the exception of the XXL base model, which was included only to demonstrate the performance of the standard model with an equal number of model parameters as the XL hypernetwork architecture. For both hypernetworks and the standard network, the XL size achieved the highest return (or not significantly different from the best) on both tasks. We then used XL for the rest of the MuJoCo tasks and for Pick-Place and ML10. Gridworld was tuned separately. For gridworld, we evaluated over M, L, and XL for the standard architecture and XS, S, and M for the hypernetwork

architecture. All models solved the domain and performed similarly. The results reported on gridworld are therefore from the smallest evaluated architecture sizes, M and XS respectively.

Additionally, on Cheetah-Dir and Walker we evaluated two-layer (rather than linear) hypernetworks, with a hidden layer of width 16 or 32. (Note that the input size, i.e., the output from the task encoder, was kept constant at 10.) In both cases, for L and XL base network sizes, adding an additional layer decreased or did not affect return.

A.4 Learning Curves

Figure A.1 presents learning curves for all remaining domains and models. These results are summarized in the main body, but are reported here for completeness. Note that reported metrics are computed over the last 1% of data for all curves, other than ML10, which uses the last 10% of data.

A.5 Meta-World Training Performance

The Pick-Place and ML10 benchmarks have distinct train and test tasks. While we present and discuss test performance in the main report, Table A.2 details the train performance of the standard architecture and hypernetwork on these benchmarks. For Pick-Place, we observe similar train and test performance. However, we found that the ML10 train performance differed from test performance.

Table A.2: Meta-train success percentage on Meta-World benchmarks.

Method	**Pick-Place**	**ML10**
Standard	4.4 ± 2.1	33.8 ± 3.0
HFI	$\mathbf{24.3 \pm 14.9}$	36.0 ± 2.3
Bias-HyperInit	$\mathbf{45.3 \pm 17.3}$	32.0 ± 0.4

At meta-train time, the standard architecture and hypernetwork with Bias-HyperInit

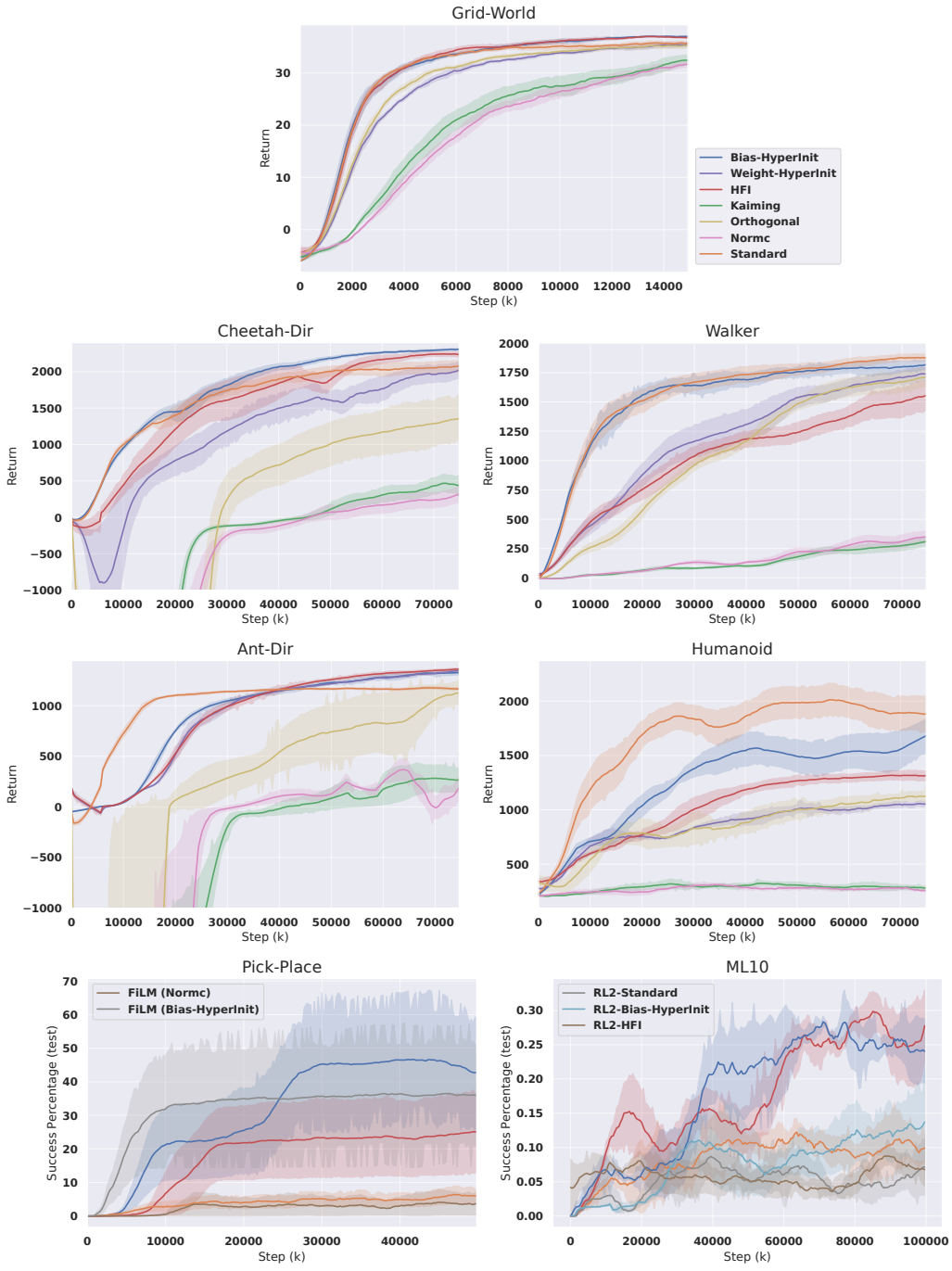


Figure A.1: Learning curves for all remaining domains. (68% confidence interval for all plots.)

achieve nearly identical success rates on ML10. However, hypernetworks with Bias-HyperInit and HFI achieve significantly greater success rates on test tasks (see main report), suggesting that hypernetworks have improved generalization compared to

standard models, which overfit to the train tasks.

A.6 Application to FiLM

When applied to FiLM [Perez et al., 2018], the Bias-HyperInit method becomes:

1. Sample parameters $\phi \sim f$, set $W = 0$, and set b to produce the bias parameters in ϕ (as in Bias-HyperInit),
2. Set the weights of base network to those in ϕ (as required by FiLM),
3. Set the remaining elements of b to 1, thereby leaving the weight distribution unchanged.

Note that Weight-HyperInit can also be adapted such that W produces distinct bias parameters for each task and scalars of 1 for all tasks, but we do not test this since Bias-HyperInit outperforms Weight-HyperInit.

B

Appendix for Chapter 5, Supervision

Contents

B.1 Additional Task-Inference Ablations

In order to select task-inference methods for ultimate comparison in the main body, we evaluated those methods, in addition to further task-inference ablations, on the Grid-World and Walker environments, and then selected the best performing methods. From these results, we chose TI++HN (on Grid-World), and TI and TI Naive (on Walker). See Figure B.1. We additionally chose to evaluate VI+HN in the main body, as it is a prior strong method [Beck et al., 2022]. A table summarizing the components in all methods can be seen in Table B.1. The additional ablations are depicted in Figure B.2, with details given below:

VI. This baseline, called VariBAD, is proposed in Zintgraf et al. [2020] and is the same as VI+HN in the main body, but without the hypernetwork.

TI++. This baseline is the same as TI++HN in the main body, but without the hypernetwork. The two additions over TI (denoted "++") are re-use of the multi-task policy parameters as an initialization for the meta-RL policy, and training of task inference over trajectories from pre-training.

TI+HN. This baseline is the same as TI in the main body, but with the addition of a hypernetwork.

BI++HN. Here, the base-network for the task is inferred (denoted "BI"), instead of the task representation directly. This method is novel and is the same as TI++HN in the main body, but task inference is trained to reconstruct the parameters for the base-network produced by the multi-task hypernet, ϕ', rather than the task representation ($c_{\mathcal{M}}$ or g):

$$\hat{\phi}' = P^{\phi'}(z \sim IB)$$
$$J_{infer}(\theta) = \mathbb{E}_{\mathcal{M}}[\mathbb{E}_{\tau|\pi}[-||\phi' - \hat{\phi}'||_2^2]].$$

Note that while the performance of BI++HN is similar to that of TI++HN on Grid-World, TI++HN was chosen as a representative baseline since it is more common to existing methods and BI++HN requires significantly more compute.

B.2 Hyperparameter Tuning

We tune each baseline over five learning rates for the policy, [3e-3, 1e-3, 3e-4, 1e-4, 3e-5], for three seeds each. We use a learning rate for the task inference modules of 0.001, both since varying the learning rate seemed to have little effect (Figure B.3)

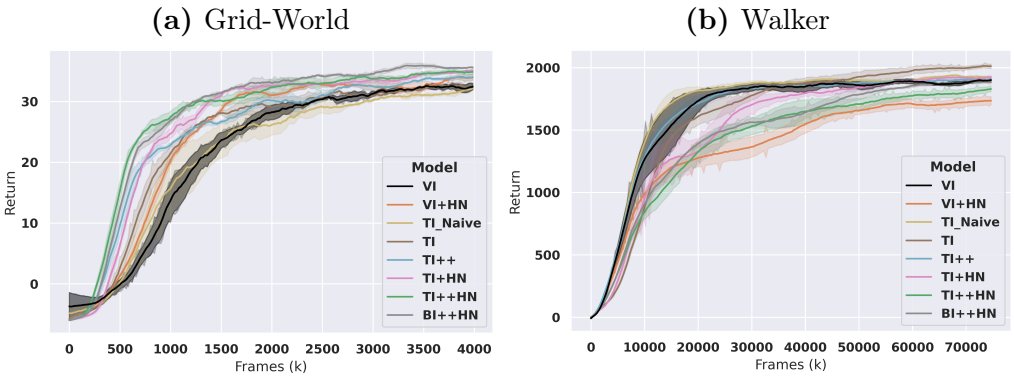


Figure B.1: TI++HN was chosen as a baseline given results from Grid-World (a). TI and TI Naive were chosen as baselines given results from Walker (b).

	Inference Target	Policy Conditions on State	Hypernetwork	Inference Training in Multi-Task Phase and Parameter Reuse
RNN	None	No	No	N/A
RNN+S	None	Yes	No	N/A
RNN+HN	None	Yes	Yes	N/A
TI Naive	Given	Yes	No	N/A
TI	Learned	Yes	No	No
TI++	Learned	Yes	No	Yes
TI+HN	Learned	Yes	Yes	No
TI++HN	Learned	Yes	Yes	Yes
VI	Transitions	Yes	No	N/A
VI+HN	Transitions	Yes	Yes	N/A
BI++HN	Base Net	Yes	Yes	Yes

Table B.1: Summary of the components in each method

and to provide a fair comparison in terms of computation. Error bars report a 68% confidence interval using bootstrapping.

Before being passed to the hypernetwork or to the policy, depending on the model, the task (c or g) are projected down to size 25, as are the concatenation of (μ, σ). This size was chosen to be in the range of the values used by [Zintgraf et al., 2020] across environments, but a single value was chosen to be consistent. Additionally, 25 is sufficiently large for a one-hot representation of all discrete task distributions

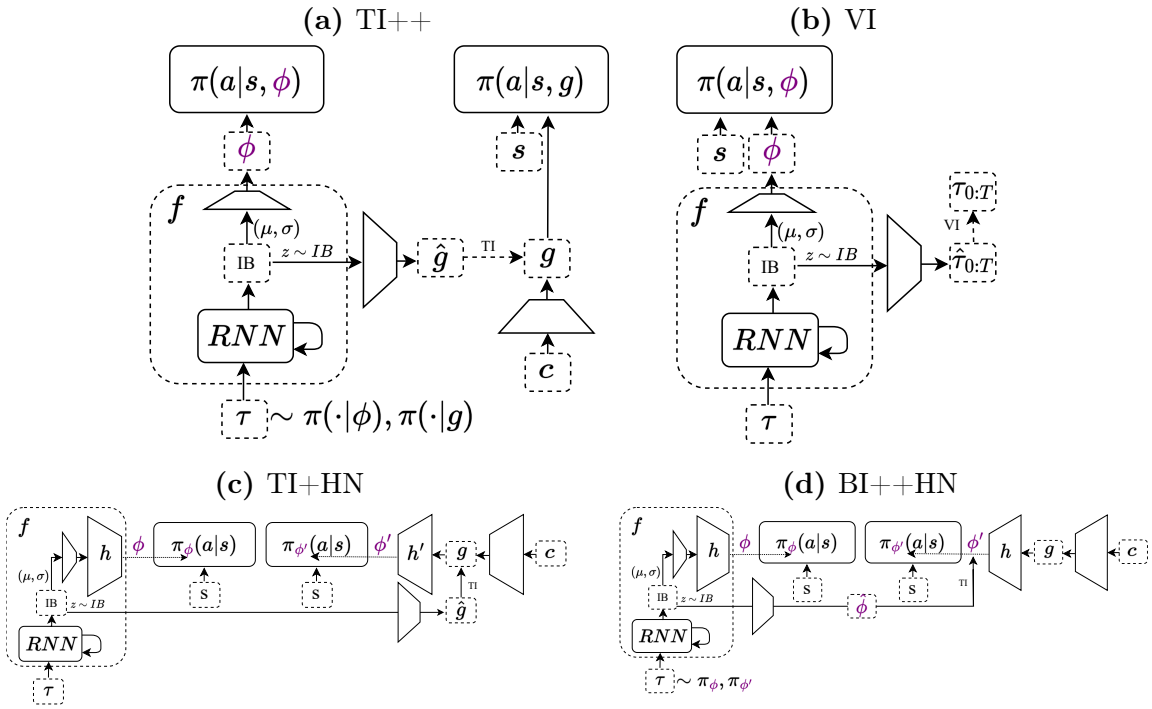


Figure B.2: Additional Task-inference baselines.

used. However, on Ant-Dir, the projection size is 10. For the state embedding size (passed to the policy) we chose 256 and for the sizes of the MLP policy we chose 256, followed by 128. This corresponds to the "XL" model size in [Beck et al., 2022], which was optimal or near optimal where reported. However, for the comparison of RNN, RNN+S, and RNN+HN on Grid-World, these results are reported with a state embedding size for the policy tuned over [5, 25, 125, 625]. For the state embedding size passed to the trajectory encoder, we used 32 for all experiments, as the default value in [Zintgraf et al., 2020]. And for all RNNs, we use a single GRU layer of size 256, as in Beck et al. [2022].

As discussed in the main body, pre-training of the multi-task policy occurs at the expense of the meta-RL policy training time, in order to ensure the same number of samples for training each method. While selecting the time for pre-training does add a hyperparameter that must be selected, we tune this once on Grid-World and then transfer to all other environments, in order to afford as little additional computation as

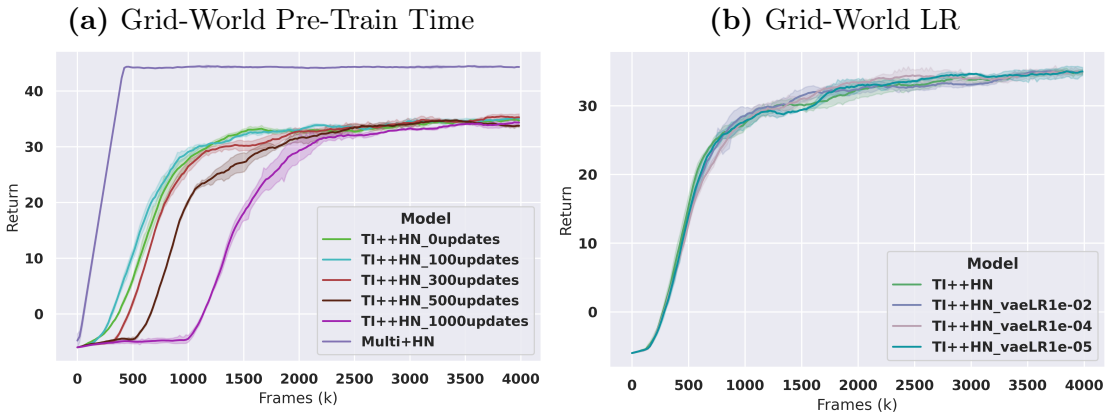


Figure B.3: Training for 100 updates yielded optimal performance despite not fully training the multi-task policy, Multi+HN (a). Varying the learning rate for task inference module had little effect (b).

possible to these baselines. Note that while this cutoff affords some extra computation to task inference methods, the recurrent hypernetwork method (RNN+HN), which we show to be the strongest, does not require any such additional tuning.

On Grid-World, we find 100 PPO updates for training the multi-task policy to be optimal. In this environment, each update consists of 960 frames. Interestingly, while the multi-task policy (Multi+HN) requires approximately 500 updates to fully train, training for 100 yielded optimal performance for TI++HN. (We saw similar results for TI, not reported here, except that the random projection provided by 0 updates already gave similar performance to 100.) Note that 100 updates is equivalent to only 20% of the total frames for training the multi-task policy, and it is only 2.4% of the total frames for training the meta-RL policy. This suggests that very little amount of computation is required to get a good task representation, and delaying the start of meta-training significantly is generally not worthwhile. For MuJoCo experiments in the main body, we use 563 updates, since that is the same percent of total frames (2.4%) for Walker and Ant-Dir. This constituted 4.8% of total frames on Cheetah-Vel and Cheetah-Dir, which transferred well on initial experiments.

For all other hyperparameters, we default to those in [Beck et al., 2022].

B.3 End-to-End and Task Inference Together

While it is possible to combine the end-to-end objective and task inference objective, prior work suggests that interference from competing objectives may be detrimental [Humplik et al., 2019]. Moreover, separating the training objectives can be practically useful, as the combined objectives may require tuning of the relative weighting of the two objectives. Still, we perform investigatory experiments on Grid-World. See Figure B.4. We find that adding task-inference objectives to the end-to-end meta-RL objective decreased return compared to the end-to-end objective alone (RNN+HN) for every method tested. The same effect can be seen regardless of the architecture (RNN+HN or RNN+S). Unsurprisingly, the performance of the combined end-to-end and task-inference methods falls in between each type of method used independently. Additionally, tuning the relative objective weighting had little effect. We experiment with giving 10% (BI10p, TI10p) and 50% (BI50p) weighting to task inference methods, but still observing the same decrease in return. In all these experiments, we use the RNN architecture from the end-to-end RNN methods, meaning that we did not use an information bottleneck, nor the linear encoding layer. In other words, the output of the RNN passed to the policy, ϕ, was directly used as $\hat{g}$ or $\hat{c}$ and compared to the respective label g or c.

B.4 Gridworld Details

Grid-World consists of a five by five grid with a goal location in one of the grid cells. The agent starts in the bottom left corner of the grid, and then has 15 steps in an episode to find the goal and remain there. The agent receives 1 reward for each timestep at the goal and -0.1 reward for all other timesteps. The goal location is held constant for each meta-episode, which consists of 4 regular episodes. In Grid-World Show, the goal position is visible to the agent at the first timestep of each episode. In

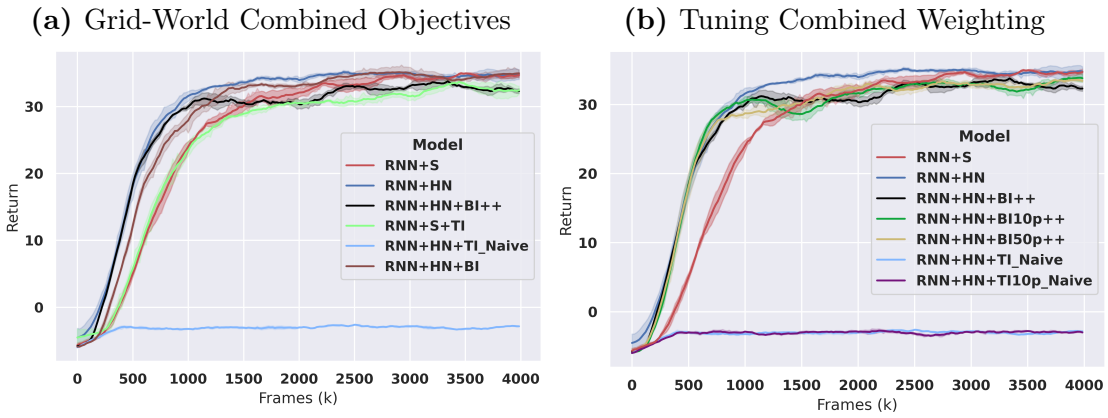


Figure B.4: Adding task inference objectives in combination with the end-to-end meta-RL objective decreases performance relative to end-to-end alone, and results in performance between the two types of methods (a). Varying the relative objective weighting had little effect (b).

Grid-World Dense, the agent receives and observes a reward equal to the Manhattan distance to the goal location.

B.5 MuJoCo Details

We evaluate on all four MuJoCo [Todorov et al., 2012] environments used by Zintgraf et al. [2020]. All environments require legged locomotion. All episodes terminate after 200 timesteps. Meta-episodes in Walker consist of two regular episodes, whereas meta-episodes in the rest consist of one regular episode.

In Cheetah-Dir, the agent controls a robotic cheetah morphology by outputting a control torque for each of six joints on the morphology. The task is to run forward or backward with as large a velocity as possible. The agent observes the position, angle, and velocity of each body part (17 dimensions in total) and is given a positive reward for its velocity in the direction given by the task and a negative reward for control costs (specifically, 5% of the magnitude of the action vector).

In Cheetah-Vel, the agent controls the same robot, but instead of the the positive reward component for running on the selected direction, the agent receives a negative

reward component equal to the absolute difference to a target velocity, which is sampled uniformly from [0,3].

In And-Dir, the agent controls a robotic ant morphology by outputting a control torque for each of eight joints on the morphology. The task is to run forward or backward, with as large a velocity as possible. The observations are 27 dimensional in this case, and the agent is given a positive reward for its velocity in the direction given by the task and a negative reward for control costs (specifically, 5% of the magnitude of the action vector). Additionally, the agent is given a negative contact reward (.05% of the magnitude of the external forces on each joint) and a positive survival reward (1 per timestep). Episodes in this environment additionally terminate if the body falls outside of a predefined height.

In Walker, the agent controls a two-legged torso morphology and outputs a control torque for each of six joints on the morphology. The observations are 17 dimensional for this environment. The tasks are defined as uniform samples of 65 different physics coefficients (e.g. body mass and friction) for the simulation. The agent is given a positive reward for the forward velocity, a positive reward of one reward per timestep, and a negative reward for the control costs (specifically, 0.1% of the magnitude of the action vector). Episodes in this environment additionally terminate if the body falls outside of a predefined height or rotation range.

B.6 Compute

Experiments were run on four to eight machines simultaneously each with eight GPUs, ranging from NVIDIA GeForce GTX 1080Ti to NVIDIA RTX A5000. Each model was tuned over five learning rates with three seeds per learning rate. Each experiment took between 4 hours for grid-worlds and 3 days for MuJoCo experiments. In total, there were approximately 360 experiments for MuJoCo and 270 for grid-worlds in the

main results. With about 5 experiments on average per GPU, this constitutes roughly 5,400 GPU hours.

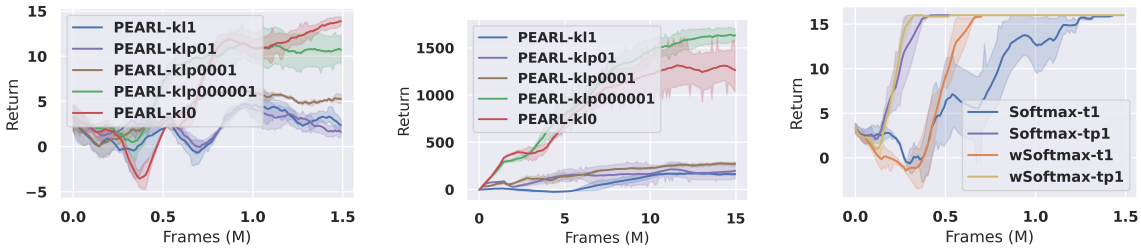
C

Appendix for Chapter 6, Sequence Modeling

Contents

C.1 Hyperparameter Tuning

We tune each baseline over five learning rates for the policy, [3e-3, 1e-3, 3e-4, 1e-4, 3e-5], for three seeds. (Results are reported with a 68% confidence interval, computed through bootstrapping with 1,000 iterations across three seeds, consistent with all plots presented.) Of the models evaluated, PEARL, and aggregators with the softmax aggregation, each require an additional hyperparameter. PEARL require a weight



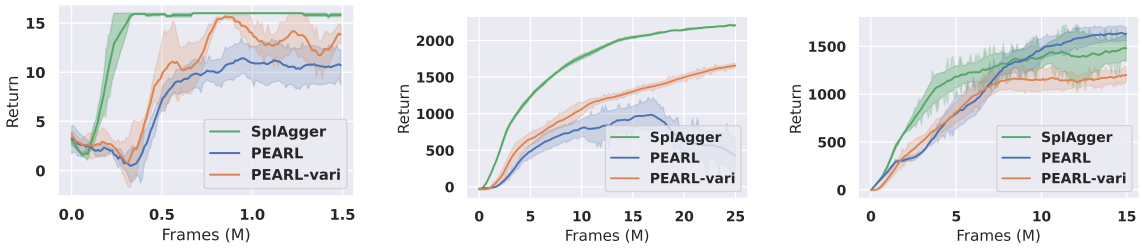
(a) T-LS **(b)** Walker **(c)** T-LS

Figure C.1: Tuning the KL-divergence weight on PEARL, in C.1a and C.1b, and tuning the softmax temperatures in Figure C.1c. For PEARL, the chosen weight, 1^{-6}, achieved the highest returns. Using a weight of 0 achieved a similar final returns, suggesting the stochastic latent was not particularly useful in our experiments. We chose 1^{-6} as it achieved the greatest average return throughout training, on both environments. For the softmax aggregators, 0.1 was the best initialization for the temperature.

on objective penalizing the KL-divergence to the prior. We tune this weight on the T-LS and Walker environments, with a weight of 1^{-6} performing the best. In Figures C.1a and C.1b, we display these results. Additionally, for softmax aggregators, we tuned the initial temperature of the softmax function. The temperature is learnable, however changes very slowly, and so is sensitive to initialization. This temperature was tuned between 1.0 and 0.1 on T-LS, with 0.1 performing the best. Note that we tuned the aggregators without using an RNN or SplAgger. This is depicted in Figure C.1c.

For all other hyperparameters, we default to those in Beck et al. [2023b], with two exceptions. First, on the Planning Game, we it necessary to set the exploration bonus in the objective to zero in order to learn any systematic exploration. Second, Beck et al. [2023b] projected the output of the RNN down to size 10 or 32, and then to size 10 or 25, depending on the environment, using a linear layer, before being passed to the hypernetwork. For consistency, we leave this size as 24 and 25 for all models and all experiments.

C.2 PEARL Design Choices and Additional Experiments



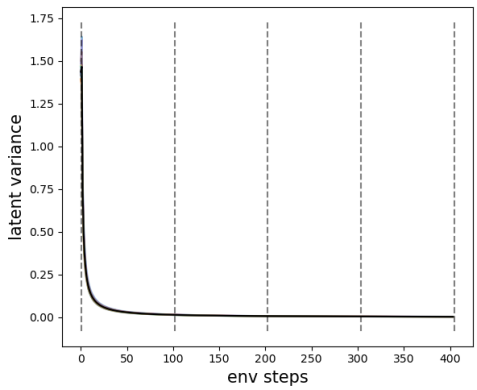
(a) T-LS **(b)** Cheetah-Dir **(c)** Walker

Figure C.2: PEARL using the VariBad-stlye supervized reconstruction from Zintgraf et al. [2021b] to train the sequence model (PEARL-vari). PEARL-vari, increases returns slightly on T-LS and Cheetah-Dir, but remains lower than SplAgger.

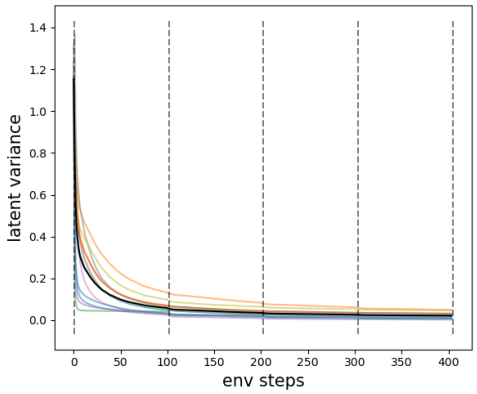
In addition to the experiments with PEARL in the main body, we present a preliminary findings here. Out of all baselines, PEARL is the only method that requires a stochastic latent variable. In the original paper, the stochastic latent variable is sampled once per episode. However, this type of exploration requires multiple episodes for exploration, and will necessarily fail on multiple of our benchmarks. Instead, we consider two alternatives. In the experiments presented in the main body of the thesis, we sample the stochastic latent at every state, and here, we additionally present experiments that use the self-supervision provided by reconstructing rewards and transitions to train the stochastic latent, as in Zintgraf et al. [2021b]. We use the default hyperparameters presented by Zintgraf et al. [2021b]. While the results here are incomplete, and the model is still far inferior to SplAgger, we did find some improvement, as shown in Figure C.2.

C.3 Analysis of PEARL Aggregation Failure

Here, we investigate why PEARL's aggregation performs worse than SplAgger, and hypothesize that the poor performance has due to with modelling assumptions regarding the variance of its latent variable. Unlike other methods, PEARL's aggregation



(a) PEARL Latent Variance (Initialization)



(b) PEARL Latent Variance (Trained)

Figure C.3: In C.3a, we see that the variance of PEARL decreases rapidly to zero at initialization. In C.3b, we see the same phenomenon, but when PEARL is fully trained.

method requires the use of a stochastic latent variable. We observe that the variance of the aggregator decreases rapidly over time. In Figure C.3a and Figure C.3b, we can see that both at initialization and after training, the variance decreases rapidly to zero. Looking at the probability density function (PDF), we can see why.

Specifically, PEARL uses a product of Gaussians,

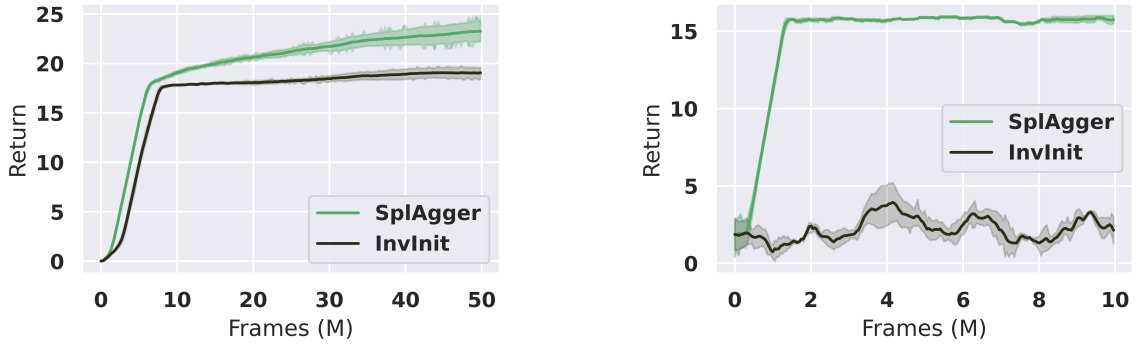
$$q_\theta(z|\tau) \propto \prod_{t=1}^{t=T} \mathcal{N}(z; \mu_t(\tau_t), diag(\sigma_t^2(\tau_t))),$$

which has a closed form for the joint mean and covariance. From Bayesian conjugate analysis, we know that the product of Gaussian PDFs is itself a Gaussian [Murphy, 2007]. The new Gaussian, $q_\theta(z|\tau) = \mathcal{N}(z; \mu', diag(\sigma'))$, has a new mean, μ', which itself is a weighted average of each individual μ_t. The respective weights are $1/\sigma_t^2$, and the variance is $\frac{1}{1/\sigma_0^2+...+1/\sigma_T^2}$. Since the denominator in the variance of PEARL grows monotonically, the variance must drop monotonically. In fact, assuming the variances are approximately equivalent at initialization, then the variance should decrease like $1/t$.

The fact that the variances decreases over time could create problems for learning.

For example, if the agent were to collect contradictory evidence about the latent, then the variance of the true posterior would increase. Well our domains do not present contradictory evidence, many transitions in our domains are uninformative, and so the sequence model would still need to learn to counteract the decreasing variance, which may hinder learning. Future work could experiment with removing the latent variable from PEARL, which would be equivalent to the weighted average aggregator presented in Section C.6, or producing the variance of the latent variable using a different permutation invariant function. Note that this pathology only applies to the aggregation method used in PEARL, and may not be the most important factor to consider in the off-policy setting in which the PEARL method originally evaluates [Rakelly et al., 2019].

C.4 LSTM Invariant Initialization



(a) Planning Game **(b)** T-Maze Agreement

Figure C.4: Here we evaluate an LSTM model initialized to be permutation invariant. The returns are far lower than SplAgger.

Finally, we experiment with using a sequence model that is initialized to be permutation invariant, but that can easily learn to be permutation variant. In order to do this, we use a long short-term memory unit (LSTM) [Hochreiter and Schmidhuber, 1997], and adjust the initialization of the gates. Specifically, we adjust the gates to compute a summation, before being normalized for the output. First, we set the weights of

the input and forget gates to zero. Then, we adjust the bias of the input and forget gates to outputs of $1 - \epsilon$, for $\epsilon = 0.0001$. This forces all inputs to be fully added to the running sum and not forgotten. We additionally we set any weight connected emanating from the recurrent state, in the cell and output gates, to zero. This forces there to be do dependence on past states. We use an LSTM instead of a GRU so that the input gate is not forced to be the compliment of the forget gate. Results were not encouraging, and are depicted in Figure C.4.

C.5 Transformer Results

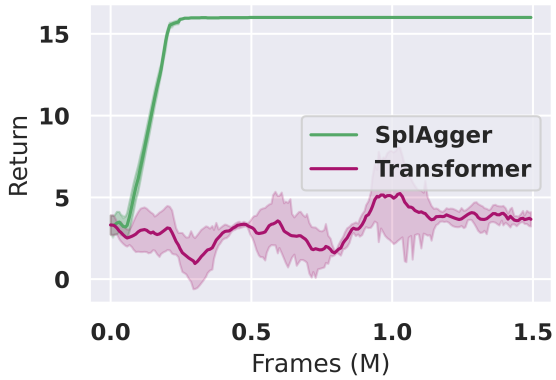


Figure C.5: The transformer model fails to learn and is significantly outperformed by SplAgger on the T-LS domain.

As discussed in the main body, attention is inherently permutation invariant. Thus, transformers without positional encodings are an obvious fit for this problem setting. However, Attention is computationally expensive: whereas both commutative aggregation and recurrent networks use $O(1)$ memory and compute per timestep, attention generally requires $O(t^2)$ memory and compute per timestep t. We therefore consider it appropriate to limit our solutions to constant memory and compute, in line with sequence models designed to quickly handle long contexts [Garnelo et al., 2018b, Beck et al., 2020]. Still, the runtime was not entirely prohibitive on one experiment: T-LS. Thus, we did run a Transformer model on T-LS, with results depicted in Figure C.5.

The results show that the transformer did not learn in the allotted number of frames, and was significantly outperformed by SplAgger.

C.6 Additional Weighted Average Aggregation Results

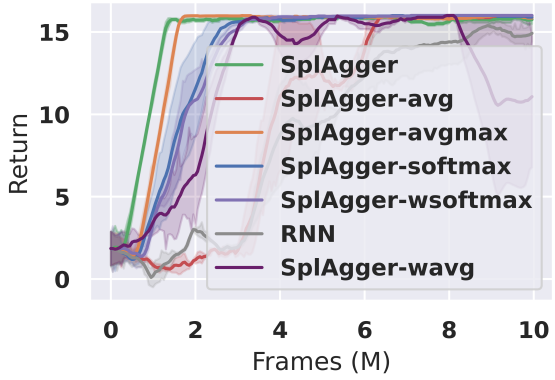


Figure C.6: The weighted average aggregator (wavg) performs worse than many others, including the default, max, on T-Maze Agreement.

In addition to the softmax aggregator, which computes a weighted softmax of the inputs, we experimented with a weighted average aggregator that does not use softmax. The weights and aggregates are still predicted separately for each input, as with wsoftmax aggregator, resulting in half the number of neurons output as are input. We chose to evaluate this aggregator as it more closely resembles the aggregator in PEARL. In fact, it is the same aggregation method if PEARL always output zero variance, and thus did not train a stochastic latent variable. Our results, shown in figure C.6 show this aggregation method to learn slightly faster than the average aggregator, but less stable. It also performed worse than max, softmax, and wsoftmax.

C.7 Environment Details

Cheetah-Dir. In Cheetah-Dir [Zintgraf et al., 2021b], the agent outputs six different torques in order to control a cheetah robot. The agent is rewarded for its velocity in

either the forward or reverse direction, depending on the sampled task, and receives a penalty for the magnitude of the torques. The agent receives a 17-dimensional observation consisting of the position, angle, and velocity of each body part. Here, the agent has one episode during which it can adapt to the task.

Walker. In Walker [Zintgraf et al., 2021b], the agent outputs six different torques in order to control a two-legged torso morphology. The agent receives 17-dimensional observations and receives a reward for running only in the forward direction. The tasks are defined by random samples of 65 different physics coefficients, such as body mass and friction, which collectively define the task. Here, the agent has two episodes during which it can adapt to the task.

T-LS. In the T-LS environment [Beck et al., 2020], the agent inhabits a T-shaped maze, called a T-Maze. The agent is shown a signal and then deterministically stepped down a corridor of length 100. At the end of the corridor, that agent opens one of two doors. If the door matches the signal, then the agent receives a reward of 4, and if not, it receives a penalty of -3. In between the start and end of the corridor, observations are augmented by a Bernoulli random variable. The agent adapts across four sequential episodes, which together constitute a single meta-episode.

MC-LS. The MC-LS environment [Beck et al., 2020] is designed to challenge an agent's long-term memory based on visual cues from Minecraft. The task involves navigating a sequence of 16 rooms. In each room, the agent must choose to go either left or right around a central column. This decision is based on the column's material: diamond or iron. Discrete actions allow for a finite set of observations. When the agent makes a decision in line with the observed material, it earns a reward of 0.1. The final decision at the end of the sequence is dictated by a color signal (red or green) presented at the beginning, which specifies the task. As in the T-Maze, the agent

receives a high reward (4) for a correct final action and a significant penalty (-3) for an incorrect one. The agent adapts across two sequential episodes, which together constitute a single meta-episode.

Planning Game. The Planning Game, is adapted from Ritter et al. [2021]. In the Planning Game [Ritter et al., 2021], the agent inhabits a 3×3 grid that wraps around each side. In this grid, the agent must navigate to a goal. The goal location, observed by the agent, changes multiple times during a single episode. The MDP is not defined by the goal location, shown to the agent, but rather by a changing transition function. The state in each cell of the grid changes in each task. A task is thus defined by a permutation of the 9 states. For instance, while the agent may be in the bottom right of the grid in two different tasks, it observes a different state there in each task. The states themselves are encoded by one-hot encodings. The agent must explore each state once to learn where each state is on the grid. Once it has seen each state, then it can immediately interpret the given goal instruction. Without knowing the state locations, there is a suboptimal policy that does not require remembering each state, namely, re-exploring the entire grid every time a goal is reached. Note the grid size in our experiments is 3×3, rather than the original 4×4, to decrease the total number of frames required for training.

C.8 Additional Baselines

SplAgger-noRNN-avg. This method replaces the max operator in SplAgger with an average and removes the RNN. Since removal of the RNN obviates the need for split aggregation, split aggregation is removed as well. This is equivalent to just computing a mean operation over linear encodings of each transition, τ_t.

SplAgger-avgmax. This method replaces the max operator in SplAgger with an average and a max operator. The avgmax aggregator averages half of the neurons and

computes a maximum over the other half. This aggregation is evaluated as a way to use both the average and max aggregation.

SplAgger-softmax. This method replaces the max operator in SplAgger with a softmax operator. The softmax aggregator computes an average over the inputs where the weights are determined by the softmax of the aggregates themselves. In order to aggregate online, in $O(1)$ memory, we store both the sum of the weighted aggregates, i.e., $n = e_0\exp(e_0/\eta) + ... + e_t\exp(e_t/\eta)$, and the sum of the weights seen so far, i.e., $d = \exp(e_0/\eta) + ... + \exp(e_t/\eta)$, where η is a learnable temperature parameter. The output is then the quotient of the weighted aggregates and the weights, n/d. This aggregator interpolates between the average and max. The initialization of the temperature is important, as it defines the interpolation, and is set to 0.1. Tuning information is available in Appendix C.1.

SplAgger-wsoftmax. This method replaces the max operator in SplAgger with a different softmax operator. This aggregator is the same as softmax, but the weights are predicted separately from the weighted encodings. Half of the input neurons are aggregated, and the other half are used to compute the logits for the softmax weights. An additional version of this aggregator without the softmax is investigated in Appendix C.6.

D

Appendix for Chapter 7, Protein Fitness Prediction

Contents

D.1 Reptile Details

This section provides additional details on how Reptile [Nichol et al., 2018] works. In order to account for fine-tuning during meta-training, gradient-based methods generally compute a *meta-gradient* that requires the computation of higher order derivatives, which can be computationally intractable for a large model. The costs can be especially burdensome when many gradient steps are needed for out of distribution adaptation. For this reason, Reptile avoids meta-gradients by changing optimization during meta-training. Here, the new parameters are updated, not by gradient descent,

but rather by moving toward the mean, after each theta is adapted to a task, $\theta_{\mathcal{T}}$, by fine-tuning on that task. From time-step t to $t+1$ of this outer-loop optimization process, Reptile can be written:

$$\theta^{t+1} = \theta^t + \beta \mathbb{E}_{\mathcal{D}_{\mathcal{T}} \in \mathcal{D}}[\theta^t_{\mathcal{T}} - \theta^t]. \tag{D.1}$$

We had to choose several implementation details for Reptile. First, note that Reptile sub-samples the support set during fine-tuning and has no distinct query set. In our implementation, we sub-sample mini-batches of size 50 to match the query size of *Metalic* for sub-sampling in the 128-shot setting. Reptile also has several options for the outer loop optimization. We choose to use the batched version with a batch size of 4 to match *Metalic*. Rather than using the direct update given in Equation (D.1), we take the difference over the learning rate, $(\theta^t_{\mathcal{T}} - \theta^t)/\alpha$ as an approximation of a gradient to be used with the Adam optimizer, as suggested by Nichol et al. [2018] and validated in Appendix D.2.

D.2 Hyper-Parameters

Since we build upon the axial attention of ProteinNPT [Notin et al., 2023], we follow their choice for most hyper-parameters, with a few exceptions. Most notably, in our experiments, we use the third layer of ESM2 as an embedding for each protein, given the strong performance and reduced number of parameters. Additionally, we use a ranking loss from Hawkins-Hooker et al. [2024], and do not use the CNN or additional inputs (such as zero-shot predictions) from ProteinNPT. We likewise found conditioning on the wild-type unhelpful. The same set of hyper-parameters are used for each setting: 0-shot, 16-shot, and 128-shot, and for multi-mutant results. We used the same learning rate for fine-tuning *Metalic* as for meta-training. We found the default ProteinNPT learning rate to be too large, and decreasing by a factor of 5 to

be sufficient. Consistent with our baselines, we select hyper-parameters using the single-mutant results and then use the same hyper-parameters for the multi-mutant setting, following Hawkins-Hooker et al. [2024], Notin et al. [2023]. Complete details on hyper-parameters used are available in Table D.1. We tuned relatively few of the hyper-parameters of our method, and mostly tuned over a single seed. There is likely room for improvement in the hyper-parameter selection of *Metalic*.

Table D.1: Hyper-Parameters for *Metalic*.

Hyper-Parameter	**Description**	**Value**
Training Steps	The total number of training steps in meta-training	50,000
Warm-Up Steps	The number of training steps spent linearly warming up, preceding cosine decay	5,000
Batch Size	The number of contexts evaluated per training step. Note that gradient accumulation is used for each context in the batch, so this scales training time linearly.	4
Weight Decay	Weight decay applied to non-bias parameters only	5e-3
Learning Rate	The learning rate for meta-training and fine-tuning	6e-5
Min LR Fraction	The minimum fraction of the LR maintained during the cosine decay in learning rate scheduling	1e-5
Adam Eps	The epsilon value for the Adam optimizer	1e-8
Adam Beta1	The beta1 value for the Adam optimizer	0.9
Adam Beta2	The beta2 value for the Adam optimizer	0.999
Gradient Clip Value	The maximum norm allowed for the gradient	1.0
ESM embed model	The full name for the ESM2 model used	esm_t6_8M_UR50D
ESM embed layer	The layer from the ESM2 model used as an embedding	3
Number Fine-tune Steps	The number of gradient updates for fine-tuning after meta-training	100
Num ProteinNPT Layers	The number of layers using axial attention, as in ProteinNPT	5

Hyper-Parameter	Description	Value
Condition on Pooled Sequence	Whether each sequence is pooled or ignored after axial attention	True
MLP Layer Sizes	The number and size of fully connected layers after axial attention	[768,768,768,768]
Embed Dim	The embedding dimension for all inputs including the protein sequences and fitness values	768
Axial Forward Embed Dim	The hidden size of the feed-forward layer within the ProteinNPT layer	400
Attention Heads	The number of heads in self-attention	4
Dropout Prob	The probability of dropout during training and fine-tuning for layers other than axial attention	0.0
Attention Dropout	The probability of dropout during training and fine-tuning for axial attention layers	0.1
Num Single Tasks	The total number of single-mutant tasks available. These tasks are included for meta-training even when testing on multi-mutants. Eight are held-out for evaluation when evaluating single-mutant performance.	121
Num Multiple Tasks	The total number of multi-mutant tasks available. These tasks are included for meta-training even when testing on single-mutants. Five are held-out for evaluation when evaluating multi-mutant performance.	68
Warm-up During Fine-tuning	Whether or not to use the linear warm-up from the learning rate scheduler during fine-tuning.	False

Table D.2: Reptile tuning results for the 128-shot setting. Results are trained over 10,000 steps for one seed each. The Adam optimizer performs the best. Both Adam, with an outer-loop learning rate of $3e^{-5}$, and Equation (D.1), with $\beta = 1$, correspond to the same scale in the outer loop.

Model Name	Adam ($3e^{-5}$)	$\beta = 1.$	$\beta = .333$	$\beta = 3e^{-5}$
Reptile-3-3	**.452**	.420	.350	.091
Reptile-3-100	**.472**	.444	.396	.181
Metalic-Reptile	**.483**	.476	.411	.190

For the majority of baselines no tuning was required, other than Reptile. For the baselines, we used reference predictions for Table 7.1 and reference implementations for Table 7.2, neither of which required tuning. For Reptile, the update in the outer-loop can be uses as written in Equation (D.1), or $(\theta_{\mathcal{T}}^t - \theta^t)/\alpha$ can be interpreted as a gradient for use with the Adam optimizer [Nichol et al., 2018]. We experiment with both and find the use of the Adam optimizer to be superior in performance. All results in the main body use the Adam optimizer for Reptile. In order to evaluate the method without the Adam optimizer, we re-tune β in Equation (D.1), which is the outer-loop learning rate. We leave the inner-loop learning rate, α, fixed at the learning rate for *Metalic*, $3e^{-5}$, which uses the same learning rate for the inner- and outer-loop. Given the increased computational cost of Reptile, we use a single seed over three learning rates for 10,000 steps. We tune over the following learning rates (β): the learning rate for *Metalic*, which is $3e^{-5}$; a learning rate of 1, which corresponds to the learning rate of *Metalic* if you interpret $(\theta_{\mathcal{T}}^t - \theta^t)/\alpha$ as the gradient in a gradient-based update [Nichol et al., 2018]; and a learning rate of $\frac{1}{n}$, where n is the number of inner-loop updates (3 in this case), which corresponds to the learning rate of *Metalic* if you interpret $(\theta_{\mathcal{T}}^t - \theta^t)/(\alpha n)$ as the gradient in a gradient-based update[1]. We found a learning rate of 1 to perform best, in line with the outer loop learning rate of *Metalic* and the interpretation of the gradient from Nichol et al. [2018]. For Adam, there is no

[1]The standard gradient update, $\theta^{t+1} = \theta^t + \alpha\nabla$, with $\nabla = (\theta_{\mathcal{T}}^t - \theta^t)/(\alpha n)$, gives $\theta^{t+1} = \theta^t + \alpha(\theta_{\mathcal{T}}^t - \theta^t)/(\alpha n)$, which is the same update as Equation (D.1) with $\beta = \frac{1}{n}$, omitting the expectation for brevity.

β, and we do not re-tune Adam's outer-loop learning rate. When using Adam, the gradient is interpreted as $(\theta_{\mathcal{T}}^t - \theta^t)/\alpha$. The results of tuning the learning rate without Adam, suggests a learning rate of $3e^{-5}$ with this gradient interpretation. Indeed, we confirm that when using Adam with this learning rate, and this gradient interpretation, it works better than using the alternative Reptile update rule with any learning rate. Results of the optimizer and learning rate tuning are presented in table Table D.2. Note that these results used an earlier set of hyper-parameters.

D.3 Augmentation Ablation

In this section we provide an additional ablation. Specifically, we ablate the augmentation of single-mutant training data with multi-mutant data (NoAug). The result is reported in Table D.3. Ablating the multi-mutant augmentation decreased performance in the zero-shot setting, with comparable 128-shot performance, indicating the benefit of additional meta-training data when fine-tuning data is most limited.

Table D.3: Augmentation Ablation in the 0 and 128-shot setting. Results show the importance of augmenting with multi-mutant data in the zero-shot setting.

Model Name	$n = 0$	$n = 128$
Metalic	**.482** ± .002	**.552** ± .009
Metalic-NoAug	.464 ± .011	**.558** ± .002

D.4 Fine-Tuning Warm-Up

Metalic does not use a warm-up period for fine-tuning as it does for meta-training. This decision was made since the warm-up normally occurs for 5,000 steps, but fine-tuning only occurs for 100 steps. As a compromise, we evaluate a warm-up period for fine-tuning, after taking a single gradient step with the full learning rate (*Metalic*-FTWarmUp). We still see that not using a warm-up for fine-tuning is superior. Results can be see in Table D.4.

Table D.4: Spearman correlation with and without warm-up.

Model Name	$n = 128$
Metalic	**.552** ± .009
Metalic-FTWarmUp	.539 ± .007

D.5 More Multi-Mutant Results

In this section we present a table of additional results for tasks with multiple mutants, including the *Metalic*-AuxIF models.

In Table D.6, we see the performance of *Metalic* increases as the amount of training data, measured in tasks, increases, providing more data is a path forward for strengthening the multi-mutant results. Here, we also see that the trend changes for *Metalic*-AuxIF, since it performs better with more data, but only if that data is multi-mutant data. This makes sense, since the auxiliary ESM-If1 predictions are most effective for multi-mutants, and training on single-mutants could discourage relying on them.

Table D.5: Spearman correlation for the 0, 16, and 128-shot setting for multi-mutant tasks.

Model Name	$n = 0$	$n = 16$	$n = 128$
Metalic (Both)	.436 ± .011	.484 ± .001	.670 ± .003
Metalic-AuxIF (Both)	.480 ± .007	.500 ± .002	.693 ± .006
Metalic-AuxIF (MultiTrain)	.533 ± .012	.577 ± .008	.692 ± .003
ESM1-v-650M	.426 ± .000	.557 ± .000	.644 ± .000
ESM2-8M	.368 ± .000	.484 ± .000	.596 ± .000
PoET	**.588** ± .014	**.638** ± .001	**.734** ± .006
ProteinNPT	N/A	.367 ± .003	.624 ± .003

Table D.6: Multi-mutant Spearman correlation by training data. Results are computed using predictions provided by ProteinGym on five tasks with multiple mutations with different amounts of single- and multi-mutant training tasks. We see performance of *Metalic* increases with more data, giving a path for future data collection. We also see here that *Metalic*-AuxIF performs better when trained with only multi-mutant data. This makes sense, since the augmented predictions are most effective for multi-mutants, and training on single-mutants could discourage relying on them. An asterisk (*) represents the default training data regime for that method.

	Single-	Multi-	Multiples
Model Name	Mutant Tasks	Mutant Tasks	$n = 0$
Metalic (Both)*	121	63	**.436 ± .011**
Metalic (MultiTrain)	0	63	.376 ± .018
Metalic (MultiTrainHalf)	0	30	.307 ± .022
Metalic-AuxIF (Both)	121	63	.480 ± .007
Metalic-AuxIF (MultiTrain)*	0	63	**.533** ± .012
Metalic-AuxIF (MultiTrainHalf)	0	30	.517 ± .021

D.6 Detailed Architecture

Here, we present Figure D.1, which gives a more detailed version of the architecture diagram.

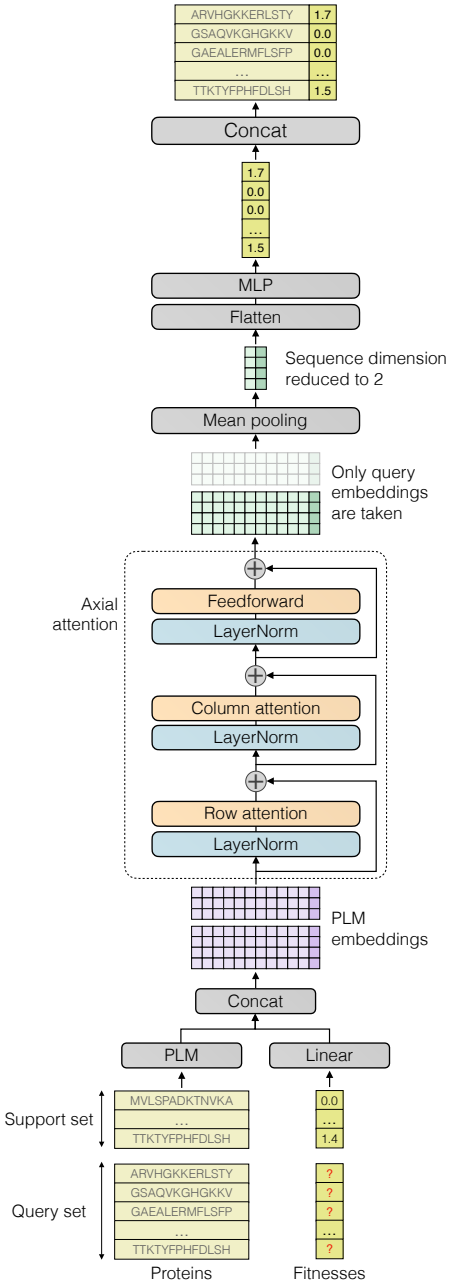


Figure D.1: Detailed diagram of architecture. Note that fitness embeddings are computed using a learned linear projection for the support set and a shared learned embedding vector for the query set.

Bibliography

K. ab Abebe Tessera, A. Rahman, and S. V. Albrecht. Hypermarl: Adaptive hypernetworks for multi-agent rl. *arXiv*, 2024.

K. Akuzawa, Y. Iwasawa, and Y. Matsuo. Estimating disentangled belief about hidden state and hidden task for meta-reinforcement learning. In *Learning for Dynamics and Control*, pages 73–86. PMLR, 2021.

M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *The International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sk2u1g-0-`.

A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy. Deep variational information bottleneck. In *The International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=HyxQzBceg`.

M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

C. Angermueller, D. Dohan, D. Belanger, R. Deshpande, K. Murphy, and L. Colwell. Model-based reinforcement learning for biological sequence design. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=HklxbgBKvr`.

A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. In *The International conference on machine learning*, 2020.

S. Bechtle, A. Molchanov, Y. Chebotar, E. Grefenstette, L. Righetti, G. Sukhatme, and F. Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE Computer Society, 2021.

J. Beck. Offline rlaif: Piloting vlm feedback for rl via sfo. *RLC Workshop on Reinforcement Learning Beyond Rewards: Ingredients for Developing Generalist Agents*, 2025. URL `https://arxiv.org/abs/2503.01062`.

J. Beck, K. Ciosek, S. Devlin, S. Tschiatschek, C. Zhang, and K. Hofmann. Amrl: Aggregated memory for reinforcement learning. In *The International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=Bkl7bREtDr`.

J. Beck, M. Jackson, R. Vuorio, and S. Whiteson. Hypernetworks in meta-reinforcement learning. *The Conference on Robot Learning (CoRL)*, 2022. URL `https://openreview.net/forum?id=N-HtsQkRotI`.

J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. A survey of meta-reinforcement learning. *arXiv*, 2023a.

J. Beck, R. Vuorio, Z. Xiong, and S. Whiteson. Recurrent hypernetworks are surprisingly strong in meta-rl. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL `https://openreview.net/forum?id=pefAAzu8an`.

J. Beck, M. Jackson, R. Vuorio, Z. Xiong, and S. Whiteson. Splagger: Split aggregation for meta-reinforcement learning. *Reinforcement Learning Conference*, 2024a. URL `https://openreview.net/forum?id=O1Vmua4RVW`.

J. Beck, S. Surana, M. McAuliffe, O. Bent, T. D. Barrett, J. J. G. Luis, and P. Duckworth. Metalic: Meta-learning in-context with protein language models. *The NeurIPS 2024 Workshop on Foundation Models for Science (FM4Science)*, 2024b. URL https://neurips.cc/virtual/2024/105896.

J. Beck, S. Surana, M. McAuliffe, O. Bent, T. D. Barrett, J. J. G. Luis, and P. Duckworth. Metalic: Meta-learning in-context with protein language models. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=TUKt7ag0qq.

J. Beck, R. Vuorio, E. Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, and S. Whiteson. A tutorial on meta-reinforcement learning. *Foundations and Trends in Machine Learning*, 2025b.

G. Bellec, D. Salaj, A. Subramoney, R. Legenstein, and W. Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *The Conference on Neural Information Processing Systems*, 2018.

M. G. Bellemare, S. Candido, P. S. Castro, J. Gong, M. C. Machado, S. Moitra, S. S. Ponda, and Z. Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

C. Boutilier, C.-w. Hsu, B. Kveton, M. Mladenov, C. Szepesvari, and M. Zaheer. Differentiable meta-learning of bandit policies. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2122–2134. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/171ae1bbb81475eb96287dd78565b38b-Paper.pdf.

D. H. Brookes, J. Otwinowski, and S. Sinai. Contrastive losses as generalized models of global epistasis. *arXiv*, 2023.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

M. Budd, B. Lacerda, and N. Hawes. Stop! planner time: metareasoning for probabilistic planning using learned performance profiles. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *The International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=H1lJJnR5Ym`.

M. Chalvidal, T. Serre, and R. VanRullen. Meta-reinforcement learning with self-modifying networks. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=cYeYzaP-5AF`.

S. C. Y. Chan, I. Dasgupta, J. Kim, D. Kumaran, A. K. Lampinen, and F. Hill. Transformers generalize differently from information stored in context vs in weights. In *Workshop on Memory in Artificial and Real Intelligence at NeurIPS*, 2022.

O. Chang, L. Flokas, and H. Lipson. Principled weight initialization for hypernetworks. In *The International Conference on Learning Representations*, 2020.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *EMNLP*, 2014.

E. Choshen and A. Tamar. Contrabar: Contrastive bayes-adaptive deep rl. *The International Conference on Machine Learning*, 2023.

A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways. *arXiv*, 2022.

J. Coda-Forno, M. Binz, Z. Akata, M. Botvinick, J. Wang, and E. Schulz. Meta-in-context learning in large language models. *Advances in Neural Information Processing Systems*, 2023.

M. Cooper, J. K. Lee, J. Beck, J. D. Fishman, M. Gillett, Z. Papakipos, A. Zhang, J. Ramos, A. Shah, and M. L. Littman. Stackelberg punishment and bully-proofing autonomous vehicles. In *Social Robotics: 11th International Conference*, 2019.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines, 2017.

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

M. O. Duff and A. Barto. *Optimal Learning: Computational Procedures for Bayes-Adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002.

A. Elawady, G. Chhablani, R. Ramrakhya, K. Yadav, D. Batra, Z. Kira, and A. Szot. Relic: A recipe for 64k steps of in-context reinforcement learning for embodied ai. *arXiv*, 2024.

D. Emukpere, X. Alameda-Pineda, and C. Reinke. Successor feature neural episodic control. *arXiv preprint arXiv:2111.03110*, 2021.

R. Fakoor, P. Chaudhari, S. Soatto, and A. J. Smola. Meta-q-learning. In *The International Conference on Learning Representations*, 2020.

C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *The International Conference on Learning Representations*, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *The International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

M. Fortunato, M. Tan, R. Faulkner, S. Hansen, A. P. Badia, G. Buttimore, C. Deck, J. Z. Leibo, and C. Blundell. Generalization of reinforcement learners with working

and episodic memory. *The Conference on Neural Information Processing Systems*, 2019.

D. M. Fowler and S. Fields. Deep mutational scanning: a new style of protein science. In *Nature Methods*, 2014.

H. Fu, H. Tang, J. Hao, C. Chen, X. Feng, D. Li, and W. Liu. Towards effective context for meta-reinforcement learning: an approach based on contrastive learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

A. Galashov, J. Schwarz, H. Kim, M. Garnelo, D. Saxton, P. Kohli, S. M. A. Eslami, and Y. W. Teh. Meta-learning surrogate models for sequential decision making. *The International Conference on Learning Representations Workshop on Structure and Priors in Reinforcement Learning*, 2019.

K. Gao and O. Sener. Modeling and optimization trade-off in meta-learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11154–11165. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/7fc63ff01769c4fa7d9279e97e307829-Paper.pdf`.

F. M. Garcia and P. S. Thomas. A meta-mdp approach to exploration for lifelong reinforcement learning. *The Conference on Neural Information Processing Systems*, 2019.

M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. M. A. Eslami. Conditional neural processes. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713, 2018a.

M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. M. A. Eslami, and Y. W. Teh. Neural processes. *The International Conference on Machine Learning*, 2018b.

M. Gerstgrasser and D. C. Parkes. Meta-RL for multi-agent RL: Learning to adapt to evolving agents. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022.

A. Ghadirzadeh, X. Chen, P. Poklukar, C. Finn, M. Björkman, and D. Kragic. Bayesian meta-learning for few-shot policy adaptation across robotic platforms. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1274–1280, 2021. doi: 10.1109/IROS51168.2021.9636628.

A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv*, abs/1410.5401, 2014. URL `http://arxiv.org/abs/1410.5401`.

Y. S. Grewal, F. de Nijs, and S. Goodwin. Variance-seeking meta-exploration to handle out-of-distribution tasks. In *Deep RL Workshop NeurIPS 2021*, 2021.

J. Grigsby, L. Fan, and Y. Zhu. Amago: Scalable in-context reinforcement learning for adaptive agents. *The International Conference on Learning Representations*, 2024.

A. Guez, D. Silver, and P. Dayan. Scalable and efficient bayes-adaptive reinforcement learning based on monte-carlo tree search. *Journal of Artificial Intelligence Research*, 48:841–883, 2013.

Z. D. Guo, M. G. Azar, B. Piot, B. A. Pires, and R. Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018.

A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine. Meta-reinforcement learning of structured exploration strategies. In S. Bengio, H. Wallach,

H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/4de754248c196c85ee4fbdcee89179bd-Paper.pdf.

S. Gurumurthy, S. Kumar, and K. Sycara. Mame : Model-agnostic meta-exploration. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 910–922, 2020.

D. Ha, A. Dai, and Q. V. Le. Hypernetworks. In *The International Conference on Learning Representation (ICLR)*, 2017.

M. J. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps. In *AAAI Fall Symposia*, 2015.

A. Hawkins-Hooker, J. Kmec, O. Bent, and P. Duckworth. Likelihood-based fine-tuning of protein language models for few-shot fitness prediction and design. In *ICML 2024 Workshop on Efficient and Accessible Foundation Models for Biological Discovery*, 2024.

J. Z.-Y. He, Z. Erickson, D. S. Brown, A. Raghunathan, and A. Dragan. Learning representations that enable generalization in assistive tasks. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=b88HF4vd_ej.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

D. O. Hebb. The organization of behavior; a neuropsychological theory. *Wiley*, 1949.

N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver. Memory-based control with recurrent neural networks. *NIPS Deep Reinforcement Learning Workshop*, 2015.

M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans. Axial attention in multidimensional transformers. *arXiv*, 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. Jonathan Ho, and P. Abbeel. Evolved policy gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/7876acb66640bad41f1e1371ef30c180-Paper.pdf`.

C. Hsu, R. Verkuil, J. Liu, Z. Lin, B. Hie, T. Sercu, A. Lerer, and A. Rives. Learning inverse folding from millions of predicted structures. In *International conference on machine learning*, 2022.

M. Huisman, J. N. Van Rijn, and A. Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021.

J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

T. Imagawa, T. Hiraoka, and Y. Tsuruoka. Off-policy meta-reinforcement learning with belief-based task inference. *IEEE Access*, 10:49494–49507, 2022.

K. Javed and M. White. Meta-learning representations for continual learning. *Advances in neural information processing systems*, 2019.

P.-A. Kamienny, M. Pirotta, A. Lazaric, T. Lavril, N. Usunier, and L. Denoyer. Learning adaptive exploration strategies in dynamic environments through informed policy regularization. *arXiv preprint arXiv:2005.02934*, 2020.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *The International Conference on Machine Learning*, 2020.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv*, 2013.

R. Kirk, A. Zhang, E. Grefenstette, and T. Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, 2023.

L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

L. Kirsch, J. Harrison, J. Sohl-Dickstein, and L. Metz. General-purpose in-context learning by meta-learning transformers. *arXiv*, 2022.

I. Korshunova, J. Degrave, J. Dambre, A. Gretton, and F. Huszar. Exchangeable models in meta reinforcement learning. In *4th Lifelong Machine Learning Workshop at The International Conference on Machine Learning 2020*, 2020. URL `https://openreview.net/forum?id=TZFlzejFPT`.

B. Krause, N. Naik, W. Liu, and A. Madani. Don't throw away that linear head: Few-shot protein fitness prediction with generative models, 2022. URL `https://openreview.net/forum?id=hHmtmT58pSL`.

A. Kumar, Z. Fu, D. Pathak, and J. Malik. RMA: rapid motor adaptation for legged robots. *RSS*, abs/2107.04034, 2021.

S. Kumar, I. Dasgupta, J. D. Cohen, N. D. Daw, and T. L. Griffiths. Meta-learning of compositional task distributions in humans and machines. In *4th Workshop on Meta-Learning at NeurIPS*, 2020.

H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

B. Kveton, M. Konobeev, M. Zaheer, C.-W. Hsu, M. Mladenov, C. Boutilier, and C. Szepesvari. Meta-thompson sampling. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5884–5893. PMLR, 2021. URL `https://proceedings.mlr.press/v139/kveton21a.html`.

Q. Lan, A. R. Mahmood, S. Yan, and Z. Xu. Learning to optimize for reinforcement learning. *arXiv preprint arXiv:2302.01470*, 2023.

J. Lee, A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill. Supervised pretraining can learn in-context reinforcement learning. *Advances in Neural Information Processing Systems*, 2024.

K. Li and J. Malik. Learning to optimize. *The International Conference on Learning Representations*, 2017.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-sgd: Learning to learn quickly for few shot learning. *CoRR*, abs/1707.09835, 2017. URL `http://arxiv.org/abs/1707.09835`.

Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, A. d. Santos Costa, M. Fazel-Zarandi, T. Sercu, S. Candido, and A. Rives. Language models of protein sequences at the scale of evolution enable accurate structure prediction. *bioRxiv*, 2022.

E. Z. Liu, A. Raghunathan, P. Liang, and C. Finn. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In *The International Conference on Machine Learning*, pages 6925–6935. PMLR, 2021.

C. Lu, J. Kuba, A. Letcher, L. Metz, C. Schroeder de Witt, and J. Foerster. Discovered policy optimisation. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 16455–16468. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/688c7a82e31653e7c256c6c29fd3b438-Paper-Conference.pdf`.

F.-M. Luo, S. Jiang, Y. Yu, Z. Zhang, and Y.-F. Zhang. Adapt to environment sudden changes by learning a context sensitive policy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

A. Madani, B. McCann, N. Naik, N. S. Keskar, N. Anand, R. R. Eguchi, P.-S. Huang, and R. Socher. Progen: Language modeling for protein generation. *arXiv*, 2020.

C. Marquet, M. Heinzinger, T. Olenyi, C. Dallago, K. Erckert, M. Bernhofer, D. Nechaev, and B. Rost. Embeddings from protein language models predict conservation and variant effects. *Human genetics*, 2022.

J. Meier, R. Rao, R. Verkuil, J. Liu, T. Sercu, and A. Rives. Language models enable zero-shot prediction of the effects of mutations on protein function. *Advances in neural information processing systems*, 2021.

L. C. Melo. Transformers are meta-reinforcement learners. *The International Conference on Machine Learning*, 2022.

T. Miconi, K. Stanley, and J. Clune. Differentiable plasticity: training plastic neural networks with backpropagation. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3559–3568, 2018.

T. Miconi, A. Rawal, J. Clune, and K. O. Stanley. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. In *The International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1lrAiA5Ym`.

T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. In *The International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1DmUzWAW`.

I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. *Conference on Computer Vision and Pattern Recognition*, 2016.

A. Moeini, J. Wang, J. Beck, E. Blaser, S. Whiteson, R. Chandra, and S. Zhang. A survey of in-context reinforcement learning. *arXiv*, 2025.

Y. Mu, Y. Zhuang, F. Ni, B. Wang, J. Chen, J. Hao, and P. Luo. DOMINO: Decomposed mutual information optimization for generalized context in meta-reinforcement learning. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=CJGUABT_COm`.

T. Munkhdalai and H. Yu. Meta networks. *The International Conference on Machine Learning*, 2017.

K. P. Murphy. Conjugate bayesian analysis of the gaussian distribution. *Technical report*, 2007.

E. Najarro and S. Risi. Meta-learning through hebbian plasticity in random networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20719–20731. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/ee23e7ad9b473ad072d57aaa9b2a5222-Paper.pdf`.

H. H. Nguyen, A. Baisero, D. Wang, C. Amato, and R. Platt. Leveraging fully observable policies for learning under partial observability. In *6th Annual Conference on Robot Learning*, 2022. URL `https://openreview.net/forum?id=pn-HOPBioUE`.

T. Nguyen and A. Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *The International Conference on Machine Learning*, 2022.

T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent model-free RL can be a strong baseline for many POMDPs. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

P. Notin, M. Dias, J. Frazer, J. Marchena-Hurtado, A. N. Gomez, D. Marks, and Y. Gal. Tranception: protein fitness prediction with autoregressive transformers and inference-time retrieval. In *International Conference on Machine Learning*, pages 16990–17017. PMLR, 2022.

P. Notin, R. Weitzman, D. Marks, and Y. Gal. Proteinnpt: Improving protein property prediction and design with non-parametric transformers. *Advances in Neural Information Processing Systems*, 2023.

P. Notin, A. Kollasch, D. Ritter, L. Van Niekerk, S. Paul, H. Spinner, N. Rollins, A. Shaw, R. Orenbuch, R. Weitzman, et al. Proteingym: Large-scale benchmarks for protein fitness prediction and design. *Neural Information Processing Systems*, 2024.

J. Oh, V. Chockalingam, S. P. Singh, and H. Lee. Control of memory, active perception, and action in minecraft. *The International Conference on Machine Learning*, 2016.

J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering reinforcement learning algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

I. Osband, D. Russo, and B. Van Roy. (more) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, 2013.

E. Parisotto, J. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *The International Conference on Learning Representations*, 2016.

E. Parisotto, H. F. Song, J. W. Rae, R. Pascanu, Ç. Gülçehre, S. M. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, M. M. Botvinick, N. Heess, and R. Hadsell. Stabilizing transformers for reinforcement learning. *The International Conference on Machine Learning*, 2020.

J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, et al. Automated reinforcement learning (autorl): A survey and open problems. *arXiv preprint arXiv:2201.03916*, 2022.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *The International Conference on Machine Learning*, 2013.

M. Peng, B. Zhu, and J. Jiao. Linear representation meta-reinforcement learning for instant adaptation. *submitted to International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=lNrtNGkr-vw`.

E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. C. Courville. Film: Visual reasoning with a general conditioning layer. *AAAI*, 2018.

M. Przewiezlikowski, P. Przybysz, J. Tabor, M. Zieba, and P. Spurek. Hypermaml: Few-shot adaptation of deep models with hypernetworks. *arXiv*, 2022.

R. Raileanu, M. Goldstein, A. Szlam, and R. Fergus. Fast adaptation via policy-dynamics value functions. *The International Conference on Machine Learning*, 2020.

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

R. M. Rao, J. Liu, R. Verkuil, J. Meier, J. Canny, P. Abbeel, T. Sercu, and A. Rives. Msa transformer. In *International Conference on Machine Learning*. PMLR, 2021.

S. C. Raparthy, E. Hambro, R. Kirk, M. Henaff, and R. Raileanu. Generalization to new sequential decision making tasks with in-context learning. *NeurIPS 2023 Workshop on Foundation Models for Decision Making*, 2023.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *The International Conference on Learning Representations*, 2017.

Z. Rimon, A. Tamar, and G. Adler. Meta reinforcement learning with finite training tasks - a density estimation approach. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=Y-sdZLIi9R9`.

S. Ritter, J. Wang, Z. Kurth-Nelson, S. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick. Been there, done that: Meta-learning with episodic recall. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4354–4363, 2018a.

S. Ritter, J. Wang, Z. Kurth-Nelson, S. Jayakumar, C. Blundell, R. Pascanu, and M. Botvinick. Been there, done that: Meta-learning with episodic recall. In *The International Conference on Machine Learning*, 2018b.

S. Ritter, R. Faulkner, L. Sartran, A. Santoro, M. Botvinick, and D. Raposo. Rapid task-solving in novel environments. In *The International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=F-mvpFpn_0q`.

A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. L. Zitnick, J. Ma, et al. Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences*, 2021.

S. R. R. Rohani, S. Hedayatian, and M. S. Baghshah. Bimrl: Brain inspired meta reinforcement learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9048–9053. IEEE, 2022.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International conference on learning representations*, 2018.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *The International Conference on Learning Representations*, 2019.

A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *The International conference on machine learning*, pages 1842–1850. PMLR, 2016.

E. Sarafian, S. Keynan, and S. Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9301–9312, 2021.

A. M. Saxe, J. L. Mcclelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural network. In *In International Conference on Learning Representations*, 2014.

J. Schmidhuber. *Evolutionary principles in self-referential learning.* PhD thesis, Technische Universität München, 1987.

J. Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In *Artificial general intelligence*, pages 199–226. Springer, 2007.

J. Schmidhuber, J. Zhao, and M. Wiering. Shifting inductive bias with success-story algorithm, adaptive levin search, and incremental self-improvement. *Machine Learning*, 28(1):105–130, 1997.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

M. Simchowitz, C. Tosh, A. Krishnamurthy, D. J. Hsu, T. Lykouris, M. Dudik, and R. E. Schapire. Bayesian decision-making under misspecified priors with applications to meta-learning. *Advances in Neural Information Processing Systems*, 34:26382–26394, 2021.

X. Song, Y. Yang, K. Choromanski, K. Caluwaerts, W. Gao, C. Finn, and J. Tan. Rapidly adaptable legged robots via evolutionary meta-learning. *The IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

B. C. Stadie, G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *The Conference on Neural Information Processing Systems*, 2018.

M. Strens. A bayesian framework for reinforcement learning. In *International Conference on Machine Learning*, 2000.

J. Subramanian, A. Sinha, R. Seraj, and A. Mahajan. Approximate information state for approximate planning and reinforcement learning in partially observed systems. *The Journal of Machine Learning Research*, 2022.

F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. *CVPR*, 2018.

R. Sutton. The bitter lesson. *Incomplete Ideas (blog)*, 2019. URL `http://www.incompleteideas.net/IncIdeas/BitterLesson.html`.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, 1999.

A. A. Team, J. Bauer, K. Baumli, S. Baveja, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

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

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*, pages 5026–5033. IEEE, 2012.

E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *International Conference on Learning Representations*, 2020.

T. Truong Jr and T. Bepler. Poet: A generative model of protein families as sequences-of-sequences. *Advances in Neural Information Processing Systems*, 2024.

J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *The Conference on Neural Information Processing Systems*, 2017.

O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 2016.

R. Vuorio, D.-Y. Cho, D. Kim, and J. Kim. Meta continual learning. *arXiv*, 2018.

R. Vuorio, S.-H. Sun, H. Hu, and J. Lim. Multimodal model-agnostic meta-learning via task-aware modulation. In *The Conference on Neural Information Processing Systems*, 2019.

R. Vuorio, J. A. Beck, G. Farquhar, J. N. Foerster, and S. Whiteson. No dice: An investigation of the bias-variance tradeoff in meta-gradients. In *Deep RL Workshop NeurIPS 2021*, 2021.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv*, 2023.

H. Wang, J. Zhou, and X. He. Learning context-aware task reasoning for efficient meta reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020a.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

J. X. Wang, M. King, N. Porcel, Z. Kurth-Nelson, T. Zhu, C. Deck, P. Choy, M. Cassin, M. Reynolds, H. F. Song, G. Buttimore, D. P. Reichert, N. C. Rabinowitz, L. Matthey, D. Hassabis, A. Lerchner, and M. M. Botvinick. Alchemy: A structured task distribution for meta-reinforcement learning. *The Conference on Neural Information Processing Systems*, 2021.

Q. Wang and H. van Hoof. Learning expressive meta-representations with mixture of expert neural processes. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL `https://openreview.net/forum?id=ju38DG3sbg6`.

Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)*, 53(3):1–34, 2020b.

Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. Erickson. Rl-vlm-f: Reinforcement learning from vision language foundation model feedback. *The International Conference on Machine Learning*, 2024.

L. Weihs, U. Jain, I.-J. Liu, J. Salvador, S. Lazebnik, A. Kembhavi, and A. Schwing. Bridging the imitation gap by adaptive insubordination. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL `https://openreview.net/forum?id=WlxODqiUTD_`.

L. Wen, S. Zhang, H. E. Tseng, B. Singh, D. Filev, and H. Peng. Improved robustness and safety for pre-adaptation of meta reinforcement learning with prior regularization. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8987–8994. IEEE, 2022.

Y. Wu, M. Ren, R. Liao, and R. Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *The International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1MczcgR-.

Z. Xian, S. Lal, H.-Y. Tung, E. A. Platanios, and K. Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. In *The International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=pHXfe1cOmA.

Z. Xiong, L. M. Zintgraf, J. A. Beck, R. Vuorio, and S. Whiteson. On the practical consistency of meta-reinforcement learning algorithms. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=xwQgKphwhFA.

Z. Xiong, J. Beck, and S. Whiteson. Universal morphology control via contextual modulation. In *The International Conference on Machine Learning*, pages 38286–38300. PMLR, 2023.

Z. Xiong, R. Vuorio, J. Beck, M. Zimmer, K. Shao, and S. Whiteson. Distilling morphology-conditioned hypernetworks for efficient universal morphology control. *arXiv preprint arXiv:2402.06570*, 2024. URL https://openreview.net/forum?id=WjvEvYTy3w.

M. Xu, Y. Shen, S. Zhang, Y. Lu, D. Zhao, J. Tenenbaum, and C. Gan. Prompting decision transformer for few-shot policy generalization. In *The International conference on machine learning*, pages 24631–24645. PMLR, 2022. URL https://openreview.net/forum?id=7rex8lEZH2.

H. Yin, J. Chen, S. J. Pan, and S. Tschiatschek. Sequential generative exploration model for partially observable reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/e1021d43911ca2c1845910d84f40aeae-Paper.pdf`.

T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn. Multi-task reinforcement learning without interference. *Advances in Neural Information Processing Systems*, 2019.

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*, pages 1094–1100. PMLR, 2020. URL `http://proceedings.mlr.press/v100/yu20a/yu20a.pdf`.

H. Zhang and Z. Kan. Temporal logic guided meta q-learning of multiple tasks. *IEEE Robotics and Automation Letters*, 7(3):8194–8201, 2022. URL `https://ieeexplore.ieee.org/document/9803859`.

J. Zhang, J. Wang, H. Hu, T. Chen, Y. Chen, C. Fan, and C. Zhang. Metacure: Meta reinforcement learning with empowerment-driven exploration. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12600–12610, 2021.

D. Zhao, S. Kobayashi, J. Sacramento, and J. von Oswald. Meta-learning via hypernetworks. In *Fourth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2020.

W. Zhou, L. Pinto, and A. Gupta. Environment probing interaction policies. In *The International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryl8-3AcFX.

L. Zintgraf, K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: A very good method for bayes-adaptive deep rl via meta-learning. In *The International Conference on Learning Representation (ICLR)*, 2020. URL https://openreview.net/forum?id=Hkl9JlBYvr.

L. Zintgraf, S. Devlin, K. Ciosek, S. Whiteson, and K. Hofmann. Deep interactive bayesian reinforcement learning via meta-learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1712–1714, 2021a.

L. Zintgraf, S. Schulze, C. Lu, L. Feng, M. Igl, K. Shiarlis, Y. Gal, K. Hofmann, and S. Whiteson. Varibad: Variational bayes-adaptive deep rl via meta-learning. *Journal of Machine Learning Research*, 22(289):1–39, 2021b. URL https://jmlr.org/papers/volume22/21-0657/21-0657.pdf.

L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. *The International Conference on Learning Representations*, 2019.

L. M. Zintgraf, L. Feng, C. Lu, M. Igl, K. Hartikainen, K. Hofmann, and S. Whiteson. Exploration in approximate hyper-state space for meta reinforcement learning. In *The International Conference on Machine Learning*, pages 12991–13001. PMLR, 2021c.