Explorations of In-Context Reinforcement Learning

by

Ethan Brooks

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2024

Doctoral Committee:

Professor Satinder Singh, Chair
Professor Richard L. Lewis, Co-Chair
Professor Honglak Lee
Professor Rada Mihalcea
Professor Thad Polk

Ethan Brooks

ethanbro@umich.edu

ORCID iD: 0000-0003-2557-4994

Dedication

I dedicate this thesis to my beloved family, who supported me emotionally, editorially, and culinarily throughout this work. When I needed wisdom, you gave it. When I needed comfort, you gave it. When I needed solitude, you gave it. When I needed caffeine, you gave it. I can never repay the debt, but you have my endless love and gratitude.

Acknowledgements

I am deeply grateful for the many individuals and institutions that have supported me on my journey through the doctoral program and ultimately made this dissertation possible.

Toyota Research Institute My initial steps into the world of reinforcement learning research were funded by the Toyota Research Institute. While their grant did not directly support the work presented here, their early investment provided me with invaluable resources and opportunities to explore the field and lay the groundwork for my future research.

Defense Advanced Research Projects Agency (DARPA) The DARPA L2M grant marked a pivotal moment in my development as a researcher. This funding enabled me to produce my first research paper and provided formative experience in the practical work of training deep RL agents and producing scientific knowledge. Moreover, the ideas explored during this project laid the foundation for my later work on generalization, which forms a significant component of this dissertation.

DeepMind My internship at DeepMind was a transformative experience. Not only did I develop the core components of Chapter 2 within this intellectually vibrant setting, but I also had the opportunity to savor a culture of scientific excellence and innovation that is unparalleled in my experience. The endless stream of mind-bending

conversations with my brilliant colleagues planted seeds of ideas that continue to grow and blossom. I would like to especially call out the incredible support and mentorship of my manager Vlad Mnih, and my colleagues Misha Laskin, Sebastian Flennerhag, and Alex Neitz.

My Labmates, My Intellectual and Emotional Pillars: No journey of this magnitude can be undertaken alone. I owe an immense debt of gratitude to my lab-mates, especially Chris Grimm, Zeyu Zhang, and Risto Vuori. Their unwavering intellectual and emotional support throughout my PhD journey has been immeasurable. From enduring countless pitches of hair-brained ideas and providing insightful feedback to fanning the flames of hope during the inevitable setbacks of research, they have consistently supported and often challenged me to push my boundaries, making this journey not only more rewarding but also far more fun.

My Advisors and my Committee Members Finally, I express my deepest gratitude to my advisors, Professor Satinder Singh and Professor Rick Lewis, and to the rest of my Committee. First, I am honored by and thankful for the time and feedback of my committee members, Professor Honglak Lee, Professor Rada Mihalcea, and Professor Thad Polk. I am also so grateful for the first-class scientific education that I received from Rick Lewis, who has been a constant sounding board throughout my research journey, subjecting my ideas and my writing to meticulous examination and providing invaluable critiques. His unwavering support, particularly in the final stretch of my dissertation, has been truly indispensible.

To Satinder, I owe a profound debt of gratitude. At times, he “imparted truth and health in rough electric shocks.” But I came to recognize that what at first could seem like inflexibility arose from an unwavering dedication to intellectual and scientific excellence. I am incredibly fortunate to have benefited from his uncompromising critical lens these past six years, a lens which saved me months of frustration by steering me away from many a dead-end project. He challenged me to meet intellectual standards that truly felt out of reach at the start of this journey, and for that he has my lifelong gratitude.

To each and every one of you, my heartfelt thanks. Your support and guidance have shaped me not only as a researcher but also as a person.

Table of Contents

List of Figures

List of Acronyms

AD: Algorithm Distillation

AD++: Algorithm Distillation ++

BUN: Bellman Update Network

GPT: Generative Pre-trained Transformer

GRU: Gated Recurrent Unit

ICPI: In-Context Policy Iteration

LSTM: Long Short-Term Memory unit

LLM: Large Language Model

RL: Reinforcement Learning

Abstract

In-Context Learning describes a form of learning that occurs when a model accumulates information in its context or memory. For example, a Long Short-Term Memory unit (LSTM) can rapidly adapt to a novel task as input/target exemplars are fed into it. While in-context learning of this kind is not a new discovery, recent work has demonstrated the capacity of large "foundation" models to acquire this ability "for free," by training on large quantities of semi-supervised data, without the sophisticated (but often unstable) meta-objectives proposed by many earlier papers (Finn et al. 2017; Rakelly et al. 2019; Stadie et al. 2018). In this work we explore several algorithms which specialize in-context learning based on semi-supervised methods to the reinforcement learning (RL) setting. In particular, we explore three approaches to in-context learning of value (expected cumulative discounted reward).

The first of these methods demonstrates a method for implementing policy iteration, a classic RL algorithm, using a pre-trained large language model (LLM). We use the LLM to generate planning rollouts and extract monte-carlo estimates of value from them. We demonstrate the method on several small, text-based domains and present evidence that the LLM can generalize to unseen states, a key requirement of learning in non-tabular settings.

The second method imports many of the ideas of the first, but trains a transformer model directly on offline RL data. We incorporate Algorithm Distillation (AD) (Laskin et al. 2022), another method for in-context reinforcement learning that directly distills the improvement operator from data that includes behavior ranging from random to optimal. Our method combines the benefits of AD with the policy iteration method proposed in our previous work and demonstrates benefits in performance and generalization.

Our third method proposes a new method for estimating value. Like the previous methods, this one implements a form of policy iteration, but eschews monte-carlo rollouts for a new approach to estimating value. We train a network to estimate Bellman updates and iteratively feed its outputs back into itself until the estimate converges. We find that this iterative approach improves the capability of the value estimates to generalize and mitigates some of the instability of other offline methods.

1 Introduction

1.1 The Importance of Rapid Adaptation

Deep neural networks optimized by gradient descent have defined the most pioneering techniques of the last decade of machine learning research. While other architectures may provide stronger guarantees, they often require certain assumptions such as linearity, convexity, and smoothness that are not present in reality. As a result, researchers have turned to general-purpose algorithms that make few assumptions about the functions they approximate.

However, these algorithms' generality comes at a cost: they learn slowly and require thousands or millions of gradient updates to converge. This contrasts with human learners, who can quickly master new tasks after brief exposure. Rapid adaptation is a key feature of human intelligence, allowing us to behave intelligently and develop new skills in unfamiliar settings (Lake et al. 2017)

In principle, we can retrain a deep network for each new task it encounters. However, this approach has several drawbacks. Retraining requires long interruptions and specialized datasets or simulations, requirements which many realistic settings do not support. Additionally, we may not be able to distinguish or define new tasks, making it difficult to determine when to retrain and what task to retrain on. Finally, non-

stationary approaches to training neural networks are prone to various failure modes (French 1999; Kirkpatrick et al. 2017)

1.2 Meta-Learning

These concerns have reinvigorated a subdiscipline called "meta-learning" (Schmidhuber 1987) within the deep learning community. Meta-learning produces learning algorithms which acquire speed through specialization to a setting, similar to earlier optimization algorithms. Unlike these algorithms which commit to a priori assumptions, meta-learning discovers the properties of its setting through trial-and-error interaction. It then uses these properties to generate fast, on-the-fly learning algorithms.

In general, meta-learning algorithms have a hierarchical structure in which an *inner-loop* optimizes performance on a specific task and an *outer-loop* optimizes the learning procedure of the inner-loop. Some approaches use gradient updates for both levels (Finn et al. 2017; Stadie et al. 2018), while others use gradients only for the outer-loop and train some kind of learned update rule for the inner-loop.

1.3 In-Context Learning

A common approach to learning an update rule involves learning some representation of each new experience of the task, along with the parameters of some operator that aggregates these representations. For example, the RL^2 (Duan et al. 2016) algorithm uses the Long-Short Term Memory (LSTM, (Hochreiter and Schmidhuber 1997)) architecture for aggregation. Others (Team et al. 2023) have used Transformers (Vaswani

et al. 2017a) in place of LSTMs. PEARL (Rakelly et al. 2019) uses a latent context that accumulates representations of the agent's observations in a product of Gaussian factors.

We call this aggregated representation *memory* or *context*. In many settings, one observes a rapid increase in performance as new experiences accumulate in the context of a neural architecture. This increase in performance is what we call *in-context learning*.

In the context of meta-learning, in-context learning corresponds to the inner-loop, which rapidly adapts the learner to a specific task. However, not all in-context learning uses the inner/outer-loop meta-learning formulation. Famously, GPT-3 (Brown et al. 2020a) demonstrated that a large language model developed the ability to learn in-context as an emergent consequence of large-scale training. Another interesting example of in-context learning outside of meta-learning is Algorithm Distillation (Laskin et al. 2022), which demonstrates that a transformer trained on offline RL data can distill and reproduce improvement operators from the algorithm that generated the data.

1.4 Meta Reinforcement Learning

Meta reinforcement learning (meta-RL) is a subdiscipline of meta-learning which focuses on reinforcement learning (RL) tasks. Meta-RL typically targets multi-task settings in which the agent cannot infer the current task from any single observation, but must discover its properties through exploration. Existing literature (Finn et al. 2017;

Rakelly et al. 2019; Zintgraf et al. 2019) typically evaluates meta-RL agents in special multi-trial episodes in which the agent's state resets once per trial and multiple times per episode, but the task remains the same. The agent must maximize cumulative or final reward per episode, with optimal performance requiring the agent to exploit in later trials information that was discovered in earlier trials.

This setting demands a different strategy than multi-task RL, in which a fully trained agent exploits a near-optimal policy, even in held-out evaluation settings. A meta-RL agent must learn to explore initially and later transition to exploitation. For example, in a gridworld, efficient exploration may entail a circuitous search policy that never revisits any state more than once. Meanwhile an exploitation policy moves in a straight path toward some goal, which earlier exploration has revealed.

1.5 Meta-Learning Challenges

The outer/inner-loop meta-learning formulation is powerful and general. However, the approach has difficulty scaling to complex problems with large task spaces. One of the difficulties inherent in this form of meta-learning is the coupling of the inner- and outer-loop optimizations. This causes instability in the learning process and forces the meta-learning algorithm to search not only the space of task solutions, but also the space of learning algorithms that might produce each solution. When signal is sparse or the task space is large, the outer loop will often fail to make progress.

Meta-RL algorithms are also susceptible to a local minimum in which they fail to recognize that the information yielded from exploration can inform an efficient exploita-

tion policy, remaining in a permanent suboptimal exploration mode. In our gridworld setting, the agent might fail to recognize that the goal appears in the same location in every trial of an episode, repeatedly searching for it instead of moving toward it efficiently. AdA (Team et al. 2023) is one of the only meta-RL works to have scaled this approach to complex domains, but only with the help of sophisticated curriculum techniques which may not apply to all settings.

1.6 An Alternative Approach to In-Context Learning

This thesis explores a meta-RL in-context-learning methodology which decouples the inner- and outer-loop by sending learning signal in one direction only, from the meta-process to the inner process, and from an *external dataset* to the meta-process. Without feedback from the inner process, the meta-process is unable to completely specialize to the downstream, inner-process setting, which limits its ability to match traditional meta-learning in terms of downstream efficiency, but yields other benefits. In particular, we note that our methods can leverage large pre-existing offline datasets but unlike existing offline RL techniques (Fujimoto et al. 2019; Kumar et al. 2020), they permit further interaction with and exploration of the environment. We defer the exact details of our methods to later chapters, but offer a cursory sketch at this point.

1.6.1 Value Estimation

We use a model trained on offline data to make context-conditional predictions, that we use to estimate state-action values. This stage of training involves standard gra-

dient-based optimization and is analogous to outer-loop optimization in a traditional meta-learning algorithm, insofar as this stage distills priors in the model that support rapid downstream learning. The inputs to the model contain information relating to the environment dynamics, the reward function, and the current policy — the parameters of a value estimate — and we train the model to condition its predictions on this information. We explore various techniques to encourage the model to attend to this context (in-context learning) as opposed to resorting to priors encoded in its weights (in-weights learning). Chan et al. (2022b) provides further discussion of this distinction. The model will then demonstrate some capability to generalize its predictions to unseen downstream settings, as long as those settings are adequately represented in the inputs and these inputs are similarly distributed to the inputs in the training dataset.

1.6.2 Policy Improvement

We target a meta-RL setting in which the agent is unable to perform optimally at the start of an episode, due to limited knowledge of the task and environment. Our method therefore requires some mechanism for improving the policy as information accumulates through interaction. This stage of training does not use gradients and is analogous to the inner-loop of a traditional meta-learning algorithm, insofar as it adapts the learner to a specific task. To induce policy improvement, our method combines the classic policy iteration algorithm with in-context learning techniques. At each timestep, we use the method described in the previous paragraph to estimate the state-action value

of each action in the action space. We then choose the action with the highest value estimate.

In general, policy iteration depends on a cycle in which actions are chosen greedily with respect to value estimates and value estimates are updated to track the newly improved policy. Our action selection method satisfies the first half of this formulation. To satisfy the second, we condition our model's predictions exclusively on data drawn from recent interactions with the environment, which reflect or approximate the current policy. As the policy improves, the behavioral data will capture this improvement, the model's predictions will reflect the improvement through this data, and the greedy action selection strategy will produce an improvement cycle.

1.7 Summary of Chapters

1.7.1 In-Context Policy Iteration

Our first chapter explores a method that bootstraps a pre-existing large language model — Codex (Chen et al. 2021a) — to produce value estimates. This work restricts itself to a set of toy domains that can be translated into text. Instead of predicting value directly, a form of quantitative reasoning that would strain the capabilities of a language model, our method uses simulated rollouts to generate monte-carlo estimations of value, similar to Model-Predictive Control (Pan et al. 2022). The work presents a series of prompting techniques for eliciting these predictions from a pre-trained language model.

Continuing our analogy to traditional meta-learning, the outer-loop corresponds to the pretraining of Codex. Of the three methods that we present in this work, this is the purest instantiation of the "one-way learning signal" formulation that we posited earlier, since the dataset on which Codex was trained was completely agnostic to the downstream tasks on which we evaluate our method. Nevertheless, we present evidence that our method leverages commonsense pattern completion and numerical reasoning distilled in Codex's weights.

1.7.2 Algorithm Distillaion + Model-Based Planning

Our second chapter extends the first by training a new model from offline reinforcement learning (RL) data instead of using an existing pre-trained language model. We demonstrate that learning planning primitives from the offline data helps speed up training and facilitate generalization to novel dynamics and reward functions. Additionally, we demonstrate that in-context policy iteration can be used in conjunction with Algorithm Distillation (Laskin et al. 2022), superimposing the policy improvement operators induced by both methods.

1.7.3 Bellman Update Networks

Our final chapter, in which we propose future work, describes an alternative to the monte-carlo rollout estimation technique used in the other two chapters. This chapter advocates a method which directly optimizes the accuracy of value predictions, instead of optimizing surrogate world-model predictions. We initially present a naive method

for estimating values with a single inference step, but contrast it with an iterative approach that learns to approximate individual Bellman updates and generates estimates through repeated applications of the model.

2 In-Context Policy Iteration

In the context of Large Language Models (LLMs), in-context learning describes the ability of these models to generalize to novel downstream tasks when prompted with a small number of exemplars (Brown et al. 2020b; Lu et al. 2021). The introduction of the Transformer architecture (Vaswani et al. 2017b) has significantly increased interest in this phenomenon, since this architecture demonstrates much stronger generalization capacity than its predecessors (Chan et al. 2022a). Another interesting property of in-context learning in the case of large pre-trained models (or "foundation models") is that the models are not directly trained to optimize a meta-learning objective, but demonstrate an emergent capacity to generalize (or at least specialize) to diverse downstream task-distributions (Brown et al. 2020b; Chan et al. 2022a; Lu et al. 2021; Wei et al. 2022). A litany of existing work has explored methods for applying this remarkable capability to downstream tasks (see Section 2.1), including Reinforcement Learning (RL). Most work in this area either (1) assumes access to expert demonstrations — collected either from human experts (Baker et al. 2022; Huang et al. 2022a), or domain-specific pre-trained RL agents (Chen et al. 2021b; Janner et al. 2021a; Lee et al. 2022; Reed et al. 2022; Xu et al. 2022) — or (2) relies on gradient-based methods — e.g. fine-tuning of the foundation models parameters as a whole (Baker et al. 2022; Lee et al. 2022; Reed et al. 2022) or newly training an adapter layer or prefix vectors while keeping the

original foundation models frozen (Karimi Mahabadi et al. 2022; Li and Liang 2021; Singh et al. 2022).

Our work presents an algorithm, In-Context Policy Iteration (ICPI) which relaxes these assumptions. ICPI is a form of policy iteration in which the prompt content is the locus of learning. Because our method operates on the prompt itself rather than the model parameters, we are able to avoid gradient methods. Furthermore, the use of policy iteration frees us from expert demonstrations because suboptimal prompts can be improved over the course of training.

We illustrate the algorithm empirically on six small illustrative RL tasks — *chain, distractor-chain, maze, mini-catch, mini-invaders*, and *point-mass* — in which the algorithm very quickly finds good policies. We also compare five pretrained Large Language Models (LLMs), including two different size models trained on natural language — OPT-30B and GPT-J — and three different sizes of a model trained on program code — two sizes of Codex as well as InCoder. On our six domains, we find that only the largest model (the `code-davinci-001` variant of Codex) consistently demonstrates learning.

2.1 Related Work

A common application of foundation models to RL involves tasks that have language input, for example natural language instructions/goals (Garg et al. 2022; Hill et al. 2020) or text-based games (Ammanabrolu and Riedl 2021; Majumdar et al. 2020; Peng et al. 2021; Singh et al. 2021). Another approach encodes RL trajectories into token

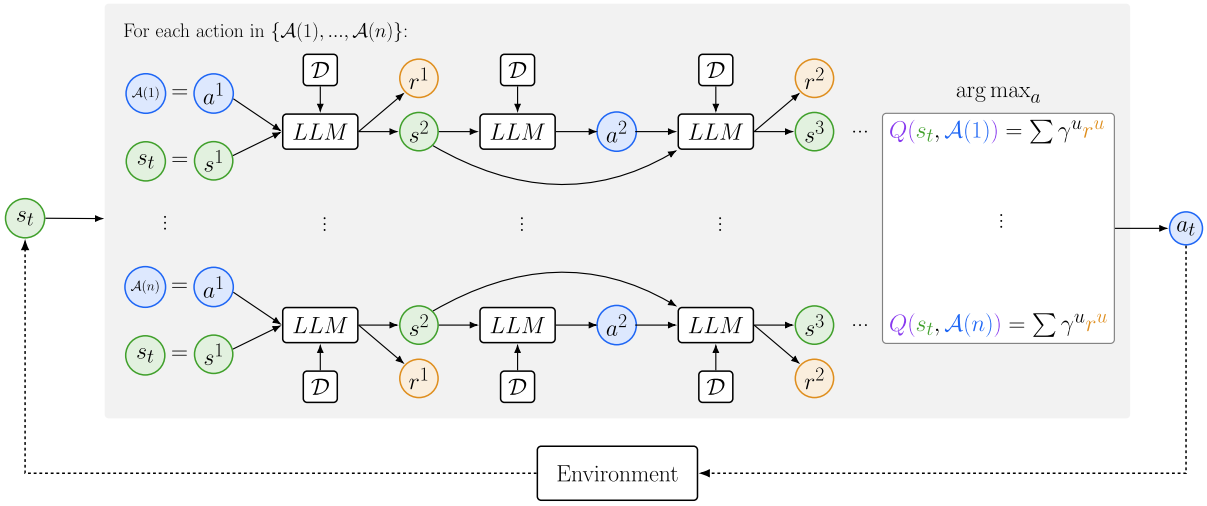


Figure 1: Diagram of procedure used to generate rollouts and select actions.

sequences, and processes them with a foundation model, and passes the model outputs to deep RL architectures (Li et al. 2022; Tam et al. 2022; Tarasov et al. 2022). Finally, a recent set of approaches (which we will focus on in this Related Work section) treat RL as a sequence modeling problem and use the foundation model itself to predict states or actions. In this related work section, we will focus on a third set of recent approaches that treat reinforcement learning (RL) as a sequence modeling problem and utilize foundation models for state prediction, action selection, and task completion. We will organize our survey of these approaches based on how they elicit these RL-relevant outputs from the foundation models. In this respect the approaches fall under three broad categories: learning from demonstrations, specialization (via training or finetuning), and context manipulation (in-context learning).

2.1.1 Learning from demonstrations

Many recent sequence-based approaches to reinforcement learning use demonstrations that come either from human experts or pretrained RL agents. For example, Huang et

al. (2022a) use a frozen LLM as a planner for everyday household tasks by constructing a prefix from human-generated task instructions, and then using the LLM to generate instructions for new tasks. This work is extended by Huang et al. (2022b). Similarly, Ahn et al. (2022) use a value function that is trained on human demonstrations to rank candidate actions produced by an LLM. Baker et al. (2022) use human demonstrations to train the foundation model itself: they use video recordings of human Minecraft players to train a foundation model to play Minecraft. Works that rely on pretrained RL agents include Janner et al. (2021a) who train a "Trajectory Transformer" to predict trajectory sequences in continuous control tasks by using trajectories generated by pre-trained agents, and Chen et al. (2021b), who use a dataset of offline trajectories to train a "Decision Transformer" that predicts actions from state-action-reward sequences in RL environments like Atari. Two approaches build on this method to improve generalization: Lee et al. (2022) use trajectories generated by a DQN agent to train a single Decision Transformer that can play many Atari games, and Xu et al. (2022) use a combination of human and artificial trajectories to train a Decision Transformer that achieves few-shot generalization on continuous control tasks. Reed et al. (2022) take task-generality a step farther and use datasets generated by pretrained agents to train a multi-modal agent that performs a wide array of RL (e.g. Atari, continuous control) and non-RL (e.g. image captioning, chat) tasks.

Some of the above works include non-expert demonstrations as well. Chen et al. (2021b) include experiments with trajectories generated by random (as opposed to expert) policies. Lee et al. (2022) and Xu et al. (2022) also use datasets that include tra-

jectories generated by partially trained agents in addition to fully trained agents. Like these works, our proposed method (ICPI) does not rely on expert demonstrations—but we note two key differences between our approach and existing approaches. Firstly, ICPI only consumes self-generated trajectories, so it does not require any demonstrations (like Chen et al. (2021b) with random trajectories, but unlike Lee et al. (2022), Xu et al. (2022), and the other approaches reviewed above). Secondly, ICPI relies primarily on in-context learning rather than in-weights learning to achieve generalization (like Xu et al. (2022), but unlike Chen et al. (2021b) & Lee et al. (2022)). For discussion about in-weights vs. in-context learning see Chan et al. (2022a).

2.1.2 Gradient-based training & finetuning on RL tasks

Many approaches that use foundation models for RL involve specifically training or fine-tuning on RL tasks. For example, Janner et al. (2021a), Chen et al. (2021b), Lee et al. (2022), Xu et al. (2022), Baker et al. (2022), and Reed et al. (2022) all use models that are trained from scratch on tasks of interest, and (Singh et al. 2022), Ahn et al. (2022), and Huang et al. (2022b) combine frozen foundation models with trainable components or adapters. In contrast, Huang et al. (2022a) use frozen foundation models for planning, without training or fine-tuning on RL tasks. Like Huang et al. (2022a), ICPI does not update the parameters of the foundation model, but relies on the frozen model's in-context learning abilities. However, ICPI gradually builds and improves the prompts within the space defined by the given fixed text-format for observations, ac-

tions, and rewards (in contrast to Huang et al. (2022a), which uses the frozen model to select good prompts from a given fixed library of goal/plan descriptions).

2.1.3 In-Context learning

Several recent papers have specifically studied in-context learning. Laskin et al. (2022) demonstrates an approach to performing in-context reinforcement learning by training a model on complete RL learning histories, demonstrating that the model actually distills the improvement operator of the source algorithm. Min et al. (2022) demonstrates that LLMs can learn in-context, even when the labels in the prompt are randomized, problemetizing the conventional understanding of in-context learning and showing that label distribution is more important than label correctness. Chan et al. (2022a) and Garg et al. (2022) provide analyses of the properties that drive in-context learning, the first in the context of image classification, the second in the context of regression onto a continuous function. These papers identify various properties, including "burstiness," model-size, and model-architecture, that in-context learning depends on. Chen et al. (2022) studies the sensitivity of in-context learning to small perturbations of the context. They propose a novel method that uses sensitivity as a proxy for model certainty. Some recent work has explored iterative forms of in-context learning, similar to our own. For example, Shinn et al. (2023) and Madaan et al. (2023) use iterative self-refinement to improve the outputs of a large language model in a natural language context. These approaches rely on the ability of the model to examine and critique its own outputs, rather than using policy iteration as our method does.

2.2 Background

2.2.1.1 Markov Decision Processes

A Markov Decision Process (MDP) is a problem formulation in which an agent interacts with an environment through actions, receiving rewards for each interaction. Each action transitions the environment from some state to some other state and the agent is provided observations which depend on this state (and from which the state can usually be inferred). MDPs can be "fully" or "partially" observed. A fully observed MDP is defined by the property that the distribution for each transition and reward is fully conditioned on the current observation. In a partially observed MDP, the distribution depends on history — some observations preceding the current one. The goal of reinforcement learning is to discover a "policy" — a mapping from observations to actions — which maximizes cumulative reward over time.

2.2.1.2 Model-Based Planning

A model is a function which approximates the transition and reward distributions of the environment. Typically a model is not given but must be learned from experience gathered through interaction with the environment. Model-Based Planning describes a class of approaches to reinforcement learning which use a model to choose actions. These approaches vary widely, but most take advantage of the fact that an accurate model enables the agent to anticipate the consequences of an action before taking it in the environment.

Algorithm 1: Training Loop

initialize $\mathcal{D}$ ▷ Replay Buffer
while training **do**
  $x_0 \leftarrow$ Reset environment ▷ Get initial state
  for each timestep t in episode **do**
    $a \leftarrow \arg\max_a Q^\pi(x_t, a)$ ▷ Choose action with highest value
    $r_t, b_t, x_{t+1} \leftarrow$ step environment with a ▷ Receive reward and next state
    $\mathcal{D} \leftarrow \mathcal{D} \cup (x_t, a_t, r_t, b_t)$ ▷ Add interaction to replay buffer
  end for
end while

2.2.1.3 Policy Iteration

Policy iteration is a technique for improving a policy in which one estimates the values for the current policy and then chooses actions greedily with respect to these value estimates. This yields a new policy which is at least as good as the original policy according to the policy improvement theorem (assuming that the value estimates are correct). This process may be repeated indefinitely until convergence. To choose actions greedily with respect to the value estimates, there must be some mechanism for estimating value conditioned not only on the current state and policy but also on an arbitrary action. The "greedy" action choice corresponds to the action with the highest value estimate. Policy iteration is possible if our value estimates are unbiased for any state-action pair and for the current policy.

2.3 Method

How can standard policy iteration make use of in-context learning? Policy iteration is either model-based (Section 2.2.1.2) — using a world-model to plan future trajectories in the environment — or model-free — inferring value-estimates without explicit

Algorithm 2: Computing Q-values

function $Q(x_t, a, \mathcal{D})$
  $u \leftarrow t$
  $x^1 \leftarrow x_t$
  $a^1 \leftarrow a_t$
  repeat
    $\mathcal{D}_b \sim$ time-steps with action a^u
    $b^u \sim \text{LLM}(\mathcal{D}_b, x^u, a^u)$ ▷ model termination
    $\mathcal{D}_r \sim$ time-steps with action a^u and termination b^u
    $r^u \sim \text{LLM}(\mathcal{D}_b, x^u, a^u)$ ▷ model reward
    $\mathcal{D}_x \sim$ time-steps with action a^u and termination b^u
    $x^{u+1} \sim \text{LLM}(\mathcal{D}_x, x^u, a^u)$ ▷ model termination
    $\mathcal{D}_a \sim c$ recent trajectories
    $a^{u+1} \sim \text{LLM}(\mathcal{D}_a, x^{u+1})$ ▷ model policy
    $u \leftarrow u + 1$
  until b^u is terminal ▷ model predicts termination
  $Q^\pi(x_t, a) = \sum_{u=t}^{T} \gamma^{u-t} r^u$ ▷ Estimate value from rollout
end function

planning. Both methods can be realized with in-context learning. We choose model-based learning because planned trajectories make the underlying logic of value-estimates explicit to our foundation model backbone, providing a concrete instantiation of a trajectory that realizes the values. This ties into recent work (Nye et al. 2021; Wei et al. 2022) demonstrating that "chains of thought" can significantly improve few-shot performance of foundation models.

Model-based RL requires two ingredients, a rollout-policy used to act during planning and a world-model used to predict future rewards, terminations, and states. Since our approach avoids any mutation of the foundation model's parameters (this would require gradients), we must instead induce the rollout-policy and the world-model using in-context learning, i.e. by selecting appropriate prompts. We induce the rollout-policy by prompting the foundation model with trajectories drawn from the

current (or recent) behavior policy (distinct from the rollout-policy). Similarly, we induce the world-model by prompting the foundation models with transitions drawn from the agent's history of experience. Note that our approach assumes access to some translation between the state-space of the environment and the medium (language, images, etc.) of the foundation models. This explains how an algorithm might plan and estimate values using a foundation model. It also explains how the rollout-policy approximately tracks the behavior policy.

How does the policy improve? When acting in the environment (as opposed to planning), we choose the action that maximizes the estimated Q-value from the current state (see Algorithm 1, line 5). At time step t, the agent observes the state of the environment (denoted x_t) and executes action $a_t = \arg\max_{a \in \mathcal{A}} Q^{\pi_t}(x_t, a)$, where $\mathcal{A} = [\mathcal{A}(1), ..., \mathcal{A}(n)]$ denotes the set of n actions available, π_t denotes the policy of the agent at time step t, and Q^π denotes the Q-estimate for policy π. Taking the greedy ($\arg\max$) actions with respect to Q^{π_t} implements a new and improved policy.

2.3.1 Computing Q-values

This section provides details on the prompts that we use in our computation of Q-values (see Algorithm 2 pseudocode & Figure 1 rollout). During training, we maintain a buffer $\mathcal{D}$ of transitions experienced by the agent. To compute $Q^{\pi_t}(x_t, a)$ at time step t in the real-world we rollout a simulated trajectory $x^1 = x_t,\ a^1 = a,$ $r^1,\ x^2,\ a^2,\ r^2,\ ...,\ x^T,\ a^T,\ r^T,\ x^{T+1}$ by predicting, at each simulation time step u: reward $r^u \sim \text{LLM}(\mathcal{D}_r, x^u, a^u)$; termination $b^u \sim \text{LLM}(\mathcal{D}_b, x^u, a^u)$; observation

$x^{u+1} \sim \text{LLM}(\mathcal{D}_x, x^u, a^u)$; action $a^1 \sim \text{LLM}(\mathcal{D}_a, x^u)$. Termination b^u decides whether the simulated trajectory ends at step u.

The prompts $\mathcal{D}_r$, $\mathcal{D}_b$ contain data sampled from the replay buffer. For each prompt, we choose some subset of replay buffer transitions, shuffle them, convert them to text (examples are provided in table Section 2.4.1) and clip the prompt at the 4000-token Codex context limit. We use the same method for $\mathcal{D}_a$, except that we use random trajectory subsequences.

In order to maximize the relevance of the prompt contents to the current inference we select transitions using the following criteria. $\mathcal{D}_b$ contains (x_k, a_k, b_k) tuples such that a_k equals a^u, the action for which the LLM must infer termination. $\mathcal{D}_r$ contains (x_k, a_k, r_k) tuples, again constraining $a_k = a^u$ but also constraining $b_k = b^k$—that the tuple corresponds to a terminal time-step if the LLM inferred $b^u = \text{true}$, and to a non-terminal time-step if $b^u = \text{false}$. For $\mathcal{D}_x$, the prompt includes $(x_k, a_k x_k + 1)$ tuples with $a_k = a^u$ and $b_k = \text{false}$ (only non-terminal states need to be modelled).

We also maintain a balance of certain kinds of transitions in the prompt. For termination prediction, we balance terminal and non-terminal time-steps. Since non-terminal time-steps far outnumber terminal time-steps, this eliminates a situation wherein the randomly sampled prompt time-steps are entirely non-terminal, all but ensuring that the LLM will predict non-termination. Similarly, for reward prediction, we balance the number of time-steps corresponding to each reward value stored in $\mathcal{D}$. In order to balance two collections of unequal size, we take the smaller and duplicate randomly chosen members until the sizes are equal.

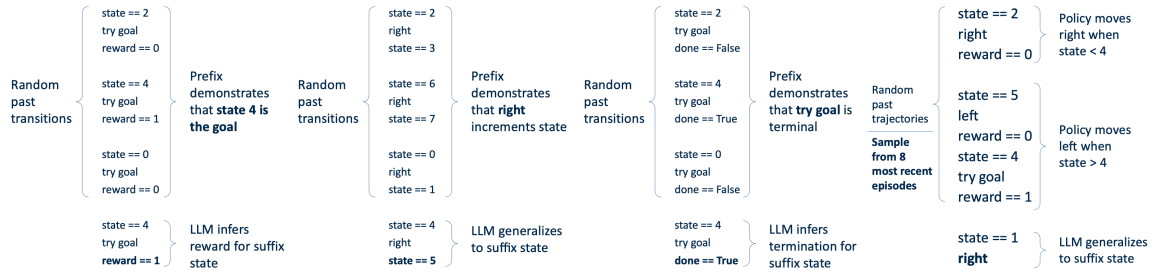


Figure 2: Illustration of prompts used to elicit the four models: from left to right, the reward model, the transition model, the termination model, and the policy model.

In contrast to the other predictions, we condition the rollout policy on trajectory subsequences, not individual time-steps. Prompting with sequences better enables the foundation model to apprehend the logic behind a policy. Trajectory subsequences consist of (x_k, a_k) pairs, randomly clipped from the c most recent trajectories. More recent trajectories will, in general, demonstrate higher performance, since they come from policies that have benefited from more rounds of improvement.

Finally, the Q-value estimate is simply the discounted sum of rewards for the simulated episode. For examples of these prompts, see Figure 2. Given this description of Q-value estimation, we now return to the concept of policy improvement.

2.3.2 Policy-Improvement

The $\arg\max$ (line 5 of Algorithm 1) drives policy improvement in Algorithm. Critically this is not simply a one-step improvement but a mechanism that builds improvement on top of improvement. This occurs through a cycle in which the $\arg\max$ improves behavior. The improved behavior is stored in the buffer $\mathcal{D}$, and then used to condition the rollout policy. This improves the returns generated by the LLM during planning rollouts. These improved rollouts improve the Q-estimates for each action. Completing

Figure 7: Diagram of the policy improvement cycle. The replay buffer is fed by a behavior policy. Next we sample the trajectories used in the policy prompt from recent episodes in the buffer. By sampling recent episodes only, we try to keep the prompt policy close to the behavior policy. Next, we rely on the language model to infer the distribution of actions in the prompt and yield a similarly distributed action for our rollout policy. Therefore the rollout policy approximates the prompt policy. Finally, by choosing actions greedily with respect to our value estimates, we implement a new behavior policy that improves the rollout policy.

the cycle, this improves the actions chosen by the arg max. Because this process feeds into itself, it can drive improvement without bound until optimality is achieved. See Figure 7 for an illustration of this cycle.

Note that this process takes advantage of properties specific to in-context learning. In particular, it relies on the assumption that the rollout policy, when prompted with trajectories drawn from a mixture of policies, will approximate something like an average of these policies. Given this assumption, the rollout policy will improve with the improvement of the mixture of policies from which its prompt-trajectories are drawn. This results in a kind of rapid policy improvement that works without any use of gradients.

2.3.3 Prompt-Format

The LLM cannot take non-linguistic prompts, so our algorithm assumes access to a textual representation of the environment—of states, actions, terminations, and rewards—and some way to recover the original action, termination, and reward values from

their textual representation (we do not attempt to recover states). Since our primary results use the Codex language model (see Table 8), we use Python code to represent these values (examples are available in Figure 9).

In our experiments, we discovered that the LLM world-model was unable to reliably predict rewards, terminations, and next-states on some of the more difficult environments. We experimented with providing domain emphhints in the form of prompt formats that make explicit useful information — similar to Chain of Thought Prompting . For example, for the emphchain domain, the hint includes an explicit comparison (`==` or `!=`) of the current state with the goal state. Note that while hints are provided in the initial context, the LLM must infer the hint content in rollouts generated from this context.

We use a consistent idiom for rewards and terminations, namely `assert reward ==` `x` and `assert done` or `assert not done`. Some decisions had to be made when representing states and actions. In general, we strove to use simple, idiomatic, concise Python. On the more challenging environments, we did search over several options for the choice

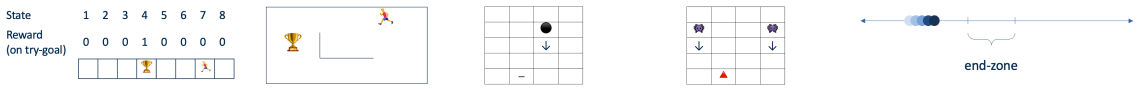
Model	**Parameters**	**Training Data**
GPT-J (Wang and Komatsuzaki 2021)	6 billion	"The Pile" (Gao et al. 2020), an 825GB English corpus incl. Wikipedia, GitHub, academic pubs
InCoder cite()	6.7 billion	159 GB of open-source StackOverflow code
OPT-30B cite()	30 billion	180B tokens of predominantly English data
Codex cite()	185 billion	179 GB of GitHub code

Table 8: Table of models and training data.

Chain	`assert state == 6` *`and state != 4`* `state = left()` `assert reward == 0` `assert not done`
Distractor	`assert state == (6, 3)` *`and state != (4, 3)`* `state = left()` `assert reward == 0` `assert not done`
Maze	`assert state == C(i=2, j=1)` *`and state != C(i=1, j=0)`* `state, reward = left()` `assert reward == 0` `assert not done`
Mini Catch	`assert paddle == C(2, 0)` *`and ball == C(0, 4)`* *`and paddle.x == 2 and ball.x == 0`* *`and paddle.x > ball.x`* *`and ball.y == 4`* `reward = paddle.left()` `ball.descend()` `assert reward == 0` `assert not done`
Mini Invaders	`assert ship == C(2, 0) and aliens == [C(3, 5), C(1, 5)]` *`and (ship.x, aliens[0].x, aliens[1].x) == (2, 3, 1)`* *`and ship.x < aliens[0].x`* *`and ship.x > aliens[1].x`* `ship.left()` `assert reward == 0` `for a in aliens:` `  a.descend()` `assert not done`
Point-Mass	`assert pos == -3.45 and vel == 0.00` *`and pos < -2 and vel == 0`* `pos, vel = decel(pos, vel)` `assert reward == 0` `assert not done`

Figure 9: Example prompts for each domain, showcasing the text format and hints (hints in italics).

of hint. For examples, see Figure 9. We anticipate that in the future, stronger foundation models will be increasingly robust to these decisions.



Chain Maze Mini-Catch Mini-Invaders Point-Mass

Figure 10: Illustration of the five of the domains used in our experiments. Distractor-Chain is omitted for brevity.

2.4 Experiments

We have three main goals in our experiments: (1) Demonstrate that the agent algorithm can in fact quickly learn good policies, using pretrained LLMs, in a set of six simple illustrative domains of increasing challenge; (2) provide evidence through an ablation that the policy-improvement step — taking the $\arg\max$ over Q-values computed through LLM rollouts — accelerates learning; and (3) investigate the impact of using different LLMs (see Table 8) — different sizes and trained on different data, in particular, trained on (mostly) natural language program code (Codex and InCoder). We next describe the six domains and their associated prompt formats, and then describe the experimental methodology and results.

2.4.1 Domains and prompt format

For a visual illustration of each domain, see Figure 10.

Chain: In this environment, the agent occupies an 8-state chain. The agent has three actions: `Left`, `right`, and `try goal`. The `try goal` action always terminates the episode, conferring a reward of 1 on state 4 (the goal state) and 0 on all other states. Because this environment has simpler transitions than the other two, we see the clearest evidence of learning here. Note that the initial batch of successful trajectories collected from random behavior will usually be suboptimal, moving inefficiently toward the goal

state. We include a discount value of 0.8 in our diagram to show the improvement in efficiency of the policy learned by the agent over the course of training. `Prompt format.` Episodes also terminate after 8 time-steps. States are represented as numbers from 0 to 7, as in `assert state == n`, with the appropriate integer substituted for `n`. The actions are represented as functions `left()`, `right()`, and `try_goal()`. For the hint, we simply indicate whether or not the current state matches the goal state, 4.

Distractor Chain: This environment is an 8-state chain, identical to the *chain* environment, except that the observation is a *pair* of integers, the first indicating the true state of the agent and the second acting as a distractor which transitions randomly within $\{0, ..., 7\}$. The agent must therefore learn to ignore the distractor integer and base its inferrences on the information contained in the first integer. Aside from the addition of this distractor integer to the observation, all text representations and hints are identical to the *chain* environment.

Maze: The agent navigates a small 3×3 gridworld with obstacles. The agent can move `up`, `down`, `left`, or `right`. The episode terminates with a reward of 1 once the agent navigates to the goal grid, or with a reward of 0 after 8 time-steps. This environment tests our algorithm's capacity to handle 2-dimensional movement and obstacles, as well as a 4-action state-space. We represent the states as namedtuples — `C(x, y)`, with integers substituted for `x` and `y`. Similar to *chain*, the hint indicates whether or not the state corresponds to the goal state.

Mini Catch: The agent operates a paddle to catch a falling ball. The ball falls from a height of 5 units, descending one unit per time step. The paddle can `stay` in

place (not move), or move `left` or `right` along the bottom of the 4-unit wide plane. The agent receives a reward of 1 for catching the ball and 0 for other time-steps. The episode ends when the ball's height reaches the paddle regardless of whether or not the paddle catches the ball. We chose this environment specifically to challenge the action-inference/rollout-policy component of our algorithm. Specifically, note that the success condition in Mini Catch allows the paddle to meander before moving under the ball —as long as it gets there on the final time-step. Successful trajectories that include movement *away* from the ball thus making good rollout policies more challenging to learn (i.e., elicit from the LLM via prompts). Again, we represent both the paddle and the ball as namedtuples `C(x, y)` and we represent actions as methods of the `paddle` object: `paddle.stay()`, `paddle.left()`, and `paddle.right()`. For the hint, we call out the location of the paddle's x-position, the ball's x-position, the relation between these positions (which is larger than which, or whether they are equal) and the ball's y-position. Figure 9 provides an example. We also include the text `ball.descend()` to account for the change in the ball's position between states.

Mini Invaders: The agent operates a ship that shoots down aliens which descend from the top of the screen. At the beginning of an episode, two aliens spawn at a random location in two of four columns. The episode terminates when an alien reaches the ground (resulting in 0 reward) or when the ship shoots down both aliens (the agent receives 1 reward per alien). The agent can move `left`, `right`, or `shoot`. This domain highlights ICPI's capacity to learn incrementally, rather than discovering an optimal policy through random exploration and then imitating that policy, which is how our

"No ArgMax" baseline learns (see Section 2.4.3). ICPI initially learns to shoot down one alien, and then builds on this good but suboptimal policy to discover the better policy of shooting down both aliens. In contrast, random exploration takes much longer to discover the optimal policy and the "No ArgMax" baseline has only experienced one or two successful trajectories by the end of training.

We represent the ship by its namedtuple coordinate (`C(x, y)`) and the aliens as a list of these namedtuples. When an alien is shot down, we substitute `None` for the tuple, as in `aliens == [C(x, y), None]`. We add the text: `for a in aliens: a.descend()`, in order to account for the change in the alien's position between states.

Point-Mass: A point-mass spawns at a random position on a continuous line between -6 and $+6$ with a velocity of 0. The agent can either `accelerate` the point-mass (increase velocity by 1) or `decelerate` it (decrease the velocity by 1). The point-mass position changes by the amount of its velocity each timestep. The episode terminates with a reward of 1 once the point-mass is between -2 and $+2$ and its velocity is 0 once again. The episode also terminates after 8 time-steps. This domain tests the algorithm's ability to handle continuous states.

States are represented as `assert pos == p and vel == v`, substituting floats rounded to two decimals for `p` and `v`. The actions are `accel(pos, vel)` and `decel(pos, vel)`. The hint indicates whether the success conditions are met, namely the relationship of `pos` to -2 and $+2$ and whether or not `vel == 0`. The hint includes identification of the aliens' and the ship's x-positions as well as a comparison between them.

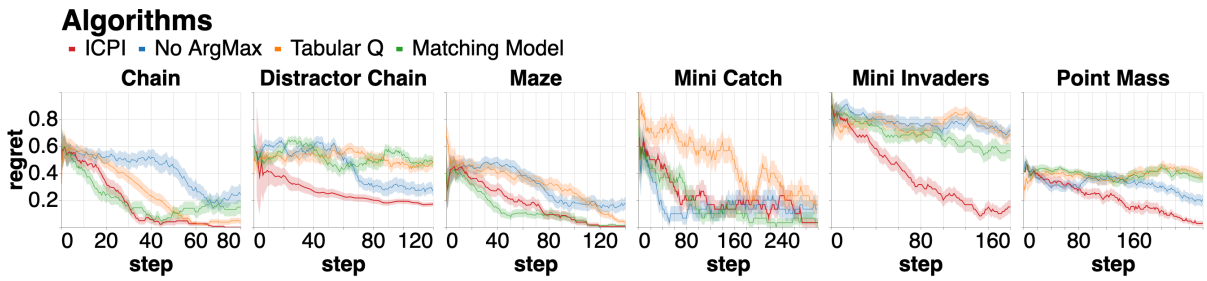


Figure 16: Comparison of ICPI with three baselines.

2.4.2 Methodology and Evaluation

For the results, we record the agent's regret over the course of training relative to an optimal policy computed with a discount factor of 0.8. For all experiments $c = 8$ (the number of most recent successful trajectories to include in the prompt). We did not have time for hyperparameter search and chose this number based on intuition. However, the $c = 16$ baseline demonstrates results when this hyperparameter is doubled. All results use 4 seeds.

For both versions of Codex, we used the OpenAI Beta under the API Terms of Use. For GPT-J (Wang and Komatsuzaki 2021) , InCoder (Fried et al. 2022) and OPT-30B (Zhang et al. 2022), we used the open-source implementations from Huggingface Transformers (Wolf et al. 2020), each running on one Nvidia A40 GPU. All language models use a sampling temperature of 0.1. Code for our implementation is available at https://github.com/ethanabrooks/icpi.

2.4.3 Comparison of ICPI with baseline algorithms.

We compare ICPI with three baselines (Figure 16).

The "No ArgMax" baseline learns a good policy through random exploration and then imitates this policy. This baseline assumes access to a "success threshold" for

each domain — an undiscounted cumulative return greater than which a trajectory is considered successful. The action selection mechanism emulates ICPI's rollout policy: prompting the LLM with a set of trajectories and eliciting an action as output. For this baseline, we only include trajectories in the prompt whose cumulative return exceeds the success threshold. Thus the policy improves as the number of successful trajectories in the prompt increases over time. Note that at the start of learning, the agent will have experienced too few successful trajectories to effectively populate the policy prompt. In order to facilitate exploration, we act randomly until the agent experiences 3 successes.

"Tabular Q" is a standard tabular Q-learning algorithm, which uses a learning rate of 1.0 and optimistically initializes the Q-values to 1.0.

"Matching Model" is a baseline which uses the trajectory history instead of an LLM to perform modelling. This baseline searches the trajectory buffer for the most recent instance of the current state, and in the case of transition/reward/termination prediction, the current action. If a match is found, the model outputs the historical value (e.g. the reward associated with the state-action pair found in the buffer). If no match is found, the modelling rollout is terminated. Recall that ICPI breaks ties randomly during action selection so this will often lead to random action selection.

As our results demonstrate, only ICPI learns good policies on all domains. We attribute this advantage to ICPI's ability to generalize from its context to unseen states and state/action pairs (unlike "Tabular Q" and "Matching Model"). Unlike "No ArgMax," ICPI is able to learn progressively, improving the policy before experiencing good trajectories.

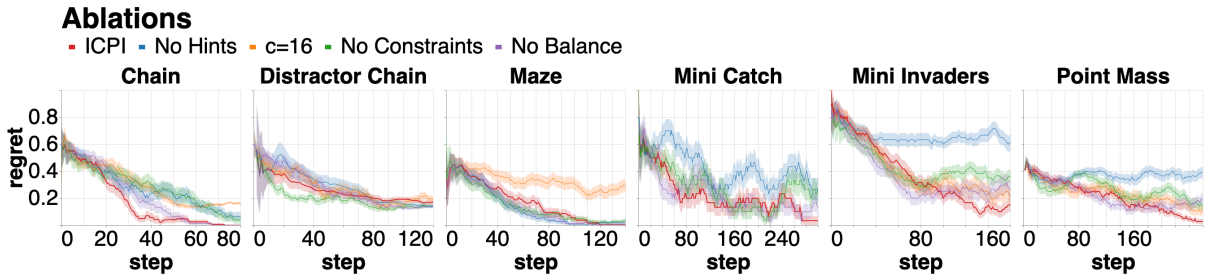


Figure 17: Comparison of ICPI with ablations.

2.4.4 Ablation of ICPI components

With these experiments, we ablate those components of the algorithm which are not, in principle, essential to learning (Figure 17). "No Hints" ablates the hints described in the paragraph. "No Balance" removes the balancing of different kinds of time-steps described in the paragraph (for example, $\mathcal{D}_b$ is allowed to contain an unequal number of terminal and non-terminal time-steps). The "No Constraints" baseline removes the constraints on these time-steps described in the same paragraph. For example, $\mathcal{D}_r$ is allowed to contain a mixture of terminal and non-terminal time-steps (regardless of the model's termination prediction). Finally, "$c = 16$" baseline prompts the rollout policy with the last 16 trajectories (instead of the last 8, as in ICPI).

In general, the ablations to ICPI exhibit a moderate drop in performance. However the two most significant are "No Hints" and "c=16." The poor performance of the "No Hints" model suggests that in several settings, Codex is unable to infer the salient parts of the observations without domain-specific hints. Our hope is that the necessity for hints of this kind will diminish as language models mature. On the other hand, the poor performance of the "c=16" model is reassuring, as it demonstrates that the recency mechanism works as intended: excluding older trajectories from the prompt helps

the action model approximate the *current* policy, as opposed to older policies. This mechanism is necessary for the policy improvement properties described in to hold.

2.4.5 Comparison of Different Language Models

While our lab lacks the resources to do a full study of scaling properties, we did compare several language models of varying size (see Figure 18). See Table 8 for details about these models. Both `code-davinci-002` and `code-cushman-001` are variations of the Codex language model. The exact number of parameters in these models is proprietary according to OpenAI, but (Chen et al. 2021c) describes Codex as fine-tuned from GPT-3 (Brown et al. 2020b), which contains 185 billion parameters. As for the distinction between the variations, the OpenAI website describes `code-cushman-001` as "almost as capable as Davinci Codex, but slightly faster."

We found that the rate of learning and final performance of the smaller models fell significantly short of Codex on all but the simplest domain, *chain*. Examining the trajectories generated by agents trained using these models, we noted that in several cases, they seemed to struggle to apprehend the underlying "logic" of successful trajectories, which hampered the ability of the rollout policy to produce good actions. Since these smaller models were not trained on identical data, we are unable to isolate the role of size in these results. However, the failure of all of these smaller models to learn suggests that size has some role to play in performance. We conjecture that larger models developed in the future may demonstrate comparable improvements in performance over our Codex model.

2.4.6 Limitations

ICPI can theoretically work on any control task with discrete actions, due to the guarantees associated with policy iteration. However, there are two limitations worth noting. First ICPI requires observations to be encoded as text. While any observation can in principle be encoded numerically and therefore in text, some might be very difficult for a language model to interpret. For example, a language model would be likely fail to extract meaning from an array of RGB pixels. Such an array would also have quite a verbose representation in text, preventing us from fitting the large number of transitions into the context window, necessary for ICPI to infer the dynamics of the environment. A second limitation is the requirement for all information about the environment to be conveyed through the context. For some environments with complex dynamics, it might be difficult for a model to infer them accurately based only on example transitions. Also, the stochastic nature of the prompt can produce unstable behavior. So in summary, we want to use a model that is somewhat specialized to its domain in order to encode some useful priors in the weights and diminish the reliance on context.

2.4.7 Societal Impacts

An extensive literature (Abid et al. 2021; Liang et al. 2021; Pan et al. 2023; Tamkin et al. 2021) has explored the possible positive and negative impacts of LLMs. Some of this work has explored mitigation strategies. In extending LLMs to RL, our work inherits

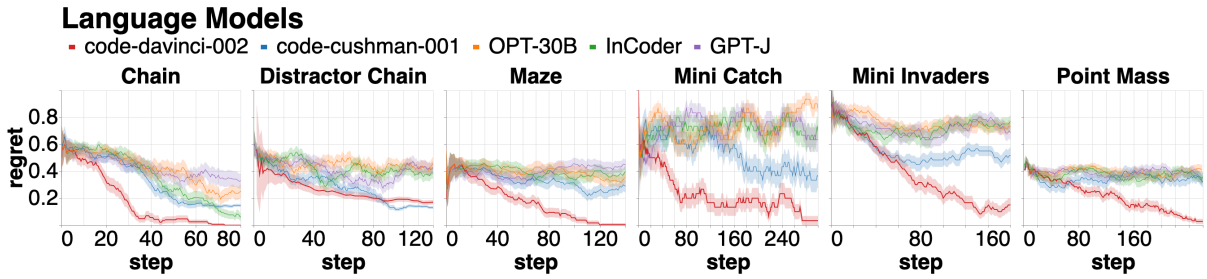


Figure 18: Comparison of different language models used to implement ICPI. Note that all other figures in this chapter use `code-davinci-002` for ICPI.

these benefits and challenges. We highlight two concerns: the use of LLMs to spread misinformation and the detrimental carbon cost of training and using these models.

2.5 Conclusion

Our main contribution in this chapter is a method for implementing the policy iteration algorithm using Large Language Models and the mechanism of in-context learning. The algorithm uses a foundation model as both a world model and policy to compute Q-values via rollouts. Although we presented the method here as text-based, it is general enough to be applied to any foundation model that works through prompting, including multi-modal models like (Reed et al. 2022) and (Seo et al. 2022). In experiments we showed that the algorithm works in six illustrative domains imposing different challenges for ICPI, confirming the benefit of the LLM-rollout-based policy improvement. While the empirical results are preliminary, we believe the approach provides an important new way to use LLMs that will increase in effectiveness as the models themselves become more powerful.

3 Algorithm Distillation + Model-Based Planning

3.1 Introduction

Generalization to novel tasks is an important challenge in multitask reinforcement learning (RL). This setting entails a training regime that includes a diversity of dynamics or reward functions, and a test regime that evaluates the agent on unseen dynamics or reward functions. A typical approach to this challenge is as follows: train a policy on the training tasks and use various mechanisms to adapt the policy to the novel setting. In this work, we explore the benefits of focusing instead on adapting a *model* as a means to adapt a policy, as opposed to adapting the policy directly. If the model successfully adapts to the evaluation setting, we may use a variety of planning methods to recover a good policy.

Our approach builds on the previous chapter, using a similar methodology to choose actions on the downstream evaluation task. Concretely, on each timestep, we use our learned model to perform multiple rollouts, each starting from a different action in the action space. We use these rollouts to estimate state-action values. Our behavior policy (as opposed to the policy used in the rollouts) chooses the action corresponding

to the highest estimate. As we demonstrated in that work, this approach implements a form of policy iteration, an algorithm proven to eventually converge to the optimal policy.

Our previous work used generic foundation models, namely Codex (Chen et al. 2021a), to implement both a world model and a policy during the aforementioned rollouts. In contrast, our present work assumes access to a dataset of RL trajectories and uses this data to train a causal transformer (Vaswani et al. 2017a) as a world model. Because the transformer is trained on RL data and the evaluation task is similarly distributed to the training tasks, the transformer is capable of modeling more complex domains than the general-purpose foundation models from which we elicited world-model predictions using a variety of prompting techniques. Another advantage of training the model from scratch is that we can more easily assess generalization by comparing training tasks with evaluation tasks. Such comparisons are not possible when using a foundation model trained on a large, opaque dataset.

One possible rationale for our hypothesis, that planning with an adapted model will generalize better than direct policy adaptation, is that a model relies only on local information whereas a policy relies on non-local information. It is worth noting that model-based planning gains this advantage at the cost of compound error — the tendency for any approximate model to accumulate error in an auto-regressive chain, compounding the errors of new predictions with the errors of prior predictions on which they are conditioned. Why should we expect the benefit to outweigh the cost?

We argue that a function approximator can only learn a policy one of two ways: either it learns the underlying logic of the policy, or it memorizes the policy without distilling this logic — a strategy which does not generalize. However, this underlying logic entails some kind of implicit value estimation, which entails implicit modeling of the environment. In principle, these implicit operations are susceptible to compound error accumulation in much the same way as explicit model-based planning methods. Therefore the switch to model-based planning does not actually introduce compound error as a new cost. Meanwhile, model-based planning does eliminate the possibility of policy memorization, since actions and values are not inferred directly but computed. Our conclusion is that the net benefit of model-based planning for generalization is positive.

We tested this hypothesis in two sets of domains, a gridworld with randomly spawned walls, and a simulated robotics domain implemented in Mujoco (Todorov et al. 2012). In the gridworld setting, we found that the model-based approach consistently outperformed a direct policy adaptation approach, even as we varied the difficulty of generalization and the quantity of training data. We reach a similar conclusion in the robotics domain, though the results are less extensive.

3.2 Background

For a review of Markov Decision Processes (MDPs) and model-based-planning, see Section 2.2.1.1 and Section 2.2.1.2, respectively.

3.2.1.1 Transformers

Transformers (Vaswani et al. 2017a) are a class of neural networks which map an input sequence to an output sequence of the same length and utilize a mechanism known as "self-attention." This mechanism maps the i^{th} element of the input sequence to a key vector k_i, a value vector v_i, and query vector q_i. The output, per index, of self-attention is a weighted sum of the value vectors:

$$\text{Attention } (q_i, k, v) = \sum_j \text{softmax } \left(q_i k_j / \sqrt{D}\right) v_j \tag{1}$$

where D is the dimensionality of the vectors. The softmax is applied across all inputs so that $\sum_j \text{softmax } \left(q_i k_j / \sqrt{D}\right) = 1$. A typical transformer applies this self-attention operation multiple times, interspersing linear projections and layer-norm operations between each application. A *causal* transformer applies a mask to the attention operation which prevents the model from attending from the i^{th} element to the j^{th} element if $i < j$, that is, if the j^{th} element appears later in the sequence. Intuitively, this prevents the model from conditioning inferences on future information.

3.2.1.2 Algorithm Distillation

Algorithm Distillation (Laskin et al. 2022) is a method for distilling the logic of a source RL algorithm into a causal transformer. The method assumes access to a dataset of "learning histories" generated by the source algorithm, which comprise observation, action, reward sequences spanning the entire course of learning, starting with initial random behavior and ending with fully optimized behavior. The transformer is trained to predict actions given the entire history of behavior or some large subset of it. Op-

timal prediction of these actions requires the model to capture not only the current policy but also the improvements to that policy applied by the source algorithm. Autoregressive rollouts of the distilled model, in which actions predicted by the model are actually used to interact with the environment and then cycled back into the model input, demonstrate improvement of the policy, often to the point of near optimality, without any adjustment to the model's parameters. This indicates that the model does, in fact, capture some policy improvement logic from the source algorithm.

3.3 Method

In this section, we describe the details of our proposed algorithm, which we call Algorithm Distillation ++ (AD++). We assume a dataset of N trajectories generated by interactions with an environment:

$$\mathcal{D} \coloneqq ((x_0^n, a_0^n, r_0^n, b_0^n, ..., x_T^n, a_T^n, r_T^n, b_T^n) \sim \pi_n)_{n=1}^N \tag{2}$$

with x_t^n referring to the t^{th} observation in the n^{th} trajectory, a_t^n to the t^{th} action, r_t^n to the t^{th} reward, and b_t^n to the t^{th} termination boolean. Each trajectory corresponds with a single task $\mathcal{T}$ and a single policy π, but the dataset contains as many as N tasks and policies. We also introduce the following nomenclature for trajectory histories:

$$h_t^n \coloneqq (a_{t-c}^n, r_{t-c}^n, b_{t-c}^n, x_{t-c+1}^n, ..., a_{t-1}^n, r_{t-1}^n, b_{t-1}^n, x_t^n) \tag{3}$$

where c is a fixed hyperparameter.

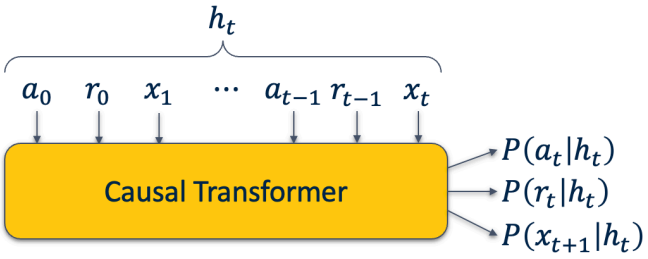


Figure 19: Diagram of inputs and outputs for the ADPI architecture. For succinctness, we omit the superscripts for the inputs. However, note that each of these transitions is sampled from the same task and learning history.

3.4 Model Training

Given such a dataset, we may train a world-model with a negative log likelihood (NLL) loss:

$$\mathcal{L}_\theta := -\sum_{n=1}^{N}\sum_{t=1}^{T-1} \log P_\theta(a_t^n \mid h_t^n) + \log P_\theta(r_t^n, b_t^n, x_{t+1}^n \mid h_t^n, a_t^n) \tag{4}$$

In this work we implement P_θ as a causal transformer. For action-prediction $P_\theta(a_t^n \mid h_t^n)$, the inputs comprise chronological $(a_{i-1}^n r_{i-1}^n b_{i-1}^n x_i^n)_{i=t-c}^{t}$ tuples. Each index of the transformer comprises one such tuple, with each component of the tuple embedded and the embeddings concatenated. For the other predictions $P_\theta(r_t^n, b_t^n, x_{t+1}^n \mid h_t^n, a_t^n)$, we use the same procedure but rotate each tuple such that index i corresponds to $r_{i-1}^n b_{i-1}^n x_{i-1}^n a_i^n$. See Figure 19 for a schematic illustrating the inputs and outputs of the architecture.

3.5 Downstream Evaluation

Our approach to choosing actions during the downstream evaluation is as follows. For each action a in a set of candidate actions (either our complete action space or some subset thereof), we compute a state-action value estimate $Q(h_t, a)$, where h_t is the his-

Algorithm 3: Estimating Q-values with monte-carlo rollouts.

1: **function** $Q(h_t, a)$
2: $u \leftarrow 1$ ▷ time step for rollout
3: $a^u \leftarrow a$
4: **while** termination condition not met **do**
5: $r^u, b^u, x^{u+1} \sim P_\theta(\cdot\,|h_t, a^u)$ ▷ Model reward, termination, next observation
6: $a^{u+1} \sim P_\theta(\cdot\,|h_t, r_t, b_t, x_{t+1})$ ▷ Model policy
7: $u \leftarrow u + 1$ ▷ Append predictions to history.
8: **end while**
9: **return** $\sum_{u=0}^{t} \gamma^{u-1} r_u$
10: **end function**

tory defined in Equation 3. We do this by modeling a rollout conditioned on the history h_t, and from a. Modelling the rollout is a cycle of sampling values from the model and feeding them back into the model auto-regressively. First we sample (r^u, b^u, x^{u+1}) (line 5 in Algorithm 3). Based on this prediction, we sample a^{u+1} (line 6). Adding this to the input allows us to sample $(r^{u+1}, b^{u+1}, x^{u+2})$, repeating this cycle until some termination condition is reached. For example, we might terminate the process once b^u is true, or once the rollout reaches some maximum length.

The final step is to choose an action. Having modelled the rollout, we compute the value estimate as the discounted sum of rewards in the rollout (line 9). Repeating this process of modelling rollouts and computing value estimates for each action in a set of candidate actions, we simply choose the action with the highest value estimate: $\arg\max_{a \in \mathcal{A}} Q(h_t, a)$.

3.6 Policy improvement

If the downstream task is sufficiently dissimilar to the tasks in the training dataset, or if the training dataset does not contain optimal policies, then it is unlikely that

the procedure described above will initially yield an optimal policy. Some method for policy improvement will be necessary.

3.6.1.1 Policy Iteration

Our method satisfies this requirement by implementing a form of policy iteration. To see this, first observe that our model is always trained to map a behavior history drawn from a single policy to actions drawn from the same policy. A fully trained model will therefore learn to match the distribution of actions in its output to the distribution of actions in its input. Since our rollouts are conditioned on histories drawn from our behavior policy, the rollout policy will approximately match this policy. Our value estimates will therefore be conditioned on the current behavior policy. However, by choosing the action corresponding to $\arg\max_{a \in \mathcal{A}} Q^{\pi}(h_t, a)$, our behavior policy always improves on π, the policy on which $Q^{\pi}(h_t, a)$ is conditioned, a consequence of the policy improvement theorem. Thus, each time an action is chosen using this $\arg\max$ method, our behavior policy improves on itself.

Walking through this process step by step, suppose π^n is some policy that we use to behave. We collect a trajectory containing actions drawn from this policy. When we perform rollouts, we condition on this trajectory and the rollout policy simulates π^n. Assuming that this simulation is accurate as well as the world model, our value estimate will be an unbiased monte carlo estimate of $Q^{\pi^n}(h_t, a)$ for any action a. Then we act with policy $\pi^{n+1} := \arg\max_{a \in \mathcal{A}} Q^{\pi_n}(h_t, a)$. But π^{n+1} is at least as good as π^n. Using the same reasoning, π^{n+2} will be at least as good as π^{n+1}, and so on. Note that

Figure 20: Diagram of the policy improvement cycle. The behavior policy π generates new actions and the environment generates new transitions which are appended to the front of the context window. By retaining only the most recent transitions in the context window, we ensure that the policy represented in the context approximates the behavior policy. Next, we rely on the policy improvement operator learned by the transformer to infer the distribution of actions in the context and yield an *improved* action for our rollout policy. Therefore the rollout policy approximates the prompt policy. Finally, by choosing actions greedily with respect to our value estimates, we implement a new behavior policy π' that improves the rollout policy.

in our implementation, we perform the arg max at each step, rather than first collecting a full trajectory with a single policy. See

3.6.1.2 Algorithm Distillation

Our setting is almost identical to Algorithm Distillation (AD) (Laskin et al. 2022), if we include the assumption that trajectories include a full learning history. Rather than competing with AD, our method is actually complementary to it. If the input to our transformer is a sufficiently long history of behavior, then the rollout policy will not only match the input policy but actually improve upon it, as demonstrated in that paper. Then the procedure described in Algorithm 3 for estimating Q-values will actually condition these estimates on a policy π'_n that is at least as good as the input policy π_n. Then $V^{\pi_n} \leq V^{\pi'_n} \leq V^{\pi_{n+1}}$. Therefore each step of improvement actually superimposes the two improvement operators, one from the arg max operator, the other from AD.

3.6.2 Extension to Continuous Actions

When evaluating AD++ on continuous action domains, the formulation we have described must be modified, since it is not possible to iterate over the action space. Instead, we sample a fixed number of actions from $P_\theta(\cdot\,|h_t, r_t, b_t, x_{t+1})$ and for each sampled action, perform a rollout per Algorithm 3. This step does not preserve the policy improvement guarantees (Section 3.6), but works well enough in practice.

3.6.3 Beam search

The algorithm that we have described can be further augmented with beam Search, similar to Janner et al. (2021b). While this proved useful in only one of the domains that we evaluated, we describe it here for the sake of completeness. Using beam search requires learning a value function which is used for pruning the tree. We augment Equation 4 with the following term:

$$\mathcal{L}'_\theta \coloneqq \mathcal{L}_\theta + \sum_{n=1}^{N}\sum_{t=1}^{T-1} \log P_\theta\left(\sum_{u=t}^{T} \gamma^{T-u} r_u \mid h_t^n\right) \tag{5}$$

When conditioning h_t^n, the model P_θ outputs two predictions, one corresponding to the next action, the other corresponding to the value of the current state. For the latter, we use the discounted cumulative sum of empirical reward as a target.

The procedure for estimating value with beam search is as follows. On each step of the rollout procedure described in Algorithm 3, we sample N actions (instead of just one per rollout). Instead of a series of parallel chains, as in the original algorithm, this procedure would result in an expanding tree with arity N. For computational

tractability, we therefore rank the paths by value (as estimated by P_θ) and discard the bottom $\frac{N-1}{N}$ so that the number of active pathways per rollout step does not increase. Ranking is performed across all rollouts, so all the descendants of a given node may eventually be pruned.

3.7 Related Work

A paper that strongly influenced this work is Trajectory Transformer (Janner et al. 2021b), from which we borrow much of our methodology. We distinguish ourselves from that work by focusing on in-context learning in partially observed multi-task domains, and through the incorporation of Algorithm Distillation (Laskin et al. 2022).

Several recent works have explored the use of in-context learning to adapt transformer-based agents to new tasks. (Raparthy et al. 2023) study the properties that are conducive to generalization in these kinds of agents, especially highlighting "burstiness" (Chan et al. 2022b) and "multi-trajectory" inputs – inputs containing multiple episodes from the same task, as used in Laskin et al. (2022) and in this work. Lee et al. (2023) propose an approach similar to AD, but instead of predicting the *next* action, they directly predict the *optimal* action. They demonstrate that this gives rise to similar forms of in-context learning and outperforms AD on several tasks. Pinon et al. (2022) train a dynamics model, similar to this work, and execute tree search, though unlike this work, they use a fixed policy.

Transformers have also been studied extensively in the capacity of world models. Micheli et al. (2022) train a transformer world-model using methods similar to our own

and demonstrate that training an agent to optimize return within this model is capable of significantly improving sample-complexity on the Atari 100k benchmark. Robine et al. (2023) augment this architecture with a variational autoencoder for generating compact representations of the observations. "TransDreamer" (Chen et al. 2022) directly emulates Dreamer V2 (Hafner et al. 2020) adjusting that algorithm to use transformers in place of GRUs to capture recurrence.

3.8 Experiments

3.8.1.1 Domains

In this work, we chose to focus on partially observable domains. This ensures that both the initial policy and the initial model in our downstream domain will be suboptimal, since the true dynamics or reward function cannot be inferred until the agent has gathered experience. Recovering the optimal policy will coincide with model improvement. Model improvement will occur as the agent collects experience and the transformer context is populated with transitions drawn from the current dynamics and reward functions. Policy improvement relies on the mechanisms detailed in the previous sections. One hypothesis that our experiments test is whether these learning processes can successfully happen concurrently.

Our first set of experiments occur within a discrete, partially observable, 5×5 grid world. The agent has four actions, up, left, down, and right. For each task, we spawn a "key" and a "door" in a random location. The agent receives a reward of 1

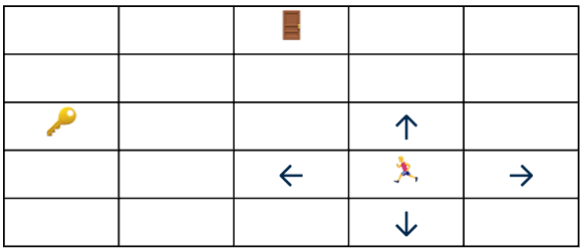


Figure 21: Top-down view of the grid-world environment. The agent must first visit the key and then the door. Note that in many of our experiments, unseen walls are added to the interstices between grid cells.

for visiting the key location and then a reward of 1 for visiting the door. The agent observes only its own position and must infer the positions of the key and the door from the history of interactions which the transformer prompt contains. The episode terminates when the agent visits the door, or after 25 time-steps. See Figure 21 for a visualization of the grid-world environment.

3.8.1.2 Baselines

We compare our method with two baselines: vanilla AD and "Ground-Truth" AD++. The latter is identical to our method, but uses a ground-truth model of the environment, which is free of error. The comparison with AD highlights the contribution of the model-based planning component of our method. The comparison with Ground-Truth AD++ establishes an approximate upper bound for our method and distinguishes the contribution of model error to the RL metrics that we record.

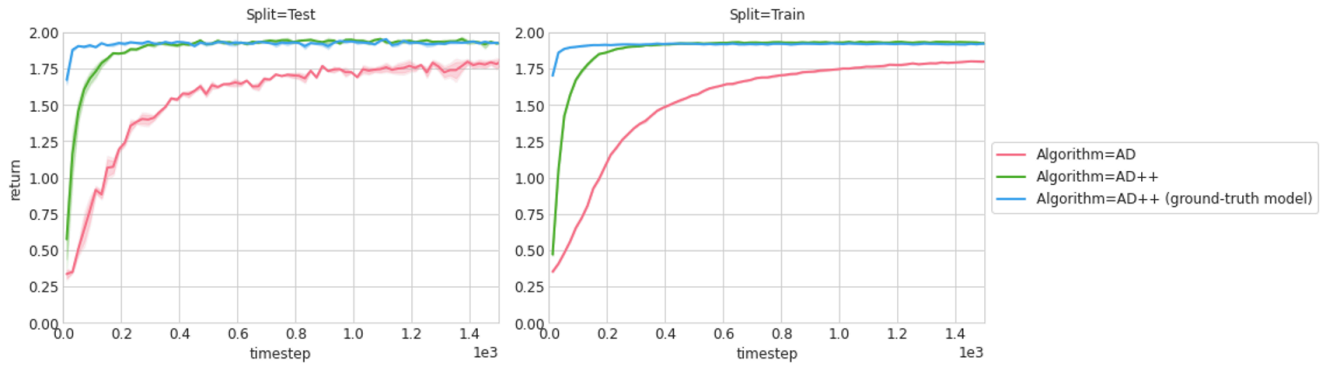


Figure 22: Evaluation on withheld location pairs.

3.8.2 Results

3.8.2.1 Evaluation on Withheld Goals

In our first experiment, we evaluate the agent on a set of withheld key-door pairs, which we sample uniformly at random (10% of all possible pairs) and remove from the training set. As Figure 22 indicates, our algorithm outperforms the AD baseline both in time to converge and final performance. We attribute this to the fact that our method's downstream policy directly optimizes expected return, choosing actions that correspond to the highest value estimate. In contrast, AD's policy only maximizes return by proxy — maximizing the probability of the actions of a source algorithm which in turn maximizes expected return. This indirection contributes noise to the downstream policy through modeling error. Moreover, we note that our method completely recovers the performance of the ground-truth baseline, though its speed of convergence lags slightly, due to the initial exploration phase in which the model learns the reward function through trial and error.

Next, we increase the generalization challenge by holding out key and door locations entirely. During training of the source algorithm, we never place keys or doors

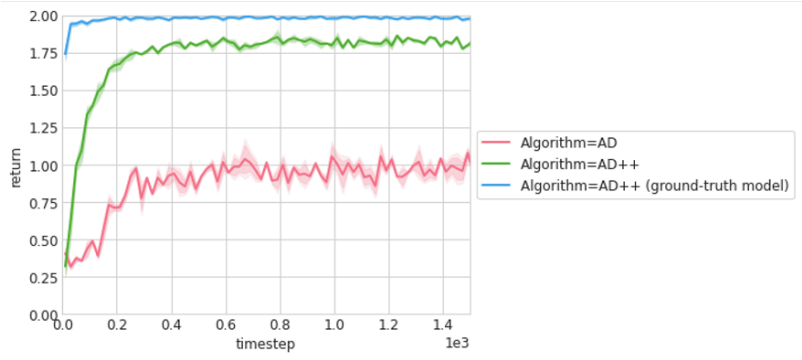


Figure 23: Evaluation on fully withheld locations.

in the upper-left four cells of the grid. During evaluation, we place both keys and doors exclusively within this region. As Figure 23 demonstrates, AD generalizes poorly in this setting, on average discovering only one of the two goals. In contrast, our method maintains relatively high performance. We attribute this to the fact that our method learns low-level planning primitives (the reward function), which generalize better than high-level abstractions like a policy. As we argued in Section 3.1, higher-level abstractions are prone to memorization since they do not perfectly distill the logic which produced them.

3.8.2.2 Evaluation on Withheld Wall Configurations

In addition to evaluating generalization to novel reward functions, we also evaluated our method's ability to generalize to novel dynamics. We did this by adding walls to the grid world, which obstruct the agent's movement. During training we placed the walls at all possible locations, sampled IID, with 10% probability. During evaluation, we tested the agent on equal or higher percentages of wall placement. As indicated by Figure 24, our method maintains performance and nearly matches the ground-truth

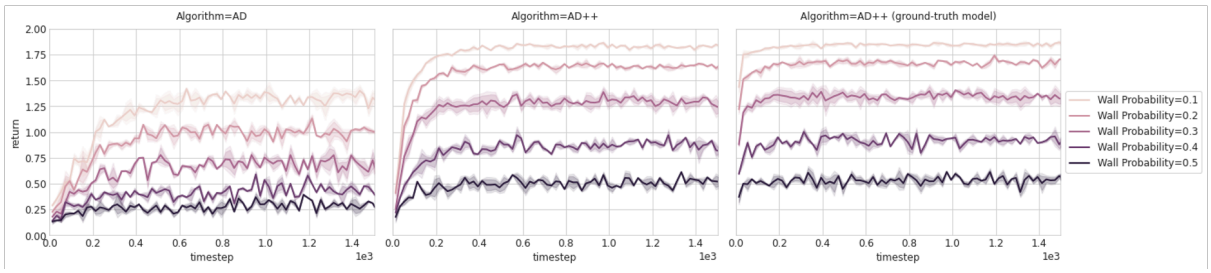


Figure 24: Generalization to higher percentages of walls.

version, while AD's performance rapidly degrades. Again we attribute this to the tendency of lower-level primitives to generalize better than higher-level abstractions.

Because walls are chosen from all possible positions IID, some configurations may wall off either the key or the door. In order to remove this confounder, we ran a set of experiments in which we train the agent in the same 10% wall setting as before, but evaluate it on a set of configurations that guarantee the reachability of both goals. Specifically, we generate procedural mazes in which all cells of the grid are reachable and sample some percentage of the walls in the maze. As Figure 25 demonstrates, this widens the performance gap between our method and the AD baseline.

3.8.2.3 Model Accuracy

In order to acquire a better understanding of the model's ability to in-context learn, we plotted model accuracy in the generalization to 10% walls setting. Note that while the percentages of walls in the training and evaluation setting are the same, the exact wall

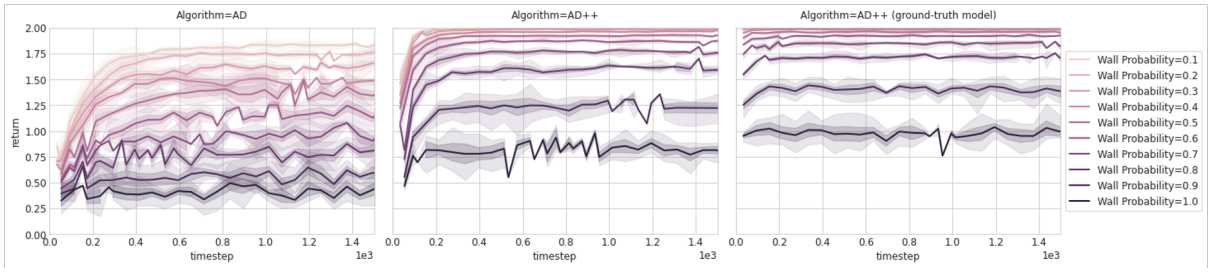


Figure 25: Generalization to higher percentages of walls with guaranteed achievability.

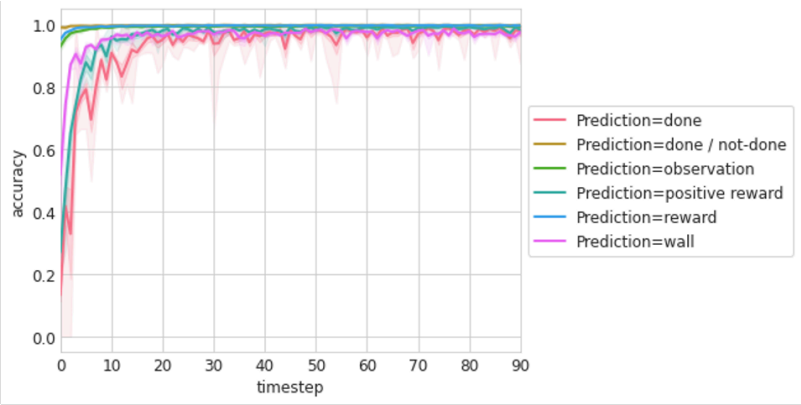


Figure 26: Accuracy of model predictions over the course of an evaluation rollout.

placements are varied during training and the evaluation wall placements are withheld, so that the model must infer them from context. In Figure 26, we measure the accuracy of the model's prediction of termination signals (labeled "done / not done"), of next observations (labeled "observation"), and of rewards (labeled "reward"). These predictions start near optimal, since the agent can rely on priors: that most timesteps do not terminate, that most transitions result in successful movement (no wall), and that the reward is 0. However, we also measure prediction accuracy for these rare events: the line labeled "done" measures termination-prediction accuracy for terminal timesteps only; the "positive reward" line measures reward-prediction accuracy on timesteps with positive reward; and the "wall" line measures accuracy on timesteps when the agent's movement is obstructed by a random wall. As Figure 26 demonstrates, even for these rare events, the model rapidly recovers accuracy near 100%.

3.8.2.4 Contribution of Model Error to Performance

While Figure 27 indicates that our model generally achieves high accuracy in these simple domains, we nevertheless wish to understand the impact of a suboptimal model

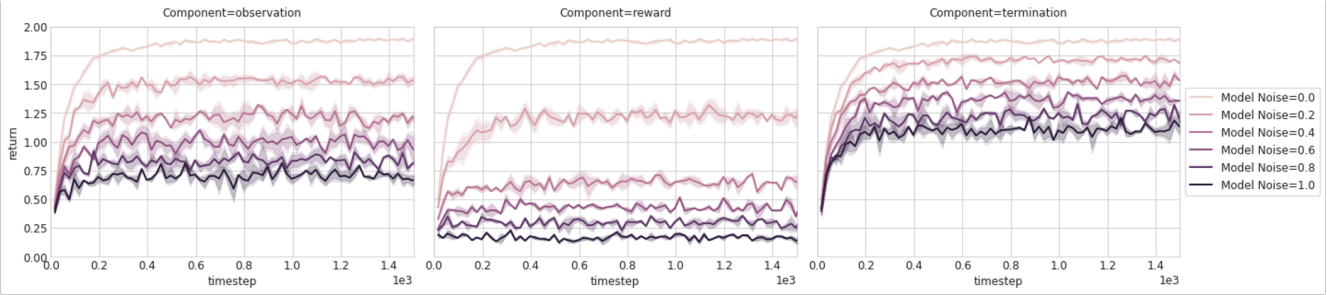


Figure 27: Impact of model error on performance, measured by introducing noise into each component of the model's predictions.

on RL performance. To test this, we introduced noise into different components of the model's predictions. In Figure 27, we note that performance is fairly robust to noise in the termination predictions, but very sensitive to noise in the reward predictions. Encouragingly, the model demonstrates reasonable performance with as much as 20% noise in the observation predictions. Also, as indicated, the method is quite robust to noise in the action model. We also note that AD's sensitivity to noise in the policy explains its lower performance in many of the settings previously discussed.

3.8.2.5 Data Scaling Properties

We also examined the impacts of scaling the quantity of data that our model was trained on. In Figure 28, we scale the quantity of the training data along the IID dimension, with the x-axis measuring the number of source algorithm histories in the training data

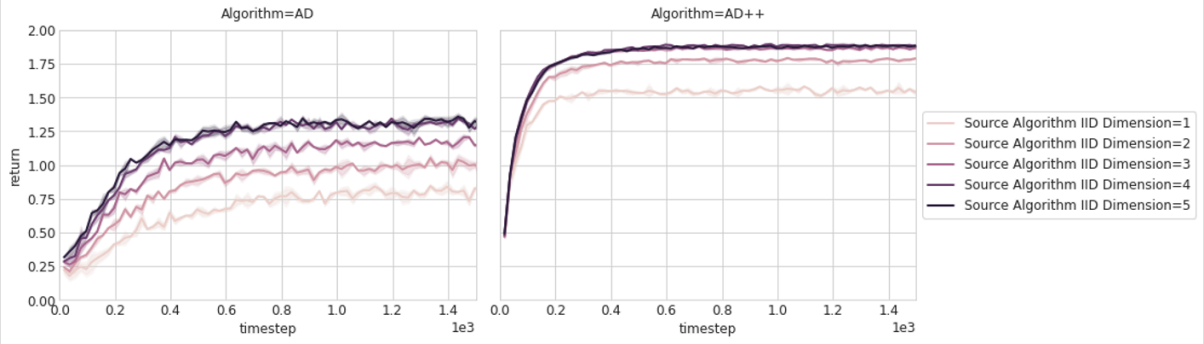


Figure 28: Impact of scaling the length of training of the source algorithm.

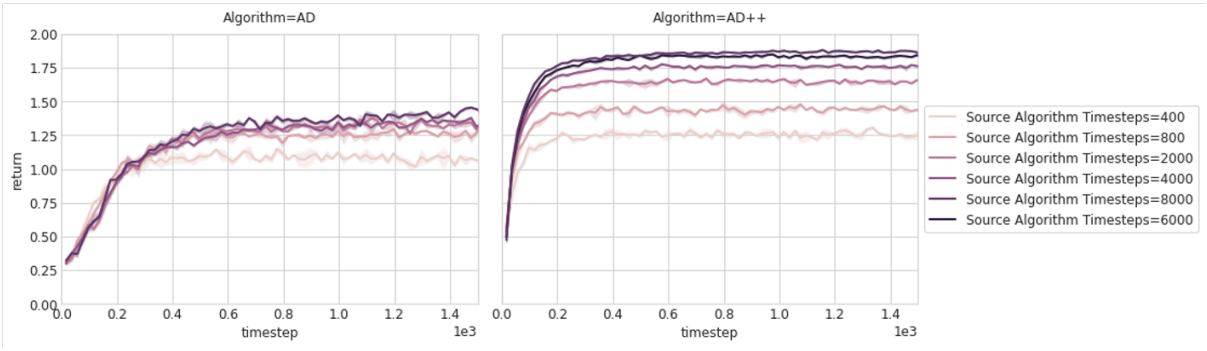


Figure 29: Impact of scaling the training data along the IID dimension.

scaled according to the equation 256×2^x. In Figure 29, we scale the length for which each source algorithm is trained, with the x-axis measuring the number of timesteps of training scaled according to the same equation. This result was surprising, as we expected AD to be *more* sensitive to reduced training time, since that algorithm is more dependent on demonstration of the optimal policy. Nevertheless, we note that our method outperforms AD in all data regimes.

3.8.3 Continuous-State and Continuous-Action Domains

Finally, we evaluate the ability of AD++ to learn in domains with continuous states and actions. In order to adapt AD++ to infinitely large action spaces, we use the sampling technique described in Section 3.6.2. In the experiments that follow, the number of actions sampled at the beginning of the rollouts is 128.

3.8.3.1 Sparse Point Environment

In the "Sparse Point" environment, a point spawns randomly on a half circle. The agent does not observe the point and has to discover it through exploration. The agent receives reward for occupying coordinates within a fixed radius of the goal. The agent's

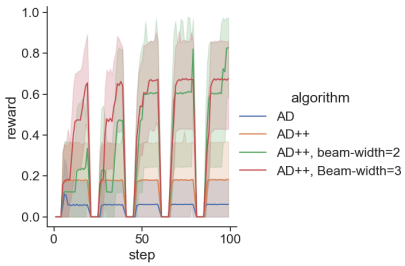


Figure 30: Evaluation on the "Sparse-Point" environment.

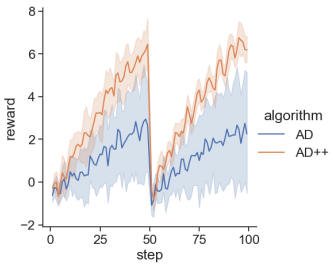


Figure 31: Evaluation on the "Half-Cheetah Direction" domain.

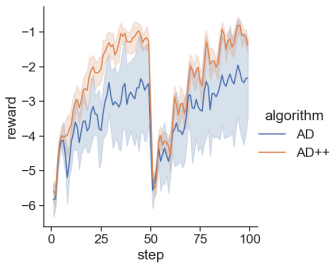


Figure 32: Evaluation on the "Half-Cheetah Velocity" domain.

observation space consists of 2d position coordinates and its action space consists of 2d position deltas. The Sparse Point environment tests the ability of the agent to explore efficiently, since an agent that concentrates its exploration on the half-circle will significantly outperform one that explores all positions with equal probability.

As Figure 30 demonstrates, vanilla AD fares quite poorly in this setting. Of the 20 seeds in the diagram, only two discover the goal and only one returns to it consistently. The AD agent either explores randomly in the vicinity of the origin — emulating policies observed early in the source data — or commits arbitrarily to a point on the arc and remains in its vicinity — emulating later policies, but ignoring the lack of experienced reward. The Sparse Point environment highlights a weakness in vanilla AD. During training, the source algorithm — which uses one agent per task — can memorize the location of the goal. It therefore never exhibits Bayes-optimal exploration patterns for AD to imitate.

Why should we expect AD++ to perform better? For the same reasons that cause vanilla AD to fail, we should not expect the simulated rollouts of AD++ to perform Bayes-optimal exploration. However, before the model experiences reward, its reward predictions will reflect the prior, which has support on the semi-circle and

nowhere else. Therefore any rollouts that do not lead to the semi-circle should result in a simulation of zero cumulative reward.

In order for AD++ to recover Bayes optimal exploration in the Sparse Point environment, two random events must co-occur: some of the rollouts must lead to the semi-circle, and the model must anticipate reward there, without having necessarily experienced it. In practice, this does not happen consistently. The rollout policy often degenerates into random dithering and the reward model often predicts no reward at all. We found that the key to eliciting consistent performance from AD++ was to perform beam-search as described in Section 3.6.3. This effectively increases the opportunities for rollouts to lead to the arc and for the reward model to anticipate reward there. For example, paths that do not lead to the arc can be pruned entirely, while those that do can benefit from node-expansion. As Figure 30 demonstrates, both beam-search methods significantly outperform vanilla AD and AD++.

3.8.3.2 Half-Cheetah Environments

In our final set of environments, we explore two variants on the well-known Mujoco "Half-Cheetah" environment. This environment uses a 2D two-legged embodiment. In order to instantiate a multi-task problem, we vary the reward function: for the "Half-Cheetah Direction" environment, we instantiate two tasks, one which rewards the agent for forward movement of the Cheetah and one that rewards it for backward movement. For the "Half-Cheetah Velocity" environment, we choose a target velocity per task. The agent receives a reward penalty for the difference between its current velocity and

the target. Per the original Half-Cheetah environment, the agent also receives a control reward that penalizes large actions. As Figure 31 and Figure 32 demonstrate, AD++ outperforms vanilla AD on both domains.

3.9 Conclusion

This chapter presents an approach for combining ICPI with AD. The resulting method scales to more complex settings than those explored in the previous chapter. Moreover, the method significantly outperforms vanilla AD in a variety of settings.

4 Bellman Update Networks

In the previous two chapters, we demonstrated the capacity of a sequence model to implement policy iteration in-context, enabling fast adaptation to novel RL settings without recourse to gradients. This approach relies on some mechanism for estimating state-action values, with respect to which the policy can choose actions greedily.

Previously, we satisfied this requirement by using a model to perform simulated rollouts and using these rollouts to make monte-carlo estimates. This approach has two drawbacks. First, monte-carlo estimates are generally susceptible to high variance. Second and perhaps more fundamentally, our previous methods focused on training an accurate model when value accuracy — not model accuracy — was our ultimate concern. As a result, our model learned to focus equally on all parts of the observation rather than skewing its resources toward those parts that contributed to the expected value (Grimm et al. 2020) and away from parts that are purely "decorative." Conversely, error in modeling any part of the observation, decorative or otherwise, could throw off the value estimate: a model trained on inputs containing only ground-truth observations might fail to generalize to observations corrupted by modeling error.

In this work, we attempt to address these issues by proposing a method for estimating value directly instead of using model-based rollouts. In the next section we will introduce the motivation, concept, and implementation for Bellman Update Networks.

In the subsequent section, we will review some results comparing this approach with some baselines.

4.1 Preliminaries

4.1.1 Review of In-Context Model-Based Planning

In the preceding chapter, we described work in which we trained a causal model to map a temporally sequential history of observations $x_{\leq t}$, actions $a_{\leq t}$, and rewards $r_{<t}$, to predictions of the next observation x_{t+1} and next reward r_t. Our model optimized the following loss:

$$\mathcal{L}_\theta^{\text{ADPI}} \coloneqq -E_{h_t^n \sim \mathcal{D}} \left[\sum_{t=1}^{T-1} \log P_\theta(a_t^n \mid h_t^n) + \log P_\theta(r_t^n, b_t^n, x_{t+1}^n \mid h_t^n, a_t^n) \right] \tag{6}$$

Here, h_t^n is a trajectory drawn from the n^{th} task/agent pair:

$$h_t^n \coloneqq (a_{t-1-c}^n, r_{t-1-c}^n, b_{t-1-c}^n, x_{t-c}^n, ..., a_{t-1}^n, r_{t-1}^n, b_{t-1}^n, x_t^n) \tag{7}$$

Note that the dataset $\mathcal{D}$ from which the trajectory h_t^n is drawn contains multiple tasks and agents, but as the superscript indicates, each trajectory corresponds to only one. Here we use the term "agent" to denote the set of policies emitted by a single learning algorithm. Henceforth in this chapter, we omit the n superscript for the sake of brevity and assume that there is one task and agent per trajectory. In the preceding chapter, we implemented our model as a causal transformer (Vaswani et al. 2017a). Because the model was conditioned on history, it demonstrated the capability to adapt in-context to settings with novel transition or reward functions.

4.1.2 Naive Alternative Method

An almost identical technique might be used to predict value functions in place of next-observations and next-rewards. In principle, a history of observations, actions, and rewards should be sufficient to infer the dynamics and reward of the environment as well as the current policy, all that is necessary to estimate value. Our model would then minimize the following loss:

$$\mathcal{L}_\theta := -E_{h_t}\left[\sum_{t=1}^{T-1}\sum_{a\in\mathcal{A}} \log P_\theta(Q(x_t, a) \mid h_t)\right] \tag{8}$$

where x_t is the last observation in h_t and $Q(x_t, a)$ is the value of observation x_t and action a. We note that in general, ground-truth targets are not available for such a loss. However, they may be approximated using bootstrapping methods as in Mnih et al. (2015) or Kumar et al. (2020).

One question that we seek to understand is the extent to which this approach generalizes to novel tasks and policies. We observe that the mapping from a history of observations, actions, and rewards to values is non-trivial, requiring the model to infer the policy and the dynamics of the environment, and to implicitly forecast these estimates for multiple time steps. As a result, it is reasonable to anticipate some degree of memorization.

4.2 Proposed Method

In the case of context-conditional models such as transformers, one factor that has a significant impact on memorization is the extent to which the model attends to infor-

mation in its context — "in-context" learning — as opposed to the information distilled in its weights during training — "in-weights" learning. The former will be sensitive to new information introduced during evaluation while the latter will not. As Chan et al. (2022b) suggests, the relevance or predictive power of information in a model's context strongly influences the balance of in-weights vs. in-context learning.

With this in mind, we note that the ground-truth values associated with each time-step in the model's context would be highly predictive of the value of the current state. A model provided this information in its context should attend more strongly to its context and memorize less. This would entail the following redefinition of h_t, the history on which we condition the model's predictions:

$$h_t := (a_{t-1-c}, r_{t-1-c}, b_{t-1-c}, x_{t-1-c}, Q(x_{t-c+1}, \cdot)..., Q(x_t, \cdot), a_{t-1}, r_{t-1}, b_{t-1}, x_t,) \quad (9)$$

The question remains how such input values might be obtained. After all, in a setting where we intend to estimate values, we should not assume that we already have access to good value estimates! To address this chicken-and-egg dilemma, we propose an alternative approach to predicting values *directly* and instead take inspiration from the classic Policy Evaluation algorithm (Sutton and Barto 2018), which iteratively improves a value estimate using the Bellman update equation:

$$Q_k(x_t, a_t) := r(x_t, a_t) + \gamma E_{x_{t+1} a_{t+1}}[Q_{k-1}(x_{t+1}, a_{t+1})] \quad (10)$$

For any Q_0 and for sufficiently large k, this equation converges to the true value of Q. We incorporate a similar approach in our method, proposing the following loss function:

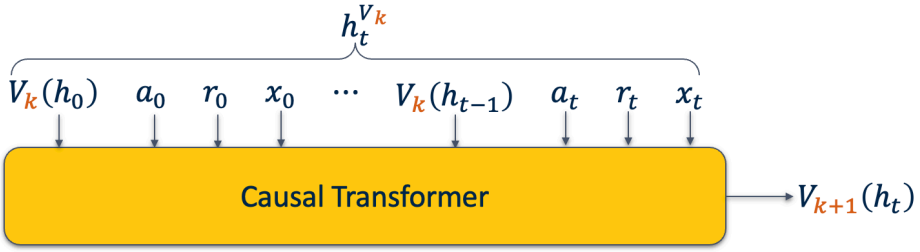


Figure 33: Sketch of inputs and outputs for the Bellman Network Architecture architecture.

$$\mathcal{L}_\theta^{\text{BUN}} := -E\left[\sum_{t=1}^{T-1}\sum_{a\in\mathcal{A}} \log P_\theta\left(Q_k(x_t,a) \mid h_t^{Q_{k-1}}\right)\right] \tag{11}$$

where $h_t^{Q_{k-1}} := (a_{t-1-c}, r_{t-1-c}, b_{t-1-c}, x_{t-c}, Q_{k-1}(x_{t-c},\cdot)..., Q_{k-1}(x_{t-1},\cdot), a_{t-1}, r_{t-1}, b_{t-1}, x_t,)$

We call a model P_θ that minimizes this loss a Bellman Update Network (BUN). Initially, we may set Q_0 to any value. By feeding this initial input into the network and then auto-regressively feeding the outputs back in, we may obtain an estimate of both the target value, $Q_k(x_t,a)$ and the context values $Q_{k-1}(x_{t-c+1},\cdot),...,Q_{k-1}(x_{t-1},\cdot)$. By conditioning predictions of Q_k on a context containing estimates of Q_{k-1}, we gain many of the benefits of the context proposed in Equation 9, in terms of promoting in-context learning over in-weights learning, without privileged access to ground-truth values. For a visualization of the inputs and outputs of the proposed architecture, see Figure 33.

This approach entails a tradeoff. By training on $k < \infty$, we significantly increase the space of inputs for which the model must learn good representations. Therefore, the method takes longer to train and demands more of the capacity of the model. What we gain are more robust representations, capable of better generalization to unseen settings.

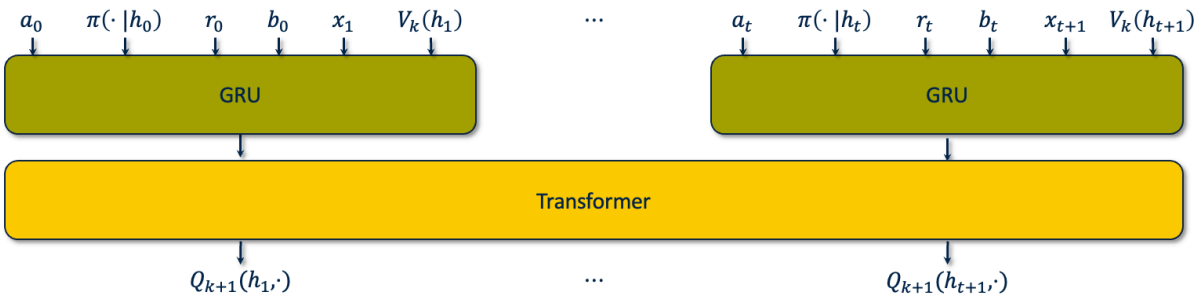


Figure 34: Diagram of the Bellman Update Network architecture.

4.2.1 Setting

We now describe the data used to train the Bellman Update Network, though we defer the details of training to a later section. We assume that we are given a dataset of trajectories containing observations x, actions a, rewards r, terminations b, and policy logits $\pi(\cdot|x)$. We assume that this dataset contains a multitude of tasks and agents, with one task/agent per trajectory. We also assume that we have estimates of value at different numbers of Bellman updates k, although we defer the explanation of how to acquire these to Section 4.2.4.

In the preceding chapter, an agent corresponded all policies generated by a learning algorithm, starting with random policies and ending with near optimal policies. In some sections of this chapter, we will assume one policy per agent, for the sake of analysis and simplicity.

4.2.2 Architecture

As Figure 34 illustrates, the inputs to the network are a sequence of transitions from the dataset. In principle, these transitions need not be chronological, except in a partially observed setting. Importantly, the observations are offset by one index from the rest of the components of the transition. This will be explained in Section 4.2.5. Each transi-

tion gets encoded and summarized into a single fixed-size vector. We compared several methods for doing this, including small positional transformers, and found that Gated Recurrent Unit (GRU) (Cho et al. 2014) demonstrated the strongest performance. Each of these transition vectors gets passed through a transformer network (one vector per transformer index) and the final outputs are projected to scalars.

Note that in Figure 34, we provide V_k and not Q_k to the model, as in Equation 11. We found that computing

$$V_k(x_t) := \sum_a \pi(a|x_t) Q_k(x_t, a) \tag{12}$$

and providing this as input to the network, rather than providing the full array of Q-values, improved the speed and stability of learning.

To regress the outputs of the model onto value targets, we use mean-square-error loss:

$$\mathcal{L}_\theta := \sum_{t=1}^{T-1} \left[Q_\theta\left(x_t, a_t \mid h_t^{V_k}\right) - \hat{Q}_{k+1}(x_t, a_t) \right]^2 \tag{13}$$

where $h_t^{V_k}$ is a sequence of transitions containing values V_k and $\hat{Q}_{k+1}$ is a target Q value computed using bootstrapping (details in Section 4.2.4). This loss is equivalent to Equation 11 when P_θ is normally distributed with fixed standard deviation.

4.2.3 Value Estimation

As we alluded in Section 4.2, in order to estimate values using the Bellman Update Network, we auto-regressively pass the outputs of the network back into it, repeating this procedure a fixed number of steps, or until the estimates converge. One detail is

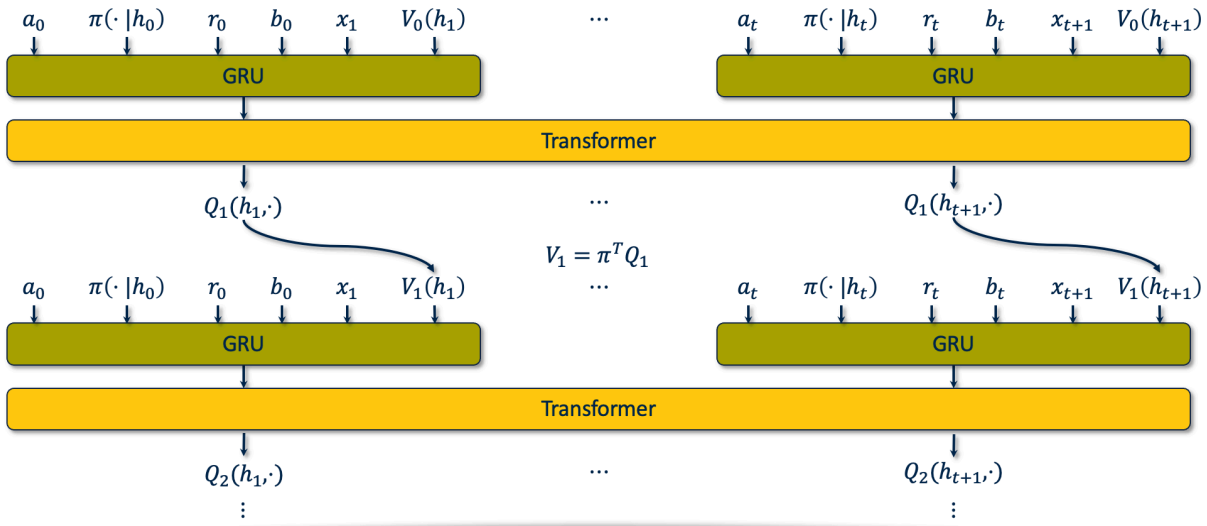


Figure 35: Diagram showing how Q-value estimates output by the Bellman Update Network are converted back into value estimates and fed back into the network.

that the network receives values as inputs and produces *Q-values* as outputs. Therefore, the output Q-values must be recombined into values before feeding them back into the network. To achieve this, we simply take the dot product of the output Q-values and the input policy logits. See Figure 35 for a visualization.

4.2.4 Training procedure

Here we describe a practical procedure for computing values to serve as inputs and targets to the network, and for training the network. For all values of k greater than 1,

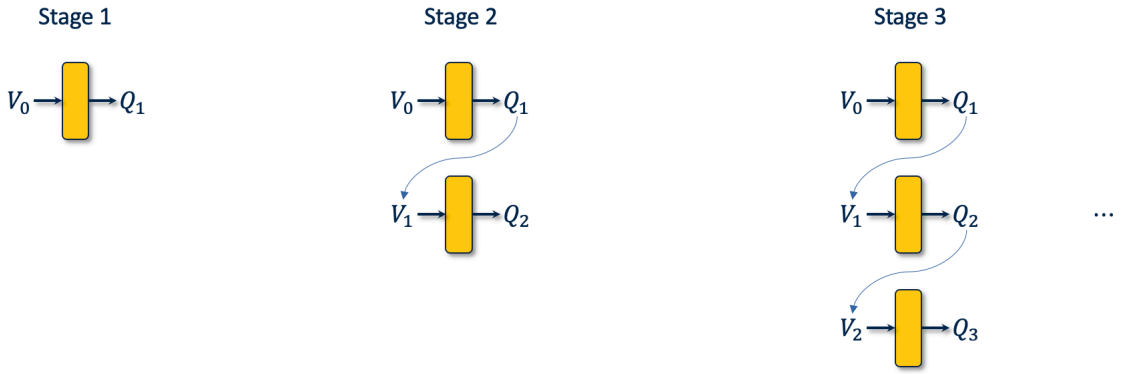


Figure 36: Visualization of the first three curriculum stages for training the Bellman Update Network.

we must choose between using inaccurate bootstrap values or adopting a curriculum. Each approach introduces a different form of non-stationarity into the training procedure. We favor the latter, since it avoids training the network on targets before they are mostly accurate.

For a visual sketch of the training procedure, see Figure 36. Our curriculum initially trains Q_1 bootstrapped from Q_0, which we set to $\mathbf{0}$. We proceed iteratively through higher order values until $Q_K \approx Q_{K-1}$. At each step in the curriculum, we continue to train Q_k for all values of $k \in 1, ..., K$ (see line 6 of Algorithm 4). This allows the network to continue improving its estimates for lower values of k even as it begins to train on higher values, thereby mitigating the effects of compound error. Another benefit of continuing to train lower values of k is that these estimates can benefit from backward transfer as the curriculum progresses. As the network produces estimates, we use them both to train the network (see line 16 in Algorithm 4) but also to produce bootstrap targets for higher values of k (see line 13).

4.2.5 Implementation Details

We implement the Bellman Update Network as a causal transformer, using the GPT2 (Radford et al. 2019) implementation from www.huggingface.co. Why is causal masking necessary, given that the target does not appear in the input to the model? To answer this question, we must draw attention to a disparity between the outputs from the model on line 15 of Algorithm 4 and the targets used to train the model on line 16. For each input observation x_t, we require the model to infer a vector of values, $\boldsymbol{Q}(\boldsymbol{x_t}, \cdot)$,

Algorithm 4: Training the Bellman Update Network.

input $c, \mathcal{D}$ ▷ Context length, RL data
$Q_0 \leftarrow \mathbf{0}$ ▷ Initialize Q-estimates to zero.
$K \leftarrow 0$
repeat
  repeat
    for $k = 0, ..., K$ **do**
      $(a_t, \pi(\cdot \mid x_t), r_t, b_t, x_{t+1})_{t=0}^{c} \sim \mathcal{D}$ ▷ sample sequence from data.
      for $t = 1, ..., 1 + c$ **do**
        $V_k(x_t) \leftarrow \sum_a \pi(a \mid x_t) Q_k(x_t, a)$ ▷ Compute values from Q-values
      end for
      $h^{V_k} \leftarrow (a_t, \pi(\cdot \mid x_t), r_t, b_t, x_{t+1}, V_k(x_{t+1}))_{t=0}^{c}$ ▷ pair transitions with values
      for $t = 1, ..., 1 + c$ **do**
        $\hat{Q}_{k+1}(x_t, a_t) \leftarrow r_t + (1 - b_t)\gamma V_k(x_{t+1})$ ▷ Bootstrap target for observed actions.
      end for
      $Q_{k+1} \leftarrow Q_\theta(h^{V_k})$ ▷ Use Bellman Update Network to estimate values
      minimize $\sum_t \left[Q_\theta\left(x_t, a_t \mid h_t^{V_k}\right) - \hat{Q}_{k+1}(x_t, a_t) \right]^2$ ▷ Optimize predictions
    end for
  until $Q_{k+1} \approx \hat{Q}_{k+1}$
  $K \leftarrow K + 1$
until $Q_{k+1} \approx Q_k$

one for each action in the action space. However, we are only able to train the model on the single action observed in the dataset for that transition. If the model is able to observe both the input observation x_t and the action a_t on which we condition the target value, the model will neglect all predictions besides $Q(x_t, a_t)$. That is, it will learn

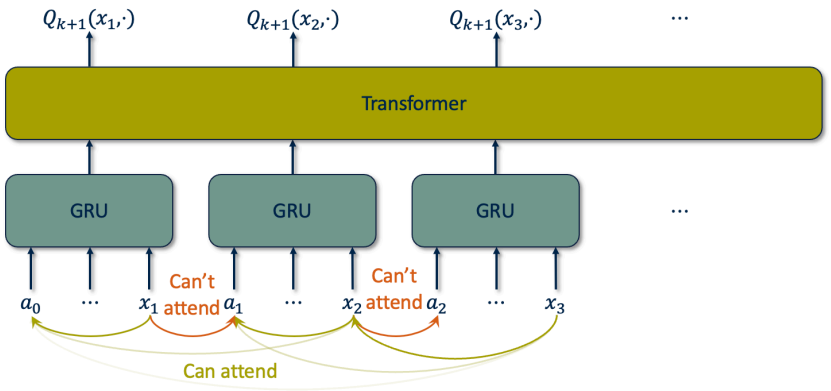


Figure 37: How causal masking prevents the model from attending to the action corresponding to the target Q-value.

good predictions of $Q(x_t, a)$ for $a = a_t$, the action that appears in the dataset, but not for the other actions in the action space. To prevent this degenerate outcome, we use masking to prevent the model from observing a_t when conditioning on x_t. This is also why we offset the observations and value predictions by one index, as in Figure 34. See Figure 37 for an illustration of how causal masking occludes the action corresponding to the target Q-value for each input transition.

One consequence of masking is that predictions for values early in the sequence are poor in comparison to predictions later in the sequence, since they benefit from less context. Repeated bootstrapping from these poor predictions propagates error throughout the sequence. To mitigate this, we rotate the sequence by fractions, retaining predictions from only the last fraction. For example, if we break the sequence into three equal fractions (X_1, X_2, X_3), we apply three rotations, yielding rotated sequences (X_1, X_2, X_3), (X_2, X_3, X_1), and (X_3, X_1, X_2). We pass each rotation through the model, and for each rotation, we retain only the predictions for X_3, X_1, and X_2 respectively. We use this rotation procedure to produce the Q estimates on line 15 of Algorithm 4.

Another important detail is that the bootstrap step on line 13 of Algorithm 4 leads to instability when generating targets for lower targets of k which the model has previously trained on. To mitigate this, we interpolate $\hat{Q}_{k+1}$ with its previous value, using an interpolation factor of .5, which we chose based on hyperparameter search.

Algorithm 5: Evaluating the Bellman Update Network.

input c, K, T ▷ Context length, iterations, evaluation length

$Q_0 \leftarrow \mathbf{0}$ ▷ Initialize Q-estimates to zero.

$(x_t, a_t, \pi(\cdot \mid x_t), r_t, b_t)_{t=0}^{c} \sim$ random behavior ▷ Fill transformer context with random behavior.

$x_{c+1} \leftarrow$ reset environment

for $t_0 = 1, ..., 1 + T$ **do**

  for $k = 0, ..., K$ **do**

    for $t = t_0, ..., t_0 + c$ **do**

      $V_k(x_t) \leftarrow \sum_a \pi(a \mid x_t) Q_k(x_t, a)$ ▷ Compute values from Q-values

    end for

    $h^{V_k} \leftarrow (a_t, \pi(\cdot \mid x_t), r_t, b_t, x_{t+1}, V_k(x_{t+1}))_{t=t_0}^{t_0+c}$ ▷ pair transitions with values

    $Q_{k+1} \leftarrow Q_\theta\left(h_{V_k}\right)$ ▷ Use the Bellman Update Network to estimate values.

  end for

  $t \leftarrow t_0 + c$

  $a_t \leftarrow \arg\max_a Q_{K+1}(x_t, a)$ ▷ Choose the action with the highest value.

  $r_t, b_t, x_{t+1} \leftarrow$ step environment with a_t

  $\pi(\cdot \mid x_t) \leftarrow$ one-hot(a_t) ▷ Use greedy policy for action logits

end for

4.2.6 Downstream Evaluation

Once the network is trained, we can use it to estimate values in a new setting by using the iterative method we described in Section 4.2. In addition, we can use the estimates to act, by choosing actions greedily by value (see Algorithm 5).

4.2.6.1 Policy Iteration

Note that acting in this way implements policy iteration, much like the algorithms discussed in previous chapters. As the model acts, it populates its own context with new actions and action-logits. Since the model has been trained on a variety of policies, it conditions its value estimates on these actions and logits and by transitivity, on the behavior policy. When we choose actions greedily, we improve on this behavior policy, completing the policy iteration cycle. Note that in practice, the context of the

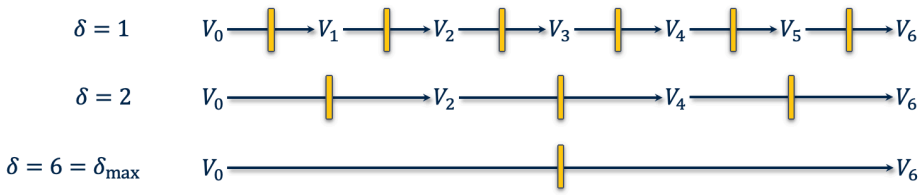


Figure 38: How the number of applications of the Bellman Update Network varies inversely with the value of δ.

model will contain a mixture of older, lower-quality actions and newer, higher-quality actions, with newer actions progressively dominating. We rely on the context-conditioning capability of the model to approximate a policy mixing the multitude of policies represented in the context.

4.2.7 Extension to multi-step Bellman Updates

The present formulation trains the Bellman Update Network to perform a single Bellman update. However, this can be generalized to multi-step updates, e.g. using the loss:

$$\mathcal{L}_\theta^\delta := -E\left[\sum_{t=1}^{T-1}\sum_{a\in\mathcal{A}} \log P_\theta\left(Q_k(x_t,a) \mid h_t^{Q_{k-\delta}}\right)\right] \tag{14}$$

where δ is some integer between 1 angitd $k-1$ (see Equation 11 for the definition of $h_t^{Q_{k-\delta}}$). In our experiments, we vary δ between 1 and the maximum number of iterations $\delta_{\max}$. We inversely vary K, the number of iterations in our evaluation (line 3 of Algorithm 6), so that $\delta \times k = \delta_{\max}$. Thus when $\delta = \delta_{\max}$, we perform $k = 1$ iterations, reducing the algorithm to the "naive" method described in Section 4.1.2. See Figure 38 for a visualization of the relationship between the number of iterations and the value of δ.

4.3 Related Work

An earlier work that anticipates many of the ideas used by Bellman Update Networks is Value Iteration Networks (Tamar et al. 2016). Like a Bellman Update Network, a Value Iteration Network uses a neural network to approximate a single step of value propagation and performs multiple steps of recurrent forward passes to produce an inference, with each new value estimate conditioned on a previous one. However, Value Iteration Networks do not target in-context learning, and instead the paper focuses on their ability to plan implicitly. Additionally, Value Iteration Networks rely on Convolutional Neural Networks which assume a representation of the environment in which the network can simultaneously observe the current state and those adjacent to it. As a result, the paper focuses exclusively on top-down 2D and graph navigation domains.

A more recent work that incorporates many related ideas is Procedure Cloning (Yang et al. 2022). In this work, the authors augment a behavior cloning dataset with information relating to the procedure used to choose an action. For example, in a maze environment, instead of cloning actions only, they also clone steps in a breadth-first-search algorithm used to choose those actions. Bellman Update Networks may be viewed as a specialization of this approach to the policy evaluation algorithm.

A variety of works, to include Schrittwieser et al. (2020), Okada and Taniguchi (2022), Zhu et al. (n.d.) and Wen et al. (2023) consider methods of planning in a latent space. We highlight two recent works in particular. Thinker (Chung et al. 2023) performs Monte-Carlo Tree Search entirely in latent space, with states augmented by anticipated rollout return and visit count. Another interesting work is Ritter et al.

(2020) which proposes "Episodic Planning Networks." This architecture augments the agent with an episodic memory that gets updated using a self-attention operation that iterates multiple times per step. The authors observe that the self-attention operation learns a kind of "value map" of states in the environment.

4.4 Experiments

Our experiments explore two settings: a tabular grid-world setting in which ground-truth values can be computed using classical policy evaluation and a continuous state, partially-observed domain implemented using Miniworld (Chevalier-Boisvert et al. 2023). In the first setting, we investigate two training regimes. The first regresses directly onto the ground-truth values, while the second incorporates the bootstrapped training regime described in Section 4.2.4. This allows us to disentangle the effects of the iterative value estimation method at the heart of the Bellman Update Network algorithm from the specific procedure used to train the network.

4.4.1 Training with ground-truth values

When regressing onto ground-truth values, we simply minimize Equation 13, regressing onto ground-truth values for $Q^n(x_t^n, a)$. Since we are able to optimize the value estimates for all actions, we dispense with masking (recall the discussion in Section 4.2.5) and positional encodings. This allows us to train estimates for all states and actions in the sequence simultaneously. To evaluate this network, we adapt the iterative procedure from Algorithm 5: we iteratively apply the network first to an initial Q estimate,

Algorithm 6: Tabular evaluation of the Bellman Update Network.

function $Q(h)$ ▷ Estimate values for all states in input sequence.
  $Q_0 \leftarrow \mathbf{0}$ ▷ Initialize Q-estimates to zero.
  for $k = 0, ..., K$ **do**
    for $x_t \in h$ **do**
      $V_k(x_t) \leftarrow \sum_a \pi(a \mid \cdot) Q_k(x_t, a)$ ▷ Compute values from Q-values
    end for
    $h^{V_k} \leftarrow (a_t, \pi(\cdot \mid x_t), r_t, b_t, x_{t+1}, V_k(x_{t+1}))_{t=0}^{c}$ ▷ pair transitions with values
    $Q_{k+1} \leftarrow Q_\theta(h^{V_k})$ ▷ Use Bellman Update Network to estimate values
  end for
  return Q_{K+1}
end function

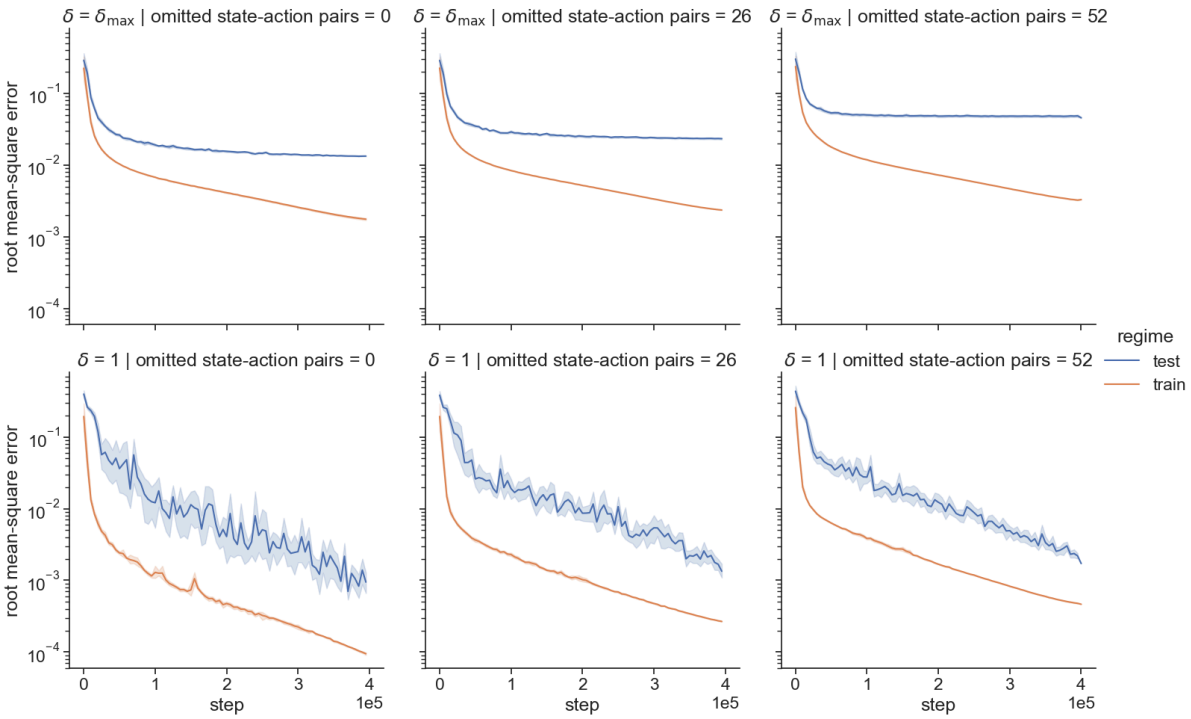


Figure 39: Comparison of root mean-square error for training vs. testing.

then auto-regressively to its own output (after computing value estimates from the Q estimates and the policy logits $\pi(\cdot \mid x_t)$ per Equation 12). For details see Algorithm 6.

4.4.1.1 Do value functions overfit?

The first point that we wish to demonstrate in this setting is that values conditioned on many policies are prone to overfitting. We therefore set $\delta = \delta_{\max}$ (Section 4.2.7) and train the network on 80,000 randomly sampled policies in a 5×5 grid-world, with a single goal of achievement. In this idealized setting, we provide the network with the full cross-product of states and actions, so that perfect estimation is possible. We evaluate the network in an identical setting but with 20,000 heldout policies. As the upper-left graph of Figure 39 illustrates, we observe a significant gap between training accuracy and test accuracy, as measured by root mean-square error. In addition, we observe that test error mostly plateaus after update 100,000, even as training error continues to decrease, indicating that all learning after this point entails memorization. In the right two graphs of Figure 39, we randomly omit $\frac{1}{4}$ and $\frac{1}{2}$ of the state-action pairs from the input. As the figures demonstrate, the gap between training and testing widens slightly and the extent of memorization increases.

4.4.1.2 Does value prediction with a Bellman Update Network mitigate overfitting?

In the lower half of Figure 39, we compare values estimated by the Bellman Update Network. Note that the "test" lines in Figure 39 describe error for the *full* value estimate produced by $\delta_{\max}$ steps of iteration (following the procedure described in Section 4.4.1), not the error for a single Bellman update. As the figure demonstrates, test error continues to diminish along with the training error, long after the test error for the $\delta_{\max}$ model has plateaued. While we observe a slight diminution in performance

as the number of omitted state-action pairs increases, the gap between train and test remains constant.

4.4.1.3 Do values predicted by a Bellman Update Network inform good policies?

The utility of a value function is not entirely captured by its accuracy, since an inaccurate value function can still induce a good policy. We therefore introduce the following procedure for evaluating value estimates in a tabular setting:

1. We perform a single step of policy improvement, choosing actions greedily by value estimate.
2. We use tabular policy evaluation to evaluate the resulting policy.
3. We compare the resulting value with the value of the optimal policy.

We refer to this metric as the "regret of the improved policy." Note that this bears some resemblance to the procedure described in Section 4.2.6 and Algorithm 5. However, the procedure does not require the model to auto-regressively consume the actions (and resulting transitions) produced by the new greedy policy and consequently there is no in-context learning.

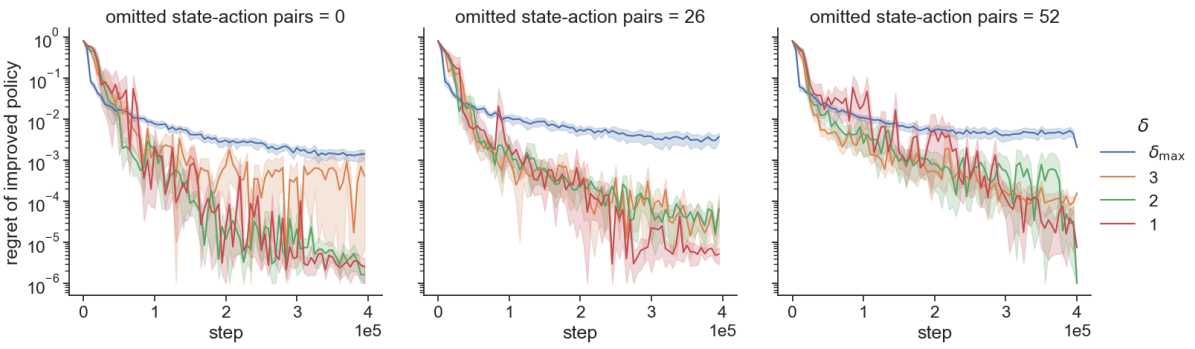


Figure 40: Regret of improved policy in the 5×5 grid-world, for different values of δ and different numbers of omitted state-action pairs.

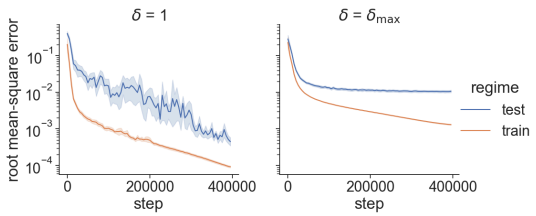


Figure 41: Root mean-square error on 5×5 grid-world with walls.

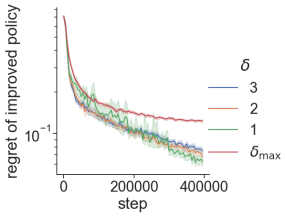


Figure 42: Regret of improved policy on 5×5 grid-world with walls.

As Figure 40 demonstrates, all models achieve good performance in this relatively simple setting. However, lower values of δ consistently outperform $\delta_{\max}$, indicating that the disparity in accuracy from Figure 39 does translate into performance. In general, $\delta = 1$ matches or slightly outperforms the higher values of δ.

4.4.1.4 Can Bellman Update Networks generalize to novel tasks?

Next we consider the effect of distribution shift in the environment dynamics from train to test. To this end, we introduce random walls into the grid-world with 25% probability per edge. The model does not observe walls and must infer their presence based on the transition patterns in the inputs — if the agent fails to move into an adjacent grid, this indicates the presence of a wall.

During testing, we evaluate the model on a randomly generated maze. This ensures that all grids are reachable, unlike the training setting in which grids may be walled off in some cases. As Figure 42 indicates, the model achieves better generalization performance when trained with lower values of δ. We also observe a similar generalization gap in Figure 41 as in Figure 39.

4.4.2 Training without ground-truth targets

Until this point we assumed access to a set of ground-truth values computed using tabular methods. In most realistic, non-tabular settings, we cannot make this assumption. In this section, we turn our attention to the algorithm proposed in Section 4.2.4, which uses a combination of curriculum-based training and bootstrapping to train the Bellman Update Network.

Algorithm 4 introduces a handful of difficulties not present in the previous section:

- The curriculum training approach introduces issues of non-stationarity.
- We can no longer assume complete coverage of state-action space, nor the capacity to sample this space IID, as we did in earlier experiments.
- The use of causal masking, as discussed in Section 4.2.5, further limits the information on which the model may condition its predictions.

In this section we test the ability of the transformer architecture to meet these challenges.

4.4.2.1 Can training without ground-truth targets yield accurate predictions?

Our first set of experiments in this new setting reproduces those the previous section, with a 5×5 grid-world using goals of achievement. Again, in order to give meaning to the accuracy estimates in Figure 43, we compare against a simple baseline, analogous to $\delta_{\max}$, which directly estimates $Q_{k=\infty}$. This baseline uses the same procedure as

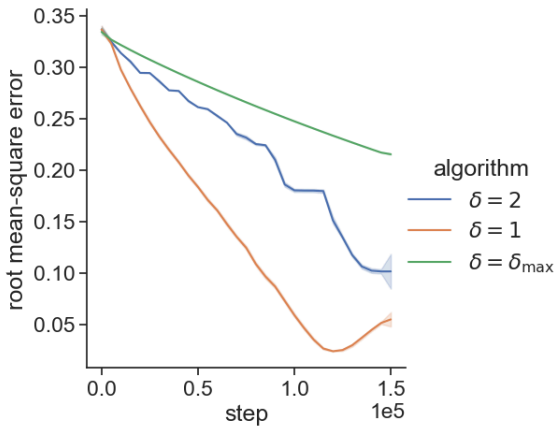


Figure 43: Root mean-square error of Bellman Update Network trained without ground-truth targets using Algorithm 4.

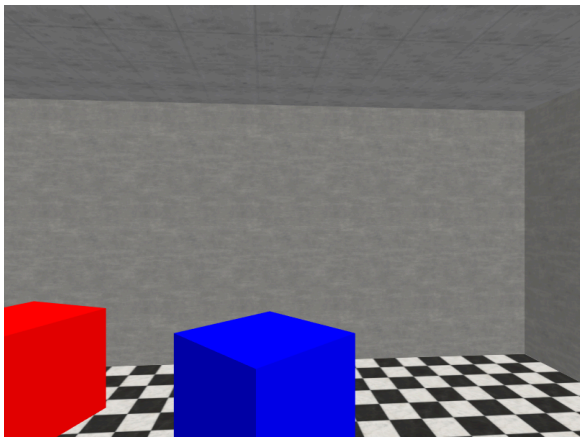


Figure 44: Example observation from Miniworld environment described in Section 4.4.2.2

the original (Algorithm 4), except for two changes. First, we eliminate the curriculum. Second, we eliminate k from our loss, minimizing

$$\mathcal{L}_\theta := \left(Q_\theta(x_t, a_t \mid h_t) - \left(r_t + \gamma E_{x_{t+1}, a_{t+1}}\left[\hat{Q}_\theta(x_{t+1}, a_{t+1})\right]\right)\right)^2 \tag{15}$$

instead of Equation 13. Here, $\hat{Q}_\theta$ is the output of Q_θ interpolated with previous values. To mitigate instability, we found it necessary to reduce the interpolation factor from 0.5 to 0.001, again using hyperparameter search. While lower values of δ clearly outperform higher values in Figure 43, we do observe some overfitting toward the end of training for $\delta = 1$.

4.4.2.2 Can a Bellman Update Network induce in-context reinforcement learning in a non-tabular setting?

Finally, we turn our attention to a non-tabular setting, implemented using Miniworld (Chevalier-Boisvert et al. 2023). Miniworld is a 3D domain in which the agent receives egocentric, RGB observations of the environment (see Figure 44). We adapt the

OneRoom environment to support multi-task training. We populate the environment with two random objects which the agent must visit in sequence. We encode the high-dimensional RGB observations used by Miniworld with a 3-layer convolutional network before feeding them into the GRU transition encoder (Section 4.2.5). To perform evaluations in this setting, we use the auto-regressive rollout mechanism described in Algorithm 5, choosing actions greedily by value estimate and feeding new transitions back into the network.

In our Miniworld experiments, we compare three settings of δ for the Bellman Update Network. Note that for $\delta = \delta_{\max}$, we eliminate the curriculum as discussed in Section 4.4.2.1. We also compare these with the Conservative Q-Learning (CQL) algorithm (Kumar et al. 2020), a state-of-the-art offline RL algorithm. Unlike the Bellman Update Network, CQL integrates policy improvement into its loss function, minimizing

$$\sum_{t=1}^{T-1} \left(Q_\theta\left(x_t, a_t \mid h_t^{V_k}\right) - \max_a \hat{Q}(x_t, a) \right)^2 \tag{16}$$

where $\hat{Q}$ is computed using bootstrapping. Note that the $\arg\max$ operation cannot be used in the loss when $k < \infty$. Therefore this form of policy improvement is not available to Bellman Update Networks. A well-known property of these kinds of losses is that

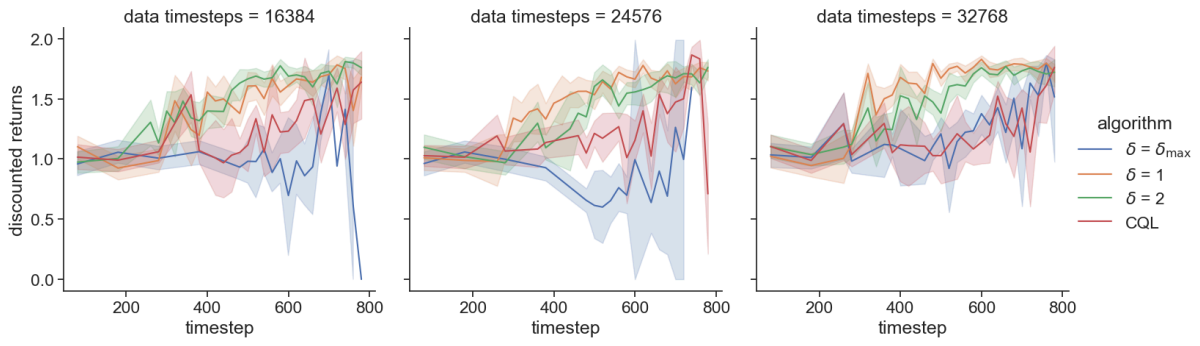


Figure 45: In-context reinforcement learning curves for Bellman Update Network and Conservative Q-Learning (CQL).

they tend to produce overly optimistic value estimates, due to the sensitivity of the $\max_a$ operator to noise. To mitigate this, CQL introduces a "conservative" auxiliary loss:

$$\alpha \sum_{t=1}^{T-1} \log \sum_{a} \exp(Q(x_t, a)) - Q(x_t, a_t) \tag{17}$$

Thus the algorithm used to train CQL departs from Algorithm 4 in only two ways:

1. We use the $\max_a$ targets from Equation 16 in place of the empirical targets from Equation 13.
2. We add the regularizer from Equation 17 to our loss function.

In Figure 45, we compare CQL with the Bellman Update Networks under several settings of δ. The error bands denote the standard error computed across 20 seeds. We also compare the results of training on different quantities of data by stopping the training of the source algorithm at different points in time. For reference, the middle graph in Figure 45, trained on 24,576 timesteps of data, terminates training just before the source algorithm reaches optimal performance.

Both CQL and $\delta = \delta_{\max}$ experience some instability for the lower data regimes, where the disparity in distribution between the policies represented in the training data and the optimal policy is greatest. Moreover, we observe that the area under the curve for CQL and $\delta = \delta_{\max}$ is smaller, perhaps reflecting limitations in the ability to generalize to the mixture policy observed during downstream evaluation. We also observe a slight advantage for $\delta = 1$ over $\delta = 2$ in the higher-data regimes, reflecting the disparity observed in our earlier grid-world results.

4.4.2.3 Qualitative analysis

We conclude by performing some qualitative analysis on the values learned by CQL and those learned by the Bellman Update Network. In Figure 46 and Figure 47, we visualize the value estimates of $\delta = 1$ and CQL on two trajectories from the offline data. In the diagram, the arrows represent the path taken by the agent. The double-headed arrows are color-coded to indicate predicted value (for the fore arrowhead) and experienced reward (for the aft arrowhead). The rings indicate the radius around the objects into which the agent must enter to receive reward. In Figure 46, the agent receives a cumulative reward of two, one for entering the perimeter of the blue circle and one for entering the perimeter of the red circle.

Note that the value predictions of the Bellman Update Network approach the maximum near the point where reward is actually received and then gradually anneal (as indicated by the yellow, orange, and red arrowheads). By contrast, the CQL predictions all appear to be at the maximum. Next we consider Figure 47, depicting a trajectory in which the agent receives no reward. Here we see much lower value predictions by CQL even as the agent approaches the rewarding blue circle. In contrast, the predictions made by the Bellman Update Network are similar in distribution to those in Figure 46. This provides one example in which CQL anticipates the remainder of a trajectory, implying memorization. The Bellman Update Network does not anticipate the remainder of the trajectory and its value predictions reflect the dynamics of the environment.

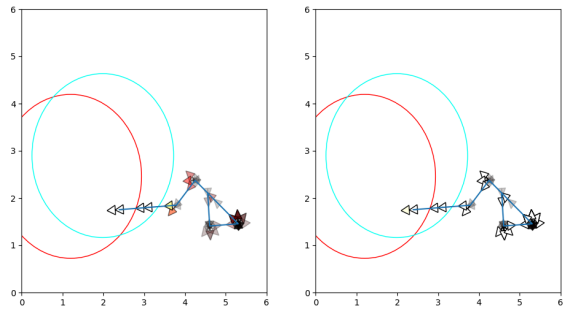


Figure 46: Predictions by the $\delta = 1$ variant (left) and by CQL (right) on an offline trajectory with cumulative return of 2.

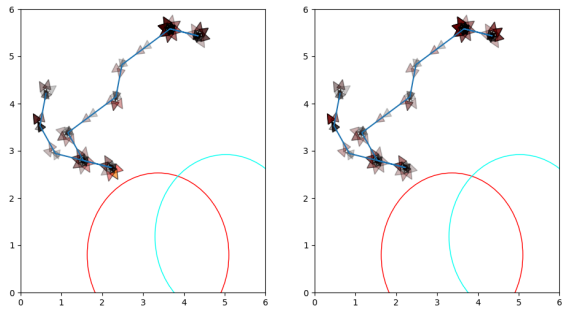


Figure 47: Predictions by the $\delta = 1$ variant (left) and by CQL (right) on an offline trajectory with cumulative return of 0.

4.5 Conclusion

This chapter presents an algorithm for performing in-context reinforcement learning. It imports many of the concepts from preceding chapters, especially the integration of context-based learning into the policy iteration algorithm. The chapter builds on the work presented in the preceding chapters by freeing the algorithm from model-based learning and monte-carlo rollouts. We observe that Bellman Update Networks are better equipped to handle high-dimensional observation spaces (like Miniworld) than AD++, since observations of this kind pose significant challenges for existing modeling approaches, especially where partial observability is involved. Certainly observations of this size cannot be modeled incrementally using the inline, sequence-based approach proposed by Janner et al. (2021a), since a single observation would consume an entire context window.

That said, AD++ retains some advantages over Bellman Update Networks. In particular, this approach may struggle to propagate values over very long timesteps (e.g. over 100), and certainly training a network for such a task could take a very long

time. It is likely that such a setting would benefit from values of δ higher than $\delta = 1$, and this should be thought of as a parameter to tune.

In general, a limitation of the approach proposed in this chapter is that it requires learning values for a very large number of policies, whereas only the optimal is ultimately of interest. However, learning only the optimal policy is in general not possible within the paradigm of in-context reinforcement learning, which requires an algorithm to yield a spectrum of policies transitioning from exploratory to exploitative behavior.

5 Conclusion

Since beginning of this work, the science of artificial intelligence has undergone a paradigm shift. Not only have foundation models, particularly language models, come to dominate the field, but priorities and expectations around research have shifted dramatically. In some ways, the work in this thesis aligns with these shifts. The thesis anticipates a world in which RL algorithms acquire most of their knowledge through supervised training on offline datasets, as we discussed in the introduction 1. In this paradigm, learning does not happen in a single unbroken arc, from tabula-rasa random weights to expert behavior, all driven by end-to-end gradient-descent-based deep RL algorithms. Instead, learning happens in *two* stages: an initial stage in which the model soaks up large quantities of information using strong supervised signals from offline datasets, and a second stage in which the model adapts to its specific setting using some form of in-context learning.

However, in some ways, this thesis is out of step with the current trajectory of AI research. In particular, it focuses on two concerns which have fallen out of favor:

1. reinforcement learning as a method of exploration and discovery.
2. generalization instead of memorization.

We will begin by describing the ways in which attitudes towards these concerns have changed, and make the case for their continued relevance.

5.1 Reinforcement Learning as a Method of Exploration and Discovery

Reinforcement learning remains a component of some state-of-the-art AI systems, but it's role has fundamentally changed from what its pioneers envisioned. Early deep RL researchers imagined agents that would explore their world with very little learning signal, progressively acquire knowledge and skills, and slowly but surely improve their behavior until optimal. Agents would acquire auxiliary skills as necessary in the service of a simple, sparse reward (Silver et al. 2021), rather than optimizing those skills directly. One of the most appealing aspects of this program was its rigorous economy in aligning objectives: all learning would serve the accumulation of reward. An agent might acquire language, for example, but only to the extent necessary for communicating concepts essential to its mission. This ethos dictated that RL would gradually shed crutches like reward shaping (Hu et al. 2020) and auxiliary losses (Burda et al. 2018) as research matured and discovered new pathways between short-term behavior and long-term reward.

In Yann LeCun's 2016 NeurIPS keynote (LeCun 2016), he laid out a paradigm that came to be known as "LeCake":

- Unsupervised learning is the "filling" of the cake, accounting for millions of bits of learning per sample.
- Supervised learning is the "icing," accounting for 10-10,000 bits per sample.
- Reinforcement learning is the "cherry on top," accounting for a few bits per sample.

The talk caused a stir because it minimized the role of reinforcement learning, then the dominant form of learning for AI problems. His argument was that there was simply not enough information in reward signal to train the large neural networks that were then coming to prominence in fields like computer vision.

In retrospect, the talk was incredibly prescient. Not only did the role of reinforcement learning in large-scale systems diminish, but many of the problems that were once thought to be its exclusive purview, such as credit assignment, are now routinely ignored. Reinforcement learning still plays a role in the training of some LLMs, but primarily as a fine-tuning step during Reinforcement Learning from Human Feedback (RLHF), after the bulk of training time has been spent on unsupervised learning — indeed the "cherry on top."

One problem that receives the most shocking neglect from modern systems and which was once the very raison d'etre of reinforcement learning is "credit assignment." This is the problem of determining which actions, in a temporally extended sequence, are responsible for a final outcome, good or bad. It is through credit assignment, for example, that we learn that losing our queen on the 20th step of a chess game is detrimental, even though we might not actually lose the game for another 80 turns. Technically, language entails credit-assignment, since a particular conclusion may only be reachable through a long sequence of thoughts, calculations or questions.

While many state-of-the-art models are not publicly documented (GPT-4, Gemini, etc.) and others are cagey about the details of RLHF (Lieber et al. 2021), it is clear that models increasingly ignore the credit-assignment problem by maximizing

one-step reward (Touvron et al. 2023) – and despite this, achieve top performance. Direct Preference Optimization (Rafailov et al. 2023), an increasingly popular technique for optimizing learned reward, actually brands itself as "RL-free" and completely disregards credit-assignment.

What explains the precipitous shift in the role of RL in modern AI systems? In short, a tradeoff: between the kind of rigorous alignment that RL prioritizes and the kind of scale necessary to train most practical AI systems. Aligning behavior with reward requires credit-assignment. Realistic settings require long-term credit assignment under conditions of noise and uncertainty. Learning signal under these conditions is inherently weak. Strong learning signal is necessary to train large networks on large amounts of data. When GPT3 (Brown et al. 2020a) demonstrated the indisputable power of scale, approaches that were incompatible began to fall out of favor.

Another important shift is that in many practical settings, imitation is enough. To reiterate, many natural language tasks do entail credit assignment, especially when multi-step, exploratory reasoning is involved (Wei et al. 2022). However, language models inherit a kind of imperfect credit assignment through imitation of the humans who wrote their data (who already possess the capacity for credit assignment). RL provides a framework for learning *optimal* credit assignment, but this framework does not empirically work in realistic language settings. Of course, an unprincipled solution that works is better than a principled solution that doesn't.

5.2 Generalization and Memorization

Another significant shift in thinking is the attitude toward memorization, once thought to be synonymous with generalization. Classical frameworks like the bias-variance tradeoff (Franklin 2005) imply that modeling noise will lead to overfitting — failure to generalize from training data to out-of-distribution test data — and that regularization of some kind or truncation of training is necessary to prevent this. However, several publications have documented the mismatch between these predictions and empirical reality (Brown et al. 2021; Zhang et al. 2021). Some work has also presented theoretical frameworks for understanding this mismatch (Feldman 2020).

In general, there has been a movement away from classical regularization techniques that effectively limit model expressivity in order to discourage memorization and encourage generalization, and a general recognition that these two tendencies may be more in cooperation than in conflict (Tirumala et al. 2022). This general trend has conveniently aligned with the rise of learning at large scales, which presents more opportunities for memorization due to the size of the architectures, and fewer costs due to the size of the training data. When training data is large enough, there is no such thing as "out-of-distribution." When all test data already appears in some form in the training data, the problem of generalization becomes obsolete.

5.3 The continued relevance of reinforcement learning and generalization

All three chapters of this thesis concern themselves extensively with temporally-extended reinforcement learning and generalization to unseen settings. How do we justify this focus, given the current trajectory of AI research?

5.3.1 Language privileges imitation

For all its success in the realm of language, imitative learning faces challenges in other domains. Language lends itself especially well to imitation — a model can reproduce the exact words in its dataset. In contrast, other domains do not permit this kind of exact reproduction, requiring the transfer of knowledge and skills from one setting to a different one. In robotics, for example, the same behavior requires widely different policies for different physical embodiments. The joint activation patterns that a 300-lb steel robot must use to walk are significantly different from those of a 150-lb human. For any robot to acquire some skill from existing embodiments, in the way that an LLM acquires language understanding from human datasets, it must overcome a significant problem of transfer.

Zero-shot transfer of the kind exhibited by LLMs is not likely for robots, especially if they learn through extensive memorization. Instead, some period of fine-tuning or in-context learning will be necessary to adapt fundamental skills acquired from offline data to specific embodiments and settings. Unlike language models, which acquire credit-assignment strategies whole-cloth from their source data, robots will need to

adapt these strategies and learn new ones. Some framework like RL, capable of evaluating and improving credit assignment, will be necessary. Andrychowicz et al. (2020) offer one compelling approach to this problem.

5.3.2 Credit assignment is necessary for expertise

Advances in the technologies that power LLMs have been rapid and difficult to predict. However, a general trend is that they improve the ability of models to imitate their sources, not their ability to outperform them by any significant margin. As a rule, LLMs do not outperform top experts in any field. This stands in stark contrast to agents trained using reinforcement learning which, to date, stand at or near the top of their class in several domains including Go (Silver et al. 2016), Starcraft (Vinyals et al. 2019), and DoTA (Berner et al. 2019).

Many areas of expertise entail reasoning or decision-making over multiple steps. This is especially true of many of the loftier aspirations for AI systems, including scientific discovery and artistic creativity. A program that neglects credit-assignment can acquire human-level credit assignment through imitation. To a degree, it can acquire super-human expertise through fine-tuning of the final result. However, the expertise of its final inference or output will fundamentally be limited by the multi-step process which led to the inference. Improving the process with respect to the final result requires some form of credit-assignment.

5.3.3 Generalization is necessary for expertise

Finally, we argue that useful expertise will always entail transfer and generalization – that is, a meaningful gap between training data and the domain of interest. Whatever their limitations in terms of expertise, LLMs unquestionably distinguish themselves by their breadth — their capacity to produce coherent, usually intelligent, responses to almost any prompt. A system that combines the expertise of RL with the breadth of LLMs does not currently exist. Therefore, the first of its kind will need to distill expertise from *specialists* and then generalize it beyond those specializations. For example, a model may acquire knowledge of advanced mathematics by training on textbooks, or even by distilling the behaviors of RL agents specialized to certain mathematical problems, like AlphaTensor (Fawzi et al. 2022). However, in order to advance the frontiers of science, such a model would need to transfer this specialized intelligence to other domains, domains which are not captured by any dataset since they potentially lie beyond the limits of current human understanding.

5.4 Foundation models for reinforcement learning

A consequence of these reflections is that foundation models for RL will not look the same as foundation models for language. "LeCake" provides a sketch that may still apply — surely, RL agents can benefit from data beyond their immediate experience. However, the exact program will need to be different in significant ways. This thesis has offered some tentative ideas about RL foundation models. In the first chapter, we observe that any sequence-based foundation model can, in principle, serve as a kind of

RL foundation model. In the second, we demonstrate that such a model gains capacity when specialized to RL data. Through the incorporation of Algorithm Distillation, we highlight the fact that such a model can not only distill the dynamics and policies in the source data but also the *learning operator* of the source algorithm. In the final chapter, we argue that such a model can benefit from the idea of value and can learn representations that generalizes.

The broader questions remain — about credit-assignment, generalization, and the challenges of scale. However, given the history and magnitude of innovations in this vibrant community of research, we can be sure that revolutionary developments in the science of RL foundation models are still to come.

Bibliography

Abid, A., M. Farooqi, and J. Zou. 2021. "Persistent anti-muslim bias in large language models". *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, 298–306

Ahn, M., A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, and M. Yan. 2022. "Do As I Can, Not As I Say: Grounding Language in Robotic Affordances". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2204.01691

Ammanabrolu, P., and M. Riedl. 2021. "Learning Knowledge Graph-based World Models of Textual Environments". *Advances in Neural Information Processing Systems*, 3720–3731. Curran Associates, Inc.

Andrychowicz, O. M., B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, and others. 2020. "Learning dexterous in-hand manipulation". *The International Journal of Robotics Research*, 39 (1): 3–20. SAGE Publications Sage UK: London, England

Baker, B., I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. 2022. "Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2206.11795

Berner, C., G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, and others. 2019. "Dota 2 with large scale deep reinforcement learning". *arXiv preprint arXiv:1912.06680*

Brown, G., M. Bun, V. Feldman, A. Smith, and K. Talwar. 2021. "When is memorization of irrelevant training data necessary for high-accuracy learning?". *Proceedings of the 53rd annual ACM SIGACT symposium on theory of computing*, 123–132

Brown, T., 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. 2020b. "Language Models are Few-Shot Learners". *Advances in Neural Information Processing Systems*, 1877–1901. Curran Associates, Inc.

Brown, T., B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, and others. 2020a. "Language models are few-shot learners". *Advances in neural information processing systems*, 33: 1877–1901

Burda, Y., H. Edwards, A. Storkey, and O. Klimov. 2018. "Exploration by random network distillation". *arXiv preprint arXiv:1810.12894*

Chan, S. C. Y., A. Santoro, A. K. Lampinen, J. X. Wang, A. Singh, P. H. Richemond, J. McClelland, and F. Hill. 2022a. "Data Distributional Properties Drive Emergent In-Context Learning in Transformers". arXiv. Accessed August 11, 2022. http://arxiv.org/abs/2205.05055

Chan, S. C., A. Santoro, A. K. Lampinen, J. X. Wang, A. K. Singh, P. H. Richemond, J. McClelland, and F. Hill. 2022b. "Data distributional properties drive emergent in-context learning in transformers". *Advances in Neural Information Processing Systems*

Chen, C., Y.-F. Wu, J. Yoon, and S. Ahn. 2022. "Transdreamer: Reinforcement learning with transformer world models". *arXiv preprint arXiv:2202.09481*

Chen, L., K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. 2021b. "Decision Transformer: Reinforcement Learning via Sequence Modeling". *arXiv:2106.01345 [cs]*

Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, and others. 2021a. "Evaluating large language models trained on code". *arXiv preprint arXiv:2107.03374*

Chen, M., J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F.

Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. 2021c. *Evaluating Large Language Models Trained on Code*

Chen, Y., C. Zhao, Z. Yu, K. McKeown, and H. He. 2022. "On the Relation between Sensitivity and Accuracy in In-context Learning". arXiv. Accessed September 28, 2022. http://arxiv.org/abs/2209.07661

Chevalier-Boisvert, M., B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. 2023. "Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks". *CoRR*

Cho, K., B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". *arXiv preprint arXiv:1406.1078*

Chung, S., I. Anokhin, and D. Krueger. 2023. "Thinker: Learning to Plan and Act". *arXiv preprint arXiv:2307.14993*

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. 2016. "RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning". arXiv. Accessed August 11, 2022. http://arxiv.org/abs/1611.02779

Fawzi, A., M. Balog, B. Romera-Paredes, D. Hassabis, and P. Kohli. 2022. "Discovering novel algorithms with AlphaTensor"

Feldman, V. 2020. “Does learning require memorization? a short tale about a long tail”. *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 954–959

Finn, C., P. Abbeel, and S. Levine. 2017. “Model-agnostic meta-learning for fast adaptation of deep networks”. *International conference on machine learning*, 1126–1135

Franklin, J. 2005. “The elements of statistical learning: data mining, inference and prediction”. *The Mathematical Intelligencer*, 27 (2): 83–85. Springer

French, R. M. 1999. “Catastrophic forgetting in connectionist networks”. *Trends in cognitive sciences*, 3 (4): 128–135. Elsevier

Fried, D., A. Aghajanyan, J. Lin, S. Wang, E. Wallace, F. Shi, R. Zhong, W.-t. Yih, L. Zettlemoyer, and M. Lewis. 2022. “InCoder: A Generative Model for Code Infilling and Synthesis”. *arXiv:2204.05999 [cs]*

Fujimoto, S., D. Meger, and D. Precup. 2019. “Off-policy deep reinforcement learning without exploration”. *International conference on machine learning*, 2052–2062

Gao, L., S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy. 2020. “The Pile: An 800GB Dataset of Diverse Text for Language Modeling”. *arXiv:2101.00027 [cs]*

Garg, D., S. Vaidyanath, K. Kim, J. Song, and S. Ermon. 2022. “LISA: Learning Interpretable Skill Abstractions from Language”. *arXiv:2203.00054 [cs]*

Garg, S., D. Tsipras, P. Liang, and G. Valiant. 2022. "What Can Transformers Learn In-Context? A Case Study of Simple Function Classes". arXiv. Accessed September 28, 2022. http://arxiv.org/abs/2208.01066

Grimm, C., A. Barreto, S. Singh, and D. Silver. 2020. "The value equivalence principle for model-based reinforcement learning". *Advances in Neural Information Processing Systems*, 33: 5541–5552

Hafner, D., T. Lillicrap, M. Norouzi, and J. Ba. 2020. "Mastering atari with discrete world models". *arXiv preprint arXiv:2010.02193*

Hill, F., S. Mokra, N. Wong, and T. Harley. 2020. "Human Instruction-Following with Deep Reinforcement Learning via Transfer-Learning from Text". *arXiv: 2005.09382 [cs]*

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

Hu, Y., W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan. 2020. "Learning to utilize shaping rewards: A new approach of reward shaping". *Advances in Neural Information Processing Systems*, 33: 15931–15941

Huang, W., P. Abbeel, D. Pathak, and I. Mordatch. 2022a. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents". *arXiv:2201.07207 [cs]*

Huang, W., F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter. 2022b. "Inner Monologue: Embodied Reasoning

through Planning with Language Models". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2207.05608

Janner, M., Q. Li, and S. Levine. 2021b. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". *Advances in Neural Information Processing Systems*

Janner, M., Q. Li, and S. Levine. 2021a. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2106.02039

Karimi Mahabadi, R., L. Zettlemoyer, J. Henderson, L. Mathias, M. Saeidi, V. Stoyanov, and M. Yazdani. 2022. "Prompt-free and Efficient Few-shot Learning with Language Models". *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3638–3652. Dublin, Ireland: Association for Computational Linguistics

Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, and others. 2017. "Overcoming catastrophic forgetting in neural networks". *Proceedings of the national academy of sciences*, 114 (13): 3521–3526. National Acad Sciences

Kumar, A., A. Zhou, G. Tucker, and S. Levine. 2020. "Conservative q-learning for offline reinforcement learning". *Advances in Neural Information Processing Systems*, 33: 1179–1191

Lake, B. M., T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. 2017. "Building machines that learn and think like people". *Behavioral and brain sciences*, 40: e253. Cambridge University Press

Laskin, M., L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. Hansen, A. Filos, E. Brooks, and others. 2022. "In-context reinforcement learning with algorithm distillation". *arXiv preprint arXiv:2210.14215*

LeCun, Y. 2016. "Predictive Learning". *Proceedings of the 30th International Conference on Neural Information Processing Systems (NeurIPS)*

Lee, J. N., A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill. 2023. "Supervised Pretraining Can Learn In-Context Reinforcement Learning". *arXiv preprint arXiv:2306.14892*

Lee, K.-H., O. Nachum, M. Yang, L. Lee, D. Freeman, W. Xu, S. Guadarrama, I. Fischer, E. Jang, H. Michalewski, and I. Mordatch. 2022. "Multi-Game Decision Transformers". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2205.15241

Li, S., X. Puig, C. Paxton, Y. Du, C. Wang, L. Fan, T. Chen, D.-A. Huang, E. Akyürek, A. Anandkumar, J. Andreas, I. Mordatch, A. Torralba, and Y. Zhu. 2022. "Pre-Trained Language Models for Interactive Decision-Making". *arXiv:2202.01771 [cs]*

Li, X. L., and P. Liang. 2021. "Prefix-Tuning: Optimizing Continuous Prompts for Generation". https://doi.org/10.48550/arXiv.2101.00190

Liang, P. P., C. Wu, L.-P. Morency, and R. Salakhutdinov. 2021. "Towards understanding and mitigating social biases in language models". *International Conference on Machine Learning*, 6565–6576

Lieber, O., O. Sharir, B. Lenz, and Y. Shoham. 2021. "Jurassic-1: Technical details and evaluation". *White Paper. AI21 Labs*, 1

Lu, K., A. Grover, P. Abbeel, and I. Mordatch. 2021. "Pretrained Transformers as Universal Computation Engines". *arXiv:2103.05247 [cs]*

Madaan, A., N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, and others. 2023. "Self-refine: Iterative refinement with self-feedback". *arXiv preprint arXiv:2303.17651*

Majumdar, A., A. Shrivastava, S. Lee, P. Anderson, D. Parikh, and D. Batra. 2020. "Improving Vision-and-Language Navigation with Image-Text Pairs from the Web". *arXiv:2004.14973 [cs]*

Micheli, V., E. Alonso, and F. Fleuret. 2022. "Transformers are sample efficient world models". *arXiv preprint arXiv:2209.00588*

Min, S., X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. 2022. "Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?". *arXiv:2202.12837 [cs]*

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, and others. 2015. "Human-level control through deep reinforcement learning". *nature*, 518 (7540): 529–533. Nature Publishing Group

Nye, M., A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, C. Sutton, and A. Odena. 2021. "Show Your Work: Scratchpads for Intermediate Computation with Language Models". arXiv. Accessed August 11, 2022. http://arxiv.org/abs/2112.00114

Okada, M., and T. Taniguchi. 2022. "DreamingV2: Reinforcement learning with discrete world models without reconstruction". *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 985–991

Pan, X., X. Chen, Q. Zhang, and N. Li. 2022. "Model Predictive Control: A Reinforcement Learning-based Approach". *Journal of Physics: Conference Series*, 12058

Pan, Y., L. Pan, W. Chen, P. Nakov, M.-Y. Kan, and W. Y. Wang. 2023. "On the Risk of Misinformation Pollution with Large Language Models". *arXiv preprint arXiv:2305.13661*

Peng, X., M. O. Riedl, and P. Ammanabrolu. 2021. "Inherently Explainable Reinforcement Learning in Natural Language". *arXiv:2112.08907 [cs]*

Pinon, B., J.-C. Delvenne, and R. Jungers. 2022. "A model-based approach to meta-Reinforcement Learning: Transformers and tree search". *arXiv preprint arXiv: 2208.11535*

Radford, A., J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. 2019. "Language Models are Unsupervised Multitask Learners"

Rafailov, R., A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. 2023. "Direct preference optimization: Your language model is secretly a reward model". *arXiv preprint arXiv:2305.18290*

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

Raparthy, S. C., E. Hambro, R. Kirk, M. Henaff, and R. Raileanu. 2023. "Generalization to New Sequential Decision Making Tasks with In-Context Learning". *arXiv preprint arXiv:2312.03801*

Reed, S., K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. 2022. "A Generalist Agent". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2205.06175

Ritter, S., R. Faulkner, L. Sartran, A. Santoro, M. Botvinick, and D. Raposo. 2020. "Rapid task-solving in novel environments". *arXiv preprint arXiv:2006.03662*

Robine, J., M. Höftmann, T. Uelwer, and S. Harmeling. 2023. "Transformer-based World Models Are Happy With 100k Interactions". *arXiv preprint arXiv:2303.07109*

Schmidhuber, J. 1987. "Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook"

Schrittwieser, J., I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and others. 2020. "Mastering atari, go, chess and shogi by planning with a learned model". *Nature*, 588 (7839): 604–609. Nature Publishing Group UK London

Seo, Y., K. Lee, F. Liu, S. James, and P. Abbeel. 2022. "HARP: Autoregressive Latent Video Prediction with High-Fidelity Image Generator". arXiv. Accessed September 28, 2022. http://arxiv.org/abs/2209.07143

Shinn, N., F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. 2023. "Reflexion: Language Agents with Verbal Reinforcement Learning"

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

Silver, D., S. Singh, D. Precup, and R. S. Sutton. 2021. "Reward is enough". *Artificial Intelligence*, 299: 103535. Elsevier

Singh, A. K., D. Ding, A. Saxe, F. Hill, and A. K. Lampinen. 2022. "Know your audience: specializing grounded language models with the game of Dixit". arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2206.08349

Singh, I., G. Singh, and A. Modi. 2021. "Pre-trained Language Models as Prior Knowledge for Playing Text-based Games". *arXiv:2107.08408 [cs]*

Stadie, B. C., G. Yang, R. Houthooft, X. Chen, Y. Duan, Y. Wu, P. Abbeel, and I. Sutskever. 2018. "Some considerations on learning to explore via meta-reinforcement learning". *arXiv preprint arXiv:1803.01118*

Sutton, R. S., and A. G. Barto. 2018. *Reinforcement learning: An introduction*. MIT press

Tam, A. C., N. C. Rabinowitz, A. K. Lampinen, N. A. Roy, S. C. Y. Chan, D. J. Strouse, J. X. Wang, A. Banino, and F. Hill. 2022. "Semantic Exploration from Language Abstractions and Pretrained Representations". *arXiv:2204.05080 [cs]*

Tamar, A., Y. Wu, G. Thomas, S. Levine, and P. Abbeel. 2016. "Value iteration networks". *Advances in neural information processing systems*, 29

Tamkin, A., M. Brundage, J. Clark, and D. Ganguli. 2021. "Understanding the capabilities, limitations, and societal impact of large language models". *arXiv preprint arXiv:2102.02503*

Tarasov, D., V. Kurenkov, and S. Kolesnikov. 2022. "Prompts and Pre-Trained Language Models for Offline Reinforcement Learning"

Team, A. A., J. Bauer, K. Baumli, S. Baveja, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, and others. 2023. "Human-Timescale Adaptation in an Open-Ended Task Space". *arXiv preprint arXiv: 2301.07608*

Tirumala, K., A. Markosyan, L. Zettlemoyer, and A. Aghajanyan. 2022. "Memorization without overfitting: Analyzing the training dynamics of large language models". *Advances in Neural Information Processing Systems*, 35: 38274–38290

Todorov, E., T. Erez, and Y. Tassa. 2012. “Mujoco: A physics engine for model-based control”. *2012 IEEE/RSJ international conference on intelligent robots and systems*, 5026–5033

Touvron, H., L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, and others. 2023. “Llama 2: Open foundation and fine-tuned chat models”. *arXiv preprint arXiv:2307.09288*

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017b. “Attention Is All You Need”. arXiv. Accessed August 11, 2022. http://arxiv.org/abs/1706.03762

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. 2017a. “Attention is all you need”. *Advances in neural information processing systems*, 30

Vinyals, O., I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, and others. 2019. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. *Nature*, 575 (7782): 350–354. Nature Publishing Group UK London

Wang, B., and A. Komatsuzaki. 2021. “GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model”. https://github.com/kingoflolz/mesh-transformer-jax

Wei, J., Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J.

Dean, and W. Fedus. 2022. “Emergent Abilities of Large Language Models”. arXiv. Accessed August 11, 2022. http://arxiv.org/abs/2206.07682

Wei, J., X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou. 2022. “Chain of Thought Prompting Elicits Reasoning in Large Language Models”. *arXiv: 2201.11903 [cs]*

Wen, L., S. Zhang, H. E. Tseng, and H. Peng. 2023. “Dream to Adapt: Meta Reinforcement Learning by Latent Context Imagination and MDP Imagination”. *arXiv preprint arXiv:2311.06673*

Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. 2020. “Transformers: State-of-the-Art Natural Language Processing”. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. Online: Association for Computational Linguistics

Xu, M., Y. Shen, S. Zhang, Y. Lu, D. Zhao, J. B. Tenenbaum, and C. Gan. 2022. “Prompting Decision Transformer for Few-Shot Policy Generalization”. arXiv. Accessed August 9, 2022. http://arxiv.org/abs/2206.13499

Yang, M. S., D. Schuurmans, P. Abbeel, and O. Nachum. 2022. “Chain of thought imitation with procedure cloning”. *Advances in Neural Information Processing Systems*, 35: 36366–36381

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals. 2021. "Understanding deep learning (still) requires rethinking generalization". *Communications of the ACM*, 64 (3): 107–115. ACM New York, NY, USA

Zhang, S., S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer. 2022. "OPT: Open Pre-trained Transformer Language Models"

Zhu, W., M. Okada, and T. Taniguchi. n.d. "Mastering Robotic Skills in Real Visual Worlds through Model-based Reinforcement Learning"

Zintgraf, L., K. Shiarlis, M. Igl, S. Schulze, Y. Gal, K. Hofmann, and S. Whiteson. 2019. "Varibad: A very good method for bayes-adaptive deep rl via meta-learning". *arXiv preprint arXiv:1910.08348*