

Reinforcement learning from imperfect data

Yicheng Luo

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
of
University College London.

Department of Computer Science
University College London

March 8, 2025

I, Yicheng Luo, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Reinforcement learning (RL) is a powerful branch of machine learning where agents learn to make decisions by interacting with an environment to maximize cumulative rewards. However, traditional RL methods often face the challenge of high sample complexity, requiring vast amounts of online interactions to achieve proficiency. This thesis delves into the untapped potential of leveraging pre-existing, imperfect data—such as sub-optimal experiences, incomplete datasets, and unstructured data—to reduce reliance on costly, high-quality datasets, making RL more practical for real-world applications.

This thesis makes three key contributions:

1. We investigate the benefits and trade-offs of RL algorithms that build on sub-optimal experiences. By studying how these algorithms can capitalize on imperfect data, we enable more sample-efficient learning and achieve performance unattainable through offline learning alone.
2. We introduce a new offline imitation learning algorithm designed to handle diverse, reward-free datasets. This approach allows learning from mixed-quality demonstrations, reducing the need for meticulously annotated behavior data, which is often challenging to obtain.
3. We present a new dataset and benchmark that uses unstructured data from chess to explore RL in a new dimension. This dataset bridges behavior and language, opening the door to the development of generalist agents capable of learning from unstructured, real-world information.

Our work demonstrates that incorporating imperfect data into RL frameworks can significantly reduce sample complexity, broadening the horizons for RL

applications in more complex, data-scarce environments. These advances offer promising new directions for future research and the practical deployment of RL in diverse domains.

Acknowledgements

I would like to express my deepest gratitude to my advisors, Marc Deisenroth and Edward Grefenstette, for their unwavering support throughout my studies. I have been fortunate to work with Marc since my undergraduate years at Imperial, and I am profoundly thankful for the many invaluable lessons he has imparted, both in research and beyond. His mentorship has been a cornerstone of my academic journey. I am equally grateful to Ed, whose guidance has instilled in me a problem-driven approach to research and the confidence to tackle even the most challenging problems.

My heartfelt thanks go to my friends, colleagues, and collaborators at Imperial and UCL, whose friendship and support have made this journey truly special. I am especially thankful to Daniel Giles, Sicelukwanda Zwane, Mathieu Alain, Vignesh Gopakumar, Jackie Kay, Noah Siegel, Daniel Augusto de Souza, Jake Cunningham, Samuel Cohen, James Rudd-Jones, Chong Chen, Dongmei Mo, Florian Pfaff, Kenta Hoshino, Yasemin Bekiroglu, So Takao, Giovanni Franzese, Oscar Key, Daniel Tan, Fabian Paischer, Alexander Terenin, Mihaela Rosca, Zhengyao Jiang, Yinchen Xu, Xidong Feng for the stimulating discussions and exchange of ideas that have enriched my work.

To my dear friends Cassie, Lulu, Nan, James, Chunyi, Husheng, and Fangyi: starting my PhD during the challenging times of COVID-19 would have been far more difficult without your companionship and support. I am immensely grateful for the solace and encouragement you've provided.

A special thank you to Audrey — your presence has been a constant source of strength during this demanding phase of my life. None of this would have been possible without your willingness to listen to me talk endlessly about computers and AI over the past four years. To the moon and back, thank you.

Finally, I want to offer my deepest gratitude to my family, whose support has been immeasurable and unconditional in ways I can never fully repay.

This research is funded by UCL through the UCL Overseas Research Scholarships (ORS) and Graduate Research Scholarships (GRS).

Yicheng Luo
London, September 2024

Impact Statement

This thesis presents novel approaches for leveraging imperfect data in reinforcement learning (RL) to make RL more practical and accessible for addressing complex real-world challenges. By developing methods that utilize sub-optimal experiences, incomplete datasets, and unstructured data, the findings of this research have the potential to substantially reduce the reliance of RL systems on expensive, high-quality datasets, thereby broadening their applicability.

The research advances data-efficient methods in reinforcement learning, which are particularly valuable in domains where data collection is either prohibitively costly or hazardous, such as robotics and autonomous systems. It contributes to the evolution of RL methodologies, equipping them for broader applications in areas such as healthcare, transportation, and industrial automation. Moreover, the techniques, software, and datasets introduced in this thesis provide a foundation for future research, offering a robust platform for studying the challenges of reinforcement learning with imperfect data.

This research holds the potential to enhance commercial activities by facilitating the application of RL in industries characterized by the availability of large yet imperfect datasets. For instance, in industrial automation and robotics, the ability to train RL agents using sub-optimal or incomplete data can substantially reduce costs and development time for autonomous systems. Additionally, sectors such as healthcare could benefit from RL agents trained on imperfect clinical data, leading to improvements in decision support systems and ultimately enhancing patient care, even in scenarios where data may be incomplete or noisy. The practical applications of these advancements are both promising and wide-reaching.

From a public policy perspective, the methodologies developed in this thesis

could inform the design of RL-driven systems aimed at optimizing resource allocation, infrastructure management, and environmental conservation. Integrating unstructured datasets—such as textual reports or satellite imagery—can further support public health initiatives and disaster management efforts, enabling governmental and organizational bodies to make more informed decisions in the presence of sub-optimal or incomplete data.

As reinforcement learning’s role in addressing complex, real-world problems continues to expand, the approaches proposed in this thesis offer the potential to make RL systems more accessible and further reduce their dependence on costly, high-quality data. This research paves the way for the broader adoption of intelligent and adaptive systems across industries, governments, and communities, thereby contributing to meaningful societal and economic advancements.

Contents

1	Introduction	17
1.1	Reinforcement learning	17
1.2	Imperfect data and where to find them	17
1.3	Contributions	21
1.4	Thesis organization	22
2	Background	24
2.1	Reinforcement learning	24
2.1.1	Markov decision process	24
2.1.2	Policy and value functions	25
2.1.3	Online reinforcement learning	26
2.1.4	Offline reinforcement learning	28
2.2	Imitation learning	31
2.2.1	Behavior cloning	32
2.2.2	Inverse reinforcement learning	33
3	Finetuning from offline reinforcement learning	36
3.1	Overview	36
3.2	Preliminaries	37
3.3	Experiments	40
3.3.1	Experimental set-up	40
3.3.2	Effect of online algorithms	41
3.3.3	Effect of offline data during finetuning	44
3.3.4	Conservative policy improvement in TD3	45
3.3.5	Comparative evaluation	49

3.3.6	Summary of empirical observations	51
3.4	Discussion	53
3.5	Conclusion	56
4	Optimal transport for offline imitation learning	57
4.1	Overview	57
4.2	Preliminaries	58
4.3	Optimal transport reward labeling	59
4.4	Experiments	65
4.4.1	Experimental setup	67
4.4.2	Experimental results	67
4.5	Discussion	76
4.6	Conclusion	77
5	Bridging policy learning and language modeling	78
5.1	Overview	78
5.2	Dataset	81
5.2.1	Game dataset	82
5.2.2	Language dataset	84
5.2.3	Mixed game-language dataset	85
5.2.4	Instruction-tuning and conversation dataset	85
5.3	Models	86
5.3.1	ChessCLIP	86
5.3.2	ChessGPT	88
5.4	Evaluation	88
5.4.1	Chess modeling ability	89
5.4.2	Value judgement ability	93
5.4.3	Policy evaluation	96
5.4.4	Qualitative results	100
5.5	Discussion	100
5.6	Conclusion	102
6	Conclusions	103

A	Supplements for Chapter 2	128
A.1	Implementation Details	128
A.2	Other choices of RL algorithms	129
A.3	Comparison with RLPD	130
A.4	Comparison to prioritized balanced replay	131
B	Supplements for Chapter 3	137
B.1	Hyperparameters	137
C	ChessGPT dataset supplements	139
C.1	Chess notations	139
C.1.1	Universal Chess Interface (UCI)	139
C.1.2	Standard Algebraic Notation (SAN)	139
C.1.3	Portable Game Notation (PGN)	140
C.1.4	Forsyth-Edwards Notation (FEN)	140
C.2	Dataset details	141
C.2.1	Dataset statistics and metrics	141
C.2.2	Preprocessing	141
C.2.3	Online chess websites	144
C.2.4	Online Chess forums	146
C.2.5	Annotated PGN	148
C.2.6	Existing datasets	150
C.2.7	YouTube transcripts dataset	152
C.2.8	Lichess dataset	154
C.2.9	Pro-player dataset	156
C.2.10	Chess books	156
C.2.11	CCRL	158
C.2.12	Chess puzzles	158
C.2.13	Chess modeling dataset	159
C.2.14	Instruction-tuning dataset	160
C.3	Implementation and evaluation Details	160
C.3.1	Implmenetation details	161
C.4	Evaluation details	165
C.5	Qualitative analysis	166

C.5.1	Experimental set-up	166
C.5.2	General knowledge	166
C.5.3	Chess modeling	170
C.6	ChessCLIP visualization	175

List of Figures

1.1	Examples of imperfect data in Minecraft	20
2.1	Overview of the reinforcement learning	24
2.2	Difference between on-policy and off-policy RL	28
2.3	Illustration of offline RL	29
3.1	Workflow for finetuning from offline RL	37
3.2	Performance comparison of different finetuning RL algorithms .	42
3.3	Effect of using offline datasets for online finetuning	45
3.4	Results of TD3, TD3-BC and TD3-C for finetuning	47
3.5	Effect of varying ϵ in TD3-C	48
3.6	Effect of delaying policy optimization	48
4.1	Illustration of offline imitation learning and OTR	59
4.2	Illustration of the computations performed by OTR	61
4.3	Overview of benchmark tasks	66
4.4	Qualitative differences between OTR, ORIL, and UDS	71
4.5	Illustration of rewards on AntMaze	72
5.1	Overview of the ChessGPT benchmark	80
5.2	Replay example in Portable Game Notation (PGN) format . . .	83
5.3	Example in chess state tracking	90
A.1	Performance of IQL using fixed ratio replay	131
A.2	Effect of data mixing ratio in RLPD	134
A.3	Comparison between RLPD and prioritized balanced replay . .	134
A.4	Balanced replay without offline pretraining	135

C.1	ChessGPT-Base attention visualization	165
C.2	Similarity matrix of different chess opening PGN and text	175

List of Tables

1.1	Summary of contributions	22
3.1	Performance comparison between ODT, IQL, and TD3	50
4.1	D4RL performance comparison	69
4.2	Aggregate performances of reward labeling algorithms	70
4.3	Performance of OTR+IQL on AntMaze	70
4.4	Performance of OTR+IQL on Adroit	71
4.5	OTR+TD3-BC results on MuJoCo	73
4.6	OTR with uniform transport plan.	74
4.7	Comparison between OTR and PWIL	75
4.8	Effect of α and β in the squashing function	75
5.1	BIG-bench state tracking in chess	91
5.2	UCI to FEN test	92
5.3	State value multi-choice	94
5.4	Chess annotation multi-choice	94
5.5	Opening2PGN and PGN2Opening	96
5.6	Checkmate in one	97
5.7	Elo Rating 1700-2000	98
5.8	ChessGPT-Base in different Elo rating results	99
A.1	TD3 and TD3-BC hyper-parameters	129
A.2	TD3-C hyper-parameters	129
A.3	Effect of online data buffer for CQL	130
A.4	Comparison with RLPD	132

A.5	Comparison with prioritized balanced replay	133
B.1	OTR hyperparameters for D4RL Locomotion.	138
B.2	OTR hyperparameters for D4RL Antmaze	138
B.3	OTR hyperparameters for D4RL Adroit	138
C.1	Dataset statistics	142
C.2	Metrics for the language dataset.	142
C.3	Training hyperparameters for chess board to FEN	153
C.4	Validation accuracy of each chess piece	154
C.5	ChessCLIP Training Hyperparameters	162
C.6	ChessGPT-Base Training Hyperparameters	163
C.7	ChessGPT-Chat Training Hyperparameters	163
C.8	General policy evaluation in Black	164
C.9	Hyperparameters for qualitative analysis.	166

Chapter 1

Introduction

1.1 Reinforcement learning

Reinforcement learning (RL) [37] is a branch of machine learning where agents learn to make decisions by interacting with an environment to maximize cumulative rewards. Unlike supervised learning, which relies on a fixed dataset of labeled examples, RL involves an agent learning from the consequences of its actions in a dynamic and often uncertain environment. The agent receives feedback in the form of rewards or penalties and uses this feedback to improve its decision-making policy over time. This trial-and-error approach allows RL to tackle complex problems such as robotics and game-playing, where optimal solutions are not easily predefined.

The general framework of RL has enabled significant impacts in various fields. For instance, RL algorithms have outperformed human champions in games like Go [29] and chess [36], and revolutionized robotic control and automation [100, 101]. These successes underscore RL's potential to solve intricate real-world problems through intelligent and adaptive behavior.

1.2 Imperfect data and where to find them

Traditional RL emphasizes learning from scratch, where agents gather sub-optimal experiences as intermediate steps toward proficiency. While this ap-

proach provides enough generality, it suffers from high sample complexity. For example, training the AlphaGo agent required 4.9 million games, equating to 27 years of human experience at 500 games per day [29]. Each interaction involves computation and time, making this approach infeasible for many real-world applications.

Offline reinforcement learning [17, 47] emerges as a promising solution to mitigate online sample complexity by utilizing pre-existing behavior datasets. Instead of relying solely on real-time interactions, offline RL leverages these datasets to pre-train agents, thus reducing the need for extensive online interactions. This approach accelerates training and enables learning from diverse and previously inaccessible data sources.

Despite these advances, today’s systems rely significantly on high-quality experience to build capable agents. Collecting these data is more expensive than in other branches of machine learning, such as computer vision [15] or natural language processing [44], since it often requires interactions with an embodied environment constrained by physics and time.

Fortunately, abundant imperfect data exist in various forms, which may lack essential information or exhibit sub-optimality yet still hold potential value for learning. Leveraging these imperfect data for building capable artificial agents remains an open research question. This thesis considers three categories of imperfect data crucial for advancing RL.

Sub-optimal experience. Sub-optimal experience refers to data derived from actions or policies that do not necessarily lead to the best possible outcomes. In many real-world scenarios, optimal behavior is either unknown or impractical. For example, gathering data from a fully optimized policy in robotics may require costly trial-and-error processes that can be time-consuming or risky. Consequently, much of the available data comes from sub-optimal policies, reflecting less-than-perfect performance. Although these sub-optimal experiences might seem less valuable, they contain essential information about the environment’s dynamics and the potential consequences of different actions. While these experiences may not be sufficient for an artificial agent to reach proficiency for a given task, they may significantly reduce the additional data needed to reach

strong performance.

Datasets with missing features. Datasets with missing features present another form of imperfect data that challenges traditional RL approaches. In many real-world applications, especially those involving complex and dynamic environments, capturing all relevant variables during data collection is impossible. For instance, sensor failures, communication lags, or limitations in data recording systems can result in incomplete datasets. These missing features may include crucial information such as the actions taken by an agent, the rewards received, or even crucial state variables that describe the environment. Incorporating these incomplete datasets into the current RL pipeline requires sophisticated techniques that can infer missing information or compensate for the lack of certain features, ensuring that the agent can still develop a robust policy despite these gaps in the data.

Unstructured datasets. Unstructured data, such as text, images, or video, offers another underutilized resource for RL, particularly in the context of behavior learning. Unlike structured RL datasets, which consist of well-defined observations and actions, unstructured data is typically more challenging to process and analyze. However, these data sources can provide rich contextual information about an agent's environment and behavior. For example, videos of humans performing tasks can serve as a valuable source of behavioral data, offering insights into strategies and actions that an RL agent might not otherwise learn from traditional datasets. The key challenge here is to develop methods to extract meaningful features and patterns from unstructured data, enabling RL agents to learn complex behaviors by observing and imitating human or other agents' actions recorded in such formats.

To see what these data look like in a practical scenario. Consider the problem of building an artificial agent capable of playing Minecraft, as illustrated in fig. 1.1. In this setting, we may have demonstrations collected using a data collection scheme such as in Guss et al. [39]. These data may consist of plays from humans with different proficiency levels; thus, they do not necessarily demonstrate the optimal behavior for completing a task, such as collecting a dia-

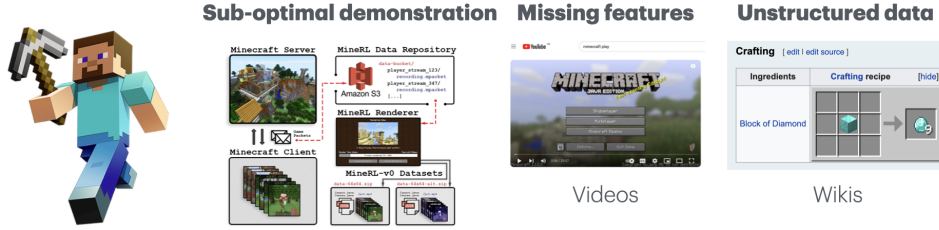


Figure 1.1: Examples of imperfect data in Minecraft. Consider the problem of building an artificial agent that can play the game of Minecraft at the level of a human player. Instead of learning to acquire skills such as building a diamond axe, which may be prohibitively challenging if the agent learns from scratch, we can consider providing the agent with access to external experiences to facilitate learning. Besides logged datasets of (sub-optimal) experiences, we may additionally consider video demonstrations such as those found on online websites such as YouTube. These datasets lack essential features such as actions and rewards for reinforcement learning. In addition, we may also consider knowledge databases such as wikis, which are often unstructured.

mond. These datasets are reward-free; that is, they do not have reward annotation, which prevents the use of offline RL. In addition to these demonstrations, we may have unstructured datasets, such as wikis, which contain helpful knowledge about playing Minecraft. Utilizing all of these data present in various forms is the main topic of the thesis.

While high-quality data remains indispensable for building effective artificial agents, the ability to learn from imperfect data – sub-optimal experiences, datasets with missing features, and unstructured data – presents a significant opportunity for advancing RL. Additional sub-optimal data may allow us to address the poor generalization performance due to limited coverage from high-quality demonstrations. Relying on less structured datasets with missing information would allow us to scale RL algorithms more economically to deal with increasingly complex real-world problems. Addressing these challenges requires novel algorithms and frameworks integrating diverse and imperfect data sources.

1.3 Contributions

This thesis explores the possibilities of leveraging imperfect data to reduce the dependence on costly, high-quality datasets, making RL more accessible and practical for a broader range of real-world applications. It encompasses research and ideas from the following papers published during this study:

1. Y. Luo, A. Filieri, and Y. Zhou. Symbolic parallel adaptive importance sampling for probabilistic program analysis. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021
2. Y. Luo, J. Kay, E. Grefenstette, and M. P. Deisenroth. Finetuning from offline reinforcement learning: challenges, trade-offs and practical solutions. In *The 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2022
3. Y. Luo, Z. Jiang, S. Cohen, E. Grefenstette, and M. P. Deisenroth. Optimal transport for offline imitation learning. In *The 11th International Conference on Learning Representations*, 2023
4. X. Feng, Y. Luo, Z. Wang, H. Tang, M. Yang, K. Shao, D. Mguni, Y. Du, and J. Wang. ChessGPT: bridging policy learning and language modeling. In *Advances in Neural Information Processing Systems*, 2023
5. S. Zwane, D. Hadjivelichkov, Y. Luo, Y. Bekiroglu, D. Kanoulas, and M. P. Deisenroth. Safe trajectory sampling in model-based reinforcement learning. In *19th IEEE International Conference on Automation Science and Engineering*, 2023
6. Z. Jiang, Y. Xu, N. Wagener, Y. Luo, M. Janner, E. Grefenstette, T. Rocktäschel, and Y. Tian. H-GAP: humanoid control with a generalist planner. In *The 12th International Conference on Learning Representations*, 2024
7. S. N. T. Zwane, D. G. Cheney, C. C. Johnson, Y. Luo, Y. Bekiroglu, M. Killpack, and M. P. Deisenroth. Learning dynamic tasks on a large-scale soft robot in a handful of trials. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2024

1.4 Thesis organization

In the following chapters, we focus on techniques and present methods for exploring the possibility of incorporating imperfect data in reinforcement learning. In chapter 2, we discuss the background of reinforcement learning and imitation learning, introducing the framework and notation used in subsequent chapters.

The subsequent chapters use increasingly imperfect data to build capable agents, evolving from narrow and directed demonstrations toward achieving specific goals to diverse and mixed experiences collected from various sources. The dataset structure transitions from complete, where all information needed for RL is available, to datasets missing key ingredients (e.g., rewards, actions) or presented in free structure (e.g., natural language). Table 1.1 summarizes the chapters’ progress towards using more imperfect data.

	Quality	Structured	Publication
Chapter 3	sub-optimal	yes	Luo et al. [86]
Chapter 4	diverse	yes	Luo et al. [104]
Chapter 5	diverse	no	Feng et al. [99]

Table 1.1: Summary of contributions. The following chapters discuss leveraging imperfect data with varying qualities and structures for reinforcement learning.

In chapter 3, we address the problem of finetuning RL agents pre-trained on sub-optimal reward-annotated behavior datasets. We empirically analyze several design decisions in the continual learning of an RL agent first pre-trained offline and summarize the trade-offs, providing recommendations about different design decisions. This scenario represents cases where behavior data is readily available for offline RL but of sub-optimal quality, making pure offline approaches insufficient. This chapter presents a data-efficient strategy for improving performance with additional online data.

In chapter 4, we explore the problem of automatically learning a reward function to label offline datasets, enabling subsequent consumption by an offline RL algorithm. This scenario deals with diverse and mixed-quality demonstrations that lack reward information.

In chapter 5, we investigate learning artificial agents using offline data not

intentionally created for learning behaviors. We propose a benchmark and dataset based on chess, exploring the possibility of learning models capable of decision-making and functioning as general assistants. This chapter exemplifies scenarios where valuable data for decision-making lacks an episodic structure but knowledge for decision-making exists implicitly in text.

Chapter 2

Background

Reinforcement learning (RL) is a powerful paradigm for training agents to make sequences of decisions by interacting with an environment to maximize cumulative reward. This chapter provides the necessary background to understand the computational frameworks used in later chapters. We introduce the fundamental concepts of online and offline reinforcement learning and explore the related field of imitation learning. This chapter also introduces the notation used for subsequent chapters.

2.1 Reinforcement learning

2.1.1 Markov decision process

Reinforcement learning (RL) is the study of a learning *agent* and an *environment*. The interaction between the two components is illustrated in fig. 2.1: the agent

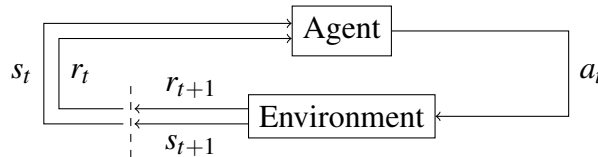


Figure 2.1: Overview of reinforcement learning. An *agent* receives states s_t and reward r_t feedback from an *environment*. The agent selects an action a_t , observes the next state s_{t+1} , and rewards r_{t+1} . The agent uses the experience of interacting with the environment to adapt its behavior.

observes the state of the environment and selects actions that influence the dynamics of the environment in discrete time. The environment provides feedback through rewards, which the agent uses to learn a better behavior adaptively.

The interaction between the agent and environment can be characterized as a *Markov Decision Process (MDP)*. A Markov decision process \mathcal{M} is a tuple $(\mathcal{S}, \mathcal{A}, p, p_0, r)$ where

- \mathcal{S} is the state space,
- \mathcal{A} is the action space,
- $p_0 : \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$ is the initial state distribution,
- $p : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_{\geq 0}$ is a stochastic transition model, and
- $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a reward function.

An MDP formulation assumes that the agent fully observes the state of the environment. In scenarios where the agent only receives partial observations, the interaction is modeled as a Partially Observable Markov Decision Process (POMDP) [6]. In this thesis, we assume that the state is fully-observed.

2.1.2 Policy and value functions

The agent's behavior is defined by a policy π , which is mapping from state to a probability distribution over actions given states. We use $\pi(a|s)$ to denote the probability of selecting the action a under state s . In the thesis, we primarily work with parameterized policies and use π_θ to denote a policy parameterized by θ .

The agent interacts with the environment with its policy and produces trajectories. A *trajectory* is a sequence of states and actions of length H defined as

$$\tau = s_0, a_0, s_1, a_1, \dots, s_H, a_H.$$

Given a policy π used to interact with an MDP \mathcal{M} , we define the *trajectory distribution* as

$$p_\pi(\tau) = p_0(s_0) \prod_{t=0}^{H-1} p(s_{t+1}|s_t, a_t) \pi(a_t|s_t).$$

The goal of reinforcement learning is to find a policy that maximizes the expected cumulative discounted return, defined as

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^H \gamma^t r(s_t, a_t) \right], \quad (2.1)$$

where $\gamma \in [0, 1)$ is the *discount factor* that trades off the goal of maximizing immediate or future rewards. We denote the *optimal policy* that maximizes the objective in eq. (2.1) as π^* . Practical reinforcement learning involves the design and implementation of an *RL algorithm*, which are computational processes that aim at solving the optimization problem of eq. (2.1) efficiently.

The discussion of reinforcement learning algorithms almost always involves estimating the value function – functions of the states to how good it is for the agent to be in that given state. Specifically, the *state value function*

$$V^\pi(s_t) = \mathbb{E}_{\tau \sim p_\pi(\cdot | s_t)} \left[\sum_{t'=t}^H \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]$$

is the expected return for a policy π starting from state s_t . Similarly, the *action value function*

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau \sim p_\pi(\cdot | s_t)} \left[\sum_{t'=t}^H \gamma^{t'-t} r(s_{t'}, a_{t'}) \right]$$

is the value of a policy that selects action a_t starting from state s_t . The action value function describes how good it is for the agent to be in a given state *and* selecting a given action. The value function summarizes a policy's short-term and long-term gains and is one of the most important concepts used in many reinforcement learning algorithms.

2.1.3 Online reinforcement learning

Online reinforcement learning studies the problem of learning an optimal policy through repeated interactions with a given environment.

Model-based and model-free RL. When the dynamic of the MDP is known, classical methods such as Dynamic Programming [4] can be used to learn an optimal policy. Generally, most RL algorithms assume that the environment dynamics are unknown. A *model-free* RL algorithm learns an optimal policy without explicitly trying to model the dynamics of the environment. A variety of model-free methods, such as Policy Gradient [37], Q-learning [3] and Actor-Critic [7] can be used for learning an optimal policy for a given MDP. A *model-based* RL algorithm first learns an approximate environment model and then performs policy learning with the help of a learned model [16]. Model-based can allow more sample-efficient learning but requires principled treatment of modeling errors such that these errors do not adversely influence efficient policy learning.

On-policy and off-policy RL. A distinction should be made between online RL algorithms that are *on-policy* and those that are *off-policy*. An RL algorithm is *on-policy* if it uses the experience generated by the *current* policy to find an improved policy. An RL algorithm is *off-policy* if it also uses experience besides those from the current policy for updating its parameters. For example, the algorithm may store experiences collected earlier in learning and repeatedly replay those experiences to update its parameters.

The difference between an on-policy and off-policy RL algorithm is illustrated in fig. 2.2. The distinction between online and offline RL algorithms will become helpful as we discuss offline RL algorithms in section 2.1.4.

Off-policy algorithms often exhibit better sample efficiency than on-policy algorithms and are more widely used in domains where data collection is more expensive (e.g., robotics). An off-policy algorithm can also be combined with a model-based learning approach to achieve further sample-efficiency gains [118].

Challenges of online RL

Online RL faces several challenges that limit its application and effectiveness in real-world scenarios [61]. One of the biggest challenges of interest is *sample efficiency*. Online RL often requires a large number of interactions with the environment to learn effective policies, which can be impractical in real-world

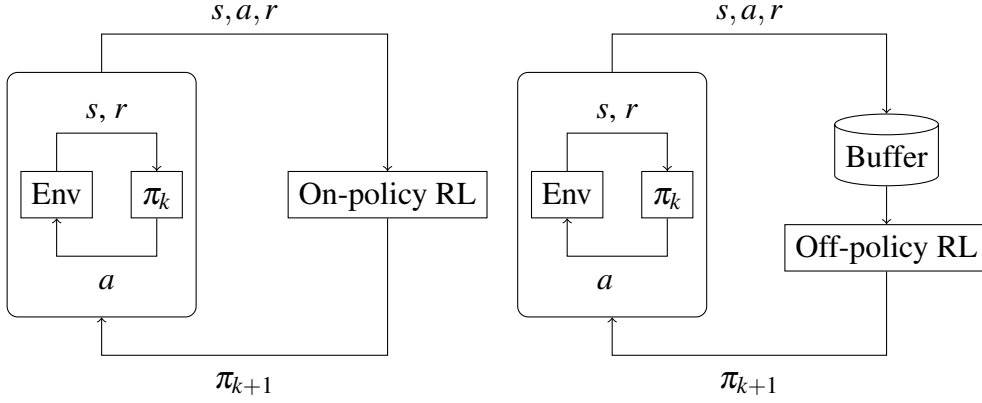


Figure 2.2: Difference between on-policy and off-policy RL. RL algorithms use an iterative process to interact with an environment and find a sequence of improved policies $\pi_0, \pi_1, \dots, \pi_k, \pi_{k+1}$. At an iteration k , an on-policy RL algorithm (left) uses the current policy π_k to collect experience (states s , actions a and rewards r) from the environment and uses the experience from π_k to find a better policy π_{k+1} . An off-policy RL algorithm (right) uses the current policy π_k to collect experience and store it in a buffer, which may consist of interactions from all previous policies π_0, \dots, π_{k-1} and uses all of the experiences to find an improved policy.

scenarios where interactions are costly or time-consuming. This issue is further exacerbated by the use of poor exploration strategies, where the agent needs to first discover interesting and good behaviors before learning to perform well consistently. Sophisticated algorithms have emerged in recent years to tackle the poor sample efficiency problem. For example, using model-based approaches [16, 118] in conjunction with learning through more off-policy data [34, 95]. As a result, these advances have enabled real-time learning of impressive behaviors in some applications [103, 118, 16]. Nevertheless, challenges remain for RL methods to be applied more broadly to more domains.

2.1.4 Offline reinforcement learning

Offline reinforcement learning studies the problem of learning policies from offline behavior datasets. The standard formulation of offline/batch reinforcement learning considers the problem of learning optimal policies directly from fixed behavior datasets [17, 47], which are datasets of behaviors consisting of state, action, reward tuples, or episodes of trajectories. This learning paradigm is

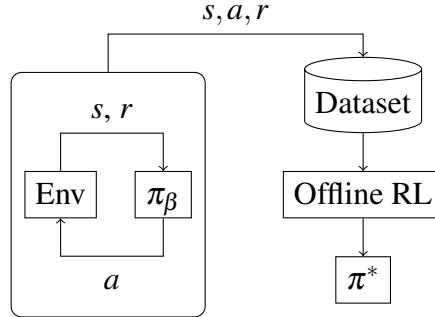


Figure 2.3: Illustration of offline RL. An offline RL algorithm uses a dataset of experience collected by some behavior policy π_β to find an optimal policy π^* for a given environment. Unlike online RL, offline RL does not require online interactions.

illustrated in fig. 2.3.

The online formulation of the RL algorithms we have considered in the previous section remains one of the biggest obstacles to their adoption. The online reinforcement learning paradigm is inadequate due to

- *Expensive data collection.* In some settings, online data collection is not feasible. Data collection can be expensive in domains such as robotics or healthcare. Data collection can be dangerous, for example, in nuclear fusion or autonomous driving.
- *Data reuse.* In some cases, it might be preferable to reuse previously collected data. For some high-dimensional problems, such as learning RL agents from pixel observations or hard-exploration problems, the policy may require a large amount of experience to learn to generalize. In such scenarios, it is more effective if the agent can reuse previously collected experience. In addition, sometimes, we might be interested in reusing experience generated for a different task. For example, suppose we have learned a policy to move a robot arm to grasp a single Lego block. We are now interested in learning a policy that can stack a set of Lego blocks. In this case, we may wish to reuse the data generated for grasping a single block to teach the agent the low-level skills needed to stack the blocks together.

Offline reinforcement learning, while discussed separately from online re-

inforcement learning, shares a close relationship with online RL, particularly with off-policy methods. Offline RL can be conceptualized as an extreme case of off-policy learning, where the agent learns from data generated by different behavior policies without any access to environmental interaction. Earlier literature frequently used the term “batch reinforcement learning” to describe this scenario, in which the agent learns exclusively from a fixed, predetermined set of experiences [17].

Challenges of offline RL

Despite the promise of solving online RL’s expensive data collection issue, offline RL faces several challenges.

Extrapolation error and distribution shift. In principle, offline RL is an off-policy learning problem since it involves learning a policy from data generated by a *different behavior policy*. Recent work has identified extrapolation error as a major challenge for offline deep reinforcement learning [69]. This error arises from the *distribution shifts* between the policy used for creating the dataset and the offline policy that we are learning. When data is missing from the offline dataset or when the function approximator has errors, applying standard online off-policy algorithms to offline datasets can result in significant extrapolation errors [38]. Ostrovski et al. [69] demonstrate that this issue is particularly severe with non-linear function approximation, where erroneous extrapolation can affect not just completely unseen data but also rarely seen state-action pairs, and can persist even with infinite training samples.

The problem is exacerbated by the absence of what Ostrovski et al. [69] call the “corrective feedback loop” — in online RL, value over-estimation is naturally corrected through exploitation during environment interaction, while under-estimation is addressed through exploration. This corrective mechanism is critically missing in the offline setting, leading to potentially severe under-performance of the learned policy.

Dataset quality. The performance of offline RL algorithms heavily depends on the quality and coverage of the offline dataset. Ostrovski et al. [69] demonstrate

through careful empirical analysis that the stochasticity and diversity of the behavior policy generating the dataset significantly impacts learning performance. Even when distribution shifts have been minimized through algorithmic advances, a performance gap may still exist between the offline optimal policy and the optimal policy for a given task due to dataset limitations. Their results suggest that this gap cannot be closed purely through algorithmic improvements — if relevant parts of the state-action space are not well represented in the offline dataset, the limitation is fundamental rather than algorithmic [78]. Ostrovski et al. [69] also show that even small amounts (10–20%) of additional self-generated data can substantially improve performance, highlighting the inherent limitations of pure offline learning. This finding also suggests that in practice a hybrid approach involving a combination of offline and online learning may provide promising improvements sample efficiency. This is closely related to the idea of *growing batch reinforcement learning* [17], where the set of sample experience may extend over time. We explore this issue in chapter 3.

2.2 Imitation learning

Imitation learning (IL) is another paradigm for creating artificial agents. Instead of relying on a reward function, imitation Learning aims at learning policies that can imitate the behavior of a demonstrator.

Addressing the challenges in reward specification. The RL formulation for creating artificial agents suffers from some drawbacks. In particular, the RL problem requires defining a reward function that accurately represents the performance objective.

Defining the reward function may be technically challenging. For some problems, it is difficult to determine a good reward function that aligns well with the intended objective. Defining a good reward function requires expertise and tuning. For example, the following reward function

$$r_t = r_t^{\text{prog}} + r_t^{\text{perc}} + r_t^{\text{cmd}} - r_t^{\text{crash}}$$

is used by Kaufmann et al. [101] to train a policy to reach champion-level performance in drone racing. The reward function consists of

$$\begin{aligned} r_t^{\text{prog}} &= \lambda_1 (d_{t-1}^{\text{Gate}} - d_t^{\text{Gate}}), \\ r_t^{\text{perc}} &= \lambda_2 \exp(\lambda_3 \cdot \delta_{\text{cam}}^4), \\ r_t^{\text{cmd}} &= \lambda_4 \mathbf{a}_t^\omega + \lambda_5 \|\mathbf{a}_t - \mathbf{a}_{t-1}\|^2, \\ r_t^{\text{cmd}} &= \begin{cases} 5.0, & \text{if } p_z < 0 \text{ or in collision with gate} \\ 0, & \text{otherwise} \end{cases} \end{aligned}$$

where $\lambda_1, \dots, \lambda_5$ are additional hyper-parameters for the reward components. Note that the reward function involves four carefully chosen components. In addition, it also includes five hyper-parameters that require tuning.

From a practical point of view, sometimes the most natural definition for a given task is a sparse reward function in which the agent receives a positive reward for succeeding in the task or zero otherwise. While a sparse reward function might be easy for a human to specify, it may not provide enough learning signal for an agent to learn to solve the task successfully.

2.2.1 Behavior cloning

Behavior Cloning (BC) [2] learns a policy by learning mapping from states to actions using the trajectories demonstrated by an expert. Concretely, consider the example of *An Autonomous Land Vehicle in a Neural Network* (ALVINN) [2] where the goal is to learn a policy for driving an autonomous landing vehicle. Pomerleau [2] curated a dataset $\{o_i, a_i\} \in \mathcal{D}$ of observations and actions where the observations o_i are road images and actions a_i are turning curvatures. Pomerleau [2] trained a policy π parameterized with a fully-connected neural network with the loss function

$$J(\pi) = \sum_{o_i, a_i \in \mathcal{D}} L(\pi(o_i), a_i), \quad (2.2)$$

where L is a cost function that computes the discrepancy between the actions generated by the training policy π and the expert actions.

Equation (2.2) appeals to many practitioners due to its simplicity. In addition, the formulation of BC as a supervised learning problem avoids the common challenge of non-stationarity in RL.

Challenges of BC

Sample-efficiency. Despite the conceptual simplicity of solving a supervised learning problem, BC presents many challenges for practical adoption. For BC to work well in various real-world scenarios, the training dataset should provide sufficient coverage of the intended behaviors. This issue may be significantly pronounced when neural networks are used for learning behavior policies due to poor generalization and over-fitting when data is limited. As a result, BC only tends to succeed with a large amount of high-quality data.

2.2.2 Inverse reinforcement learning

The problem of learning from demonstrations can be approached from two perspectives: directly learning the policy (behavior cloning) or inferring the underlying reward function that explains the demonstrated behavior. This latter approach is known as *inverse reinforcement learning (IRL)*, and was formally introduced by Ng and Russell [9]. The key insight is that the reward function is often the most succinct, robust, and transferable representation of a task. Instead of directly learning to predict the actions of an expert given a particular state, IRL learns a reward function that explains the expert's actions [9]. The learned reward function may then be used to train a policy using an RL algorithm.

Building on theoretical work in reward shaping [8], which showed how different reward formulations can lead to the same optimal policy while significantly affecting learning efficiency, IRL offers the potential to find reward functions that are more well-suited for learning. While manually specified reward functions might be sparse or difficult to optimize, the reward functions learned through IRL can naturally encode the structure present in expert demonstrations, potentially leading to more efficient learning of the desired behavior.

Distribution matching perspective for IRL. Imitation learning may be approached probabilistically as a *distribution matching* problem [24, 45]. Let d^{exp} denote the stationary distribution of the expert policy π_{exp} . Suppose that we are interested in learning a policy π that imitates the expert π_{exp} . The policy has a corresponding stationary distribution d^π . The goal of imitation learning is to

$$\min_{\pi} D(d^\pi || d^{\text{exp}}), \quad (2.3)$$

where D is some measure of discrepancy between two probability distributions. The distribution matching perspective recasts the problem of imitation as finding a policy whose stationary distribution matches that from the expert policy.

In the case the KL-divergence is used as the measure of discrepancy, then we have

$$-\max_{\pi} D_{\text{KL}}(d^\pi || d^{\text{exp}}) = \max_{\pi} \mathbb{E}_{(s,a) \sim d^\pi} \left[\log \frac{d^{\text{exp}}(s,a)}{d^\pi(s,a)} \right]. \quad (2.4)$$

By learning a density ratio estimator $\tilde{r}(s,a) \approx \log \frac{d^{\text{exp}}(s,a)}{d^\pi(s,a)}$, Equation (2.4) is equivalent to an RL problem using \tilde{r} as the reward function. This connection underpins the development of the Generative Adversarial Imitation Learning (GAIL) [24] algorithm.

Compared to BC, IRL is advantageous as the learned reward function may capture the underlying reasons for the expert behavior. As a result, this would allow the agent to learn from experiences absent from the demonstration dataset. Consequently, IRL methods provide better demonstration sample efficiency than BC, requiring only a dozen demonstrations to learn a robust imitating policy using algorithms such as the Generative Adversarial Imitation Learning (GAIL) [24]. Some IRL methods are also capable of distribution matching on the stationary state distribution. This property enables algorithms to learn from observation-only datasets.

Challenges of IRL

However, these IRL methods are not *training-sample-efficient*, as learning the inverse reward function requires the collection of a large amount of online samples. In GAIL, learning the reward function requires collecting millions

of samples via an on-policy algorithms such as PPO [28] despite using only a few episodes of demonstration data. Nevertheless, progress has been made to improve the sample efficiency, for example, using an off-policy algorithm [45].

Besides training inefficiency, algorithms such as GAIL follow a training paradigm that is similar to Generative Adversarial Networks (GANs) [21], by formulating the problem of IRL as a minimax optimization problem to learn a discriminator that implicitly minimizes an f-divergence. The minimax formulation presents a challenging optimization problem, and Adversarial Imitation Learning (AIL) methods such as GAIL usually require careful tuning of hyperparameters [68].

More recently, IRL methods based on Optimal Transport (OT) have demonstrated success as an alternative method for IRL compared to AIL approaches. Unlike AIL approaches, OT methods minimize the Wasserstein distance between the expert's and the agent's state-action distributions. Building on this formulation, Xiao et al. [42] proposes to minimize the Wasserstein distance via its dual formulation, which may lead to potential optimization issues. More recently, Dadashi et al. [60] introduces PWIL, which instead minimizes the Wasserstein distance via its primal formulation, avoiding the potential optimization issues in the dual formulation. Along this line of work, Cohen et al. [59] suggests improvements to the primal Wasserstein formulation and demonstrates improved empirical results in both sample efficiency and final performance. Still, these approaches require many online samples to learn good policies. While progress has been made to improve the sample efficiency of these approaches [45], imitation learning without any online interaction remains an active research area.

Chapter 3

Finetuning from offline reinforcement learning

3.1 Overview

In this chapter, we explore methods to enhance offline pre-trained policies through a limited amount of additional online interactions.

The datasets we consider in this chapter are imperfect because they do not have high quality or sufficient coverage to allow pure offline methods to learn optimal policies. Nevertheless, they contain the necessary information (states, actions, and rewards) for offline learning. These offline datasets may originate from previous RL training runs or other data sources, such as data generated by a low-quality policy.

Despite their lower quality, integrating these additional datasets offers new opportunities to leverage preexisting information, potentially accelerating online RL. However, it can also introduce challenges that necessitate careful consideration.

We conduct an empirical evaluation to gain deeper insights into integrating offline and online RL. Building on existing offline and online RL algorithms, we demonstrate that the optimal combination of these approaches is more nuanced than it may initially appear. Our findings indicate that the best design choices depend on factors such as dataset composition, sample efficiency, and learning stability, often requiring practitioners to navigate their trade-offs.

This chapter presents our observations and recommendations, providing guidance on effectively merging offline and online RL to improve data efficiency.

3.2 Preliminaries

We consider a specific setting of combining offline and online RL where we first perform offline RL to learn an optimal policy using only offline data and continue learning with online RL. This approach is illustrated in fig. 3.1.

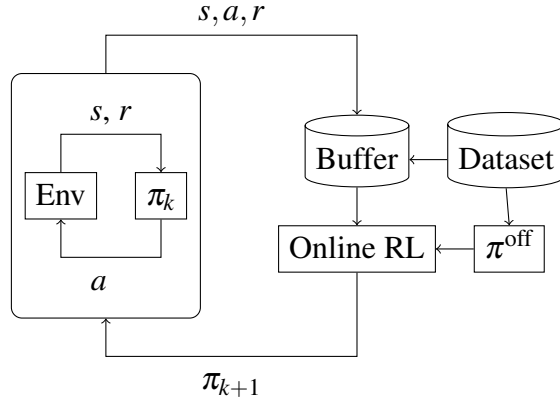


Figure 3.1: Workflow for finetuning from offline RL. We consider the problem of first learning a policy π^{off} from an offline behavior dataset and improving the offline policy with online RL.

This approach is analogous to *finetuning*, a paradigm widely adopted in machine learning where a model is first pretrained on a large dataset before being refined on a smaller, task-specific dataset. The concept has also gained more traction in recent years with the emergence of large language models (LLMs) and Reinforcement Learning from Human Feedback (RLHF) [89]. In RLHF, a supervised-trained LLM is further optimized using RL to better align with human preferences, demonstrating how pretrained models can be enhanced through subsequent RL-based optimization. While our setting shares core similarities with RLHF—both start with a model trained on a fixed dataset and improve it through interactive learning—a key distinction lies in our use of RL techniques throughout both phases rather than just during finetuning. These parallels suggest that our insights on combining offline and online RL may inform better practices

for applying RL to LLMs, a direction we explore in chapter 5.

Even with the restriction of explicitly splitting the hybrid pipeline into pre-training and finetuning stages, this recipe of combining offline RL and online RL is fairly general and leaves many open questions, such as

1. *What algorithms should we use for offline learning / online finetuning?*

Many offline RL algorithms can be directly applied to online learning.

However, are offline RL algorithms a good choice for online learning?

What happens if we use an algorithm designed for online RL?

2. *How should we leverage offline data during online finetuning?* The first question to ask is whether we should reuse the offline dataset at all during finetuning. If yes, what is the best way to incorporate offline data during online finetuning?

Actor-critic algorithms. To make the investigation tractable, we investigate these design choices in the context of actor-critic methods, a popular architecture for developing online and offline RL algorithms. The actor-critic architecture consists of two components: an *actor* learns to select actions given states and a *critic* that predicts the quality of actions selected by the actor.

DDPG and TD3. Among actor-critic algorithms, the *Deep Deterministic Policy Gradient* (DDPG) [26] is a popular off-policy, actor-critic algorithm used for learning optimal policies in continuous action spaces. DDPG learns an approximate state-action value function Q_ϕ and deterministic policy π_θ parameterized by neural networks by alternating between policy evaluation and improvement. During policy evaluation, we learn an approximate action value function Q_ϕ by minimizing the Bellman error

$$\phi^* = \arg \min_{\phi} \mathbb{E}_{s,a,r,s' \sim \mathcal{B}} [(Q_\phi(s,a) - (r + \gamma Q_{\phi'}(s', \pi_{\theta'}(s'))))^2], \quad (3.1)$$

where $Q_{\phi'}$ and $\pi_{\theta'}$ are the target critic and policy networks used to stabilize Temporal-Difference learning with function approximation. The transitions $s,a,r,s' \sim \mathcal{B}$ are drawn from an online replay buffer similar to the set-up in DQN [22].

During policy improvement, the policy parameters θ are updated to maximize the current action value function

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{s \sim \mathcal{B}} [Q_{\phi}(s, \pi_{\theta}(s))] . \quad (3.2)$$

For exploration, DDPG adds a state-independent noise sampled from a multivariate Gaussian distribution or an Ornstein-Uhlenbeck process [26] to the deterministic policy. Fujimoto et al. [33] proposed the TD3 algorithm that improves upon the original DDPG formulation by

1. Taking the minimum value of a pair of critic networks to reduce overestimation.
2. Delaying policy updates to reduce per-update error.

These changes help reduce the function approximation error that leads to value overestimation and sub-optimal policies in the original DDPG paper.

TD3-BC. Actor-critic algorithms such as DDPG or TD3 suffer from distribution shifts when used for offline learning. To make these algorithms amenable to learning from fixed offline datasets, TD3-BC [64], for example, adds an additional behavior cloning (BC) loss

$$\theta^* = \arg \max_{\theta} \mathbb{E}_{s, a \sim \mathcal{B}} [\lambda Q_{\phi}(s, \pi_{\theta}(s)) + (a - \pi_{\theta}(s))^2] , \quad (3.3)$$

to the policy loss objective in eq. (3.2). The additional term $(a - \pi_{\theta}(s))^2$ encourages the policy output $\pi_{\theta}(s)$ to stay close to actions a available in the offline dataset and $\lambda \geq 0$ modulates the relative strength of the original policy optimization objective and the behavior cloning regularization.

In the next section, we conduct experiments using selections of offline and online algorithms and different ways of incorporating offline data during online learning. We compare results for different choices with multiple tasks and datasets, which allows us to draw conclusions that hold in different settings. We choose to focus on actor-critic algorithms such as TD3 and TD3-BC as they are the one of the most popular class of algorithms in use today. Furthermore, we run

experiments on continuous control environments which makes algorithms such as DQN [22] not directly applicable.

3.3 Experiments

In this section, we empirically study and analyze the challenges in performing online finetuning after pretraining with offline RL.

3.3.1 Experimental set-up

Tasks and dataset

Our analysis builds on top of MuJoCo [18] tasks in the D4RL benchmark suite [63]. We consider datasets from the Walker2D, HalfCheetah, Hopper, and Ant tasks. For each task, we perform finetuning given the corresponding medium and medium-replay datasets. The medium datasets consist of transitions collected by the evaluation policy of an early-stopped, sub-optimal agent. medium-replay datasets refer to the transitions stored in the replay buffer of an early-stopped agent. Both datasets contain transitions that enable the agent to acquire medium performance. The medium dataset includes only transitions from the learned policy, which, on average, have higher quality than the medium-replay datasets but lack diversity.

Algorithms

We use TD3 [33] and TD3-BC [64] as our offline and online RL algorithms representatives. For offline pretraining, we use TD3-BC because of its good empirical performance on the MuJoCo locomotion benchmarks and its simplicity. While many offline RL algorithms have been proposed based on introducing additional regularization during offline training [46], a close inspection of these algorithms based on our reimplementation experience and observations from Fujimoto and Gu [64] reveal that they often incorporate additional code-level optimization as well as significant hyper-parameter changes to achieve good performance. It is also easy to compare TD3-BC with its off-policy counterparts (TD3) for

finetuning performance since switching from TD3-BC to TD3 requires only removing the BC penalty, keeping all other hyper-parameters fixed.

During online finetuning, we load the weights for the neural networks obtained from offline training and use either TD3 or TD3-BC as the finetuning algorithm. We use the same hyper-parameters as in the offline setting; it is possible that finetuning via online RL requires a significantly different set of hyper-parameters, but we leave this investigation as future work. In both cases, we pretrain the actor and the critic networks offline for 500K iterations and then train online for 200K environment steps. We perform one gradient step after every transition in the online environment during online finetuning. Therefore, one learner step after the pretraining stage corresponds to one step of online environment interaction. Note that the online training set-up differs from the online batch setting considered in [83, 67] but is closer to the standard online benchmark setting used in [33].

3.3.2 Effect of online algorithms

We start by analyzing how the choice of online algorithms impacts finetuning performance. For this experiment, we use either TD3 or TD3-BC as the online algorithms for finetuning.

Following the finetuning protocol in [94, 81, 67], we incorporate the offline data during finetuning by initializing the online replay buffer with transitions from the offline dataset. The transitions are sampled uniformly during both offline pretraining and online finetuning. We defer the discussion of alternative ways of using the offline dataset during finetuning to section 3.3.3.

Figure 3.2 shows the comparison between the different choices of online algorithms on evaluation performance. The results reveal a few interesting findings:

Offline RL algorithms improve more slowly than their online counterparts. Finetuning online with TD3-BC improves slowly compared to using TD3. This suggests that offline RL algorithms, which constrain the target policy to be close to the behavior policy, may improve more slowly than their standard off-policy counterparts. While we restrict our comparison to using TD3 and TD3-BC,

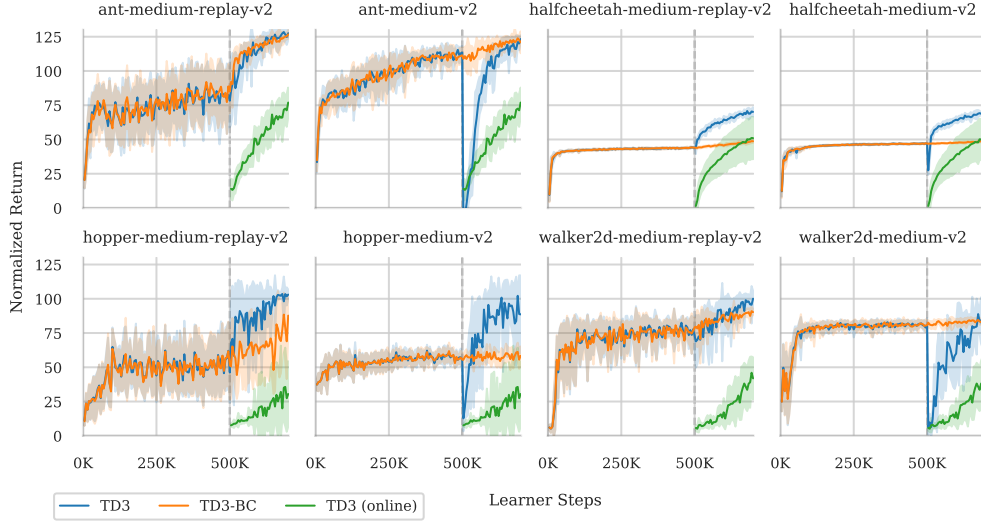


Figure 3.2: Comparison of TD3, TD3-BC for online finetuning on the D4RL benchmark suite. The agents are pretrained for 500K steps with TD3-BC before finetuning with additional online data for 200K environment steps. We perform one gradient step for every environment step, so the number of learner steps after 500K corresponds to the number of environment steps taken during finetuning. TD3-BC (orange) improves slowly with finetuning compared to TD3. However, TD3 (blue) shows policy collapse during initial finetuning. Policy collapse is more observable in the medium datasets than in the medium-replay datasets, which are more diverse. We also include results for training a TD3 agent (green) without any offline pretraining, and it performs worse than agents with offline pretraining, demonstrating that offline pretraining is useful in accelerating sample efficiency in online learning. Results are averaged from ten random seeds.

previous work [83, 67] shows similar findings for other offline RL algorithms. We experiment with using other algorithms in appendix A.2 and reach similar conclusions.

Online finetuning with online off-policy algorithms suffers from policy collapse. While online finetuning with TD3 achieves a better evaluation score compared to TD3-BC, there is a noticeable training instability for some datasets in the early stages of online finetuning. This is illustrated by the sudden drop in performance as finetuning starts (i.e., 500K learner steps). This is distinct from the fluctuations in performance that are typical in deep off-policy RL algorithms. This phenomenon is sometimes referred to as *policy collapse*. Policy collapse happens as the critic is inaccurate when finetuning starts and is over-optimistic

on novel states encountered early during finetuning. Nevertheless, finetuning with TD3 after 200K environment steps always rivals or surpasses TD3-BC. Note that policy collapse is not a specific problem for TD3: [67, 83] consider offline pretraining with CQL followed by finetuning with SAC and observe similar training instability in their experiments.

Policy collapse is more severe when the diversity of the dataset is lower. The extent of policy collapse varies across domains and dataset qualities and is more noticeable as the diversity of the dataset decreases. Although the offline performance on the medium and medium-replay datasets are comparable, finetuning from agents pretrained with TD3 on the medium datasets is more unstable. This is expected since the medium datasets are less diverse than the medium-replay datasets. Notice that the rate at which TD3 recovers its original performance also varies across the datasets, and it is more difficult to recover the performance when pretrained on less diverse datasets.

We also see the different extents of policy collapse between the medium datasets. In particular, there is significant policy degradation during early finetuning for the Ant, Hopper, and Walker2D medium datasets while the degradation is less severe for HalfCheetah. We found this to be due to the difference in environment dynamics between the environments: the Ant, Hopper and Walker2D environments in Gym define terminal states, and an episode may terminate either due to entering these terminal states or due to timeout. However, the HalfCheetah environment does not use terminal states, and episodes are always truncated at 1000 steps. We find that most of the episodes in Ant, Hopper and Walker2D medium datasets finish due to a timeout, and there are much fewer transitions in these datasets that lead to terminal states compared to the medium-replay datasets in these environments. The small number of terminal transitions in these datasets may result in the pretrained critic to significantly overestimating the state-action values around states that lead to termination. As finetuning begins and the agents encounter more terminal states, the inaccurate value estimates result in unstable policy improvement, as seen in the results above. However, finetuning with policy-constrained offline methods such as TD3-BC maintains a strong stability guarantee despite the lack of these terminal transitions in the

medium dataset.

3.3.3 Effect of offline data during finetuning

We investigate how different approaches to sampling online and offline experiences affect finetuning efficiency.

In section 3.3.2, we followed the finetuning protocol used in [81, 94], which loads the offline data into the online replay before finetuning starts. This is a commonly used strategy to utilize offline datasets during online learning, but it is not immediately clear whether this choice is good. As we will see in the following experiments, uniform sampling with offline dataset initialization can have significant performance implications.

Finetuning without offline dataset initialization

To understand the effect of offline dataset initialization, we first consider the exact opposite approach of utilizing offline datasets – that is, we do not use offline data during online finetuning and initialize an empty replay buffer during the online stage. In this case, we are just initializing the online policy and critic networks with the pretrained weights.

Figure 3.3 shows the effect of loading the offline data into the online replay before finetuning begins. TD3-BC remains stable during finetuning even when offline data is not utilized. In fact, discarding the offline data allows it to obtain higher sample efficiency and rival or even outperform online finetuning on six out of the eight datasets. This indicates that constraining the policy outputs to be close to actions stored in the replay offers improved stability even when the replay data is collected purely from the online environment. At the same time, however, this regularization via BC may hurt finetuning if the goal is to maximize online improvement given a small number of online interactions. TD3-BC can still perform worse than online RL algorithms, as evident from the lower sample efficiency in the HalfCheetah datasets. TD3 can also obtain higher performance without utilizing offline data, and finetuning is more stable with pretraining on medium-replay datasets than medium datasets. The only exception is walker2d-medium-replay, where not initializing with offline data



Figure 3.3: Effect of using offline datasets for online finetuning. We compare what happens if we do not initialize the online replay with transitions from the offline dataset. TD3-BC enjoys significant improvement if we do not sample offline transitions during online finetuning. TD3 exhibits policy collapse independent of whether offline data is utilized during finetuning.

results in policy collapse. However, for the less diverse medium datasets, policy collapse happens independent of whether offline data is reused, and the drop in performance happens immediately as finetuning starts.

3.3.4 Conservative policy improvement in TD3

While offline RL algorithms constrain the policy to be close to the empirical data distribution, such a constraint is inadequate for finetuning since it may be too conservative to allow for fast online learning. On the other hand, constraining policy optimization to the vicinity of a historically good policy may be beneficial since it may limit the influence of an inaccurate critic.

Therefore, we try to improve the online TD3 algorithm by changing the unconstrained policy improvement step to a constrained update that penalizes large policy updates. Concretely, we propose TD3-C, an off-policy deep RL algorithm based on TD3 that uses the following constrained policy improvement

step in place of the original TD3 policy optimization step:

$$\max_{\theta} \mathbb{E}_{s \sim \mathcal{B}}[Q_{\phi}(s, a)|_{a=\pi_{\theta}(s)}] \quad \text{s.t.} \quad \mathbb{E}_{s \sim \mathcal{B}}[(\pi_{\theta}(s) - \pi_{\theta'}(s))^2] \leq \varepsilon. \quad (3.4)$$

Here θ is the online policy network parameter, θ' is the target policy network parameter, and ε is a hyper-parameter that controls the tightness of the constraint. The constraint regularizes the online policy to not deviate too much from a moving target policy. This formulation resembles the constrained optimization used in MPO [32], except that we work with a deterministic policy and use the ℓ_2 norm as the constraint. The constraint has the interpretation of a KL divergence between two Gaussian policies with fixed location and scale parameterized with the output from the online and target policies' output. For a practical implementation, we can optimize the objective by formulating the Lagrangian with dual variables $\lambda \geq 0$. The constrained optimization now becomes

$$\max_{\theta} \min_{\lambda \geq 0} \mathbb{E}_{s \sim \mathcal{B}}[Q_{\phi}(s, a) - \lambda[\varepsilon - (a - \pi_{\theta'}(s))^2]], \quad a = \pi_{\theta}(s), \quad (3.5)$$

where the primal θ and dual variables λ can be jointly optimized by stochastic gradient descent.

Unlike the behavior cloning regularization in eq. (3.3), which constrains the policy network to predict actions similar to those in the sampled batches, we constrain the policy optimization not to take big steps. Thus, our formulation should be less conservative than TD3-BC but more robust than TD3 without policy regularization.

Is conservative policy optimization effective at mitigating policy collapse?

We investigate if incorporating conservative policy optimization helps mitigate policy collapse. Figure 3.4 shows the finetuning performance between TD3, TD3-BC and TD3-C on the eight datasets with or without initializing the online replay from the offline dataset. While TD3-C improves training stability in both settings, the effect is more significant when no offline data is used during finetuning. The results illustrate the difference between the constraint used in TD3-C and the behavior cloning regularizer in TD3-BC. For TD3-BC, the policy is regularized towards the empirical behavior policy, which is crucial for

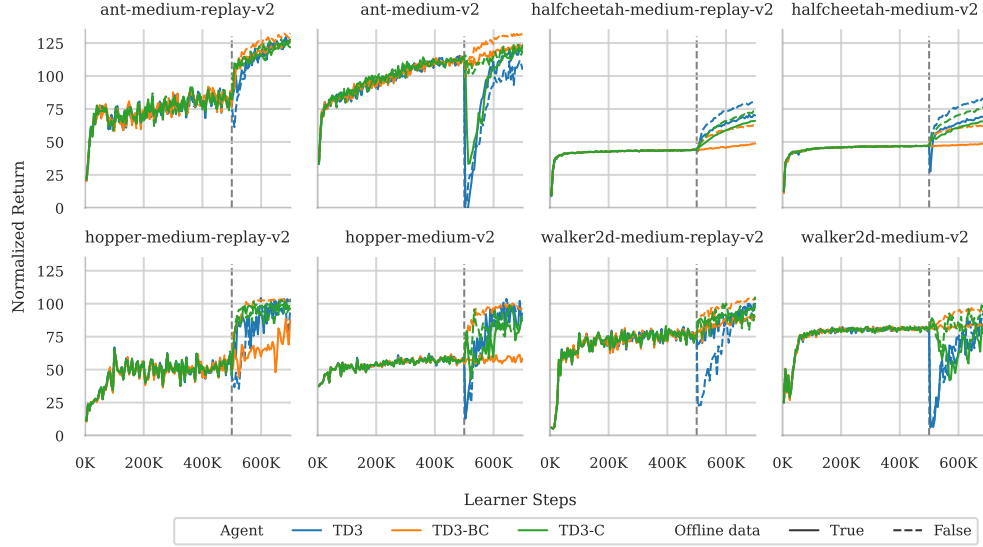


Figure 3.4: Results of TD3, TD3-BC, and TD3-C for finetuning. We compare the three approaches for finetuning by varying whether we initialize the online replay buffer with transitions from the offline dataset. TD3-C demonstrates better stability compared to TD3 and can improve faster compared to finetuning with TD3-BC.

minimizing extrapolation errors but may result in slow improvement. For TD3-C, we regularize the policy towards the changing target policy, avoiding large policy changes during optimization due to an inaccurate critic. However, TD3-C can still collapse when preloading the online replay with offline data, as seen in `ant-medium-v2`. As explained in section 3.3.3, the online sampling distribution remains almost unchanged during the initial period of finetuning. Using TD3 or TD3-C on this distribution can suffer from overestimation errors due to the lack of prompt online feedback. Thus, the regularizer in TD3-C, coupled with uniform sampling from an online replay buffer initialized with offline transitions, can still suffer from policy collapse due to optimization with a “fixed” dataset.

Improving training stability with more conservative policy improvement.

TD3-C introduces an ϵ parameter that determines the level of the constraint used during policy improvement. We investigate different choices of ϵ . The results are shown in fig. 3.5. Generally, we find that choosing stronger constraints leads to more stable finetuning but may result in slower finetuning efficiency.



Figure 3.5: Effect of varying the constraint parameter ϵ . We found that smaller ϵ leads to more stable online finetuning but slower improvement.

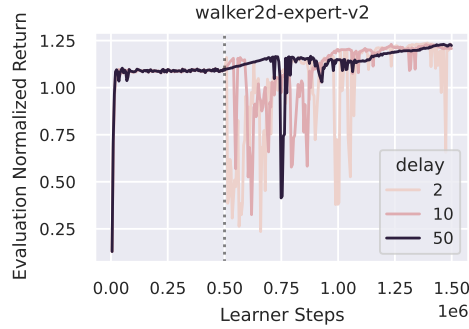


Figure 3.6: Effect of increasing the number of delay steps in TD3-C. We found that constrained updates alone do not prevent policy collapse, but reducing the amount of policy improvement steps mitigates it.

Improving training stability with more delayed policy improvements. The policy improvement constraint penalty stabilizes learning for the medium datasets. However, when only expert demonstrations are used in offline learning, the critic suffers from severe overfitting, and constrained updates alone are insufficient to prevent policy collapse. In this case, updating the critic more frequently stabilizes finetuning. Note that reducing update frequency is different from constraining policy updates: constraining the policy optimization has the added benefit of making policy optimization more robust to critic error. Our concrete implementation takes advantage of the delay parameter in TD3, which determines the frequency of policy optimization.

Figure 3.6 compares different delayed step values with evaluation perfor-

mance. We see that increasing the delay helps prevent policy collapse during initial finetuning. This ablation suggests that reducing the ratio of policy improvement to policy evaluation steps may help when adopting standard off-policy algorithms for finetuning.

3.3.5 Comparative evaluation

In this section, we position our results in the context of several recent literature that also considers finetuning from offline pretrained agents. The goal of this section is not to argue that our proposed approach, either using TD3-BC, TD3-BC, or TD3-C introduced above, is a better algorithm compared to related work since neither TD3 nor TD3-BC represents state-of-the-art online/offline RL algorithms. Instead, this section positions a relatively simple approach to online finetuning to other methods introduced in this line of work. Our analysis in section 3.3 aims to disentangle and understand the merits of individual algorithmic choices used by the different algorithms. We hope the comparative analysis in this section can inspire the development of more sample-efficient RL finetuning algorithms.

We present our evaluation results with respect to Online Decision Transformer (ODT) [94] and Implicit Q-learning (IQL). ODT is a recent approach based on the Decision Transformer (DT) [94]. Similar to DT, it formulates RL as a sequence modeling problem and leverages the expressiveness of transformer architectures to learn in the RL setting. IQL is an offline Q-learning method that learns the optimal Q-function using in-sample transitions and extracts a policy with Advantage Weighted Regression (AWR) [14, 41]. Both approaches have demonstrated better performance when used in the finetuning setting compared to previous work [67, 46, 38]. For a consistent comparison with IQL and ODT, we consider the setting where we initialized the online replay buffer with the offline dataset. As noted above, initializing the online buffer with offline data may be sub-optimal, especially with policy-constrained offline RL algorithms. For all methods, we report the offline performance, performance after 200K steps of online interactions and the relative performance improvement δ , computed as the difference between the online and offline performance.

Table 3.1 shows that our proposed approaches are consistently better than ODT and IQL. This is evident in the better evaluation results after finetuning

Table 3.1: Results with ODT [94], and IQL [81] on the D4RL MuJoCo locomotion tasks. The ODT and IQL results were obtained from [94]. Results are averaged over ten seeds. We report the average final offline performance (Offline), the final online performance after 200K of online steps (Online), and the relative performance improvement $\delta = \text{Online} - \text{Offline}$. Results for the method with the best performance and comparable alternatives (within 10% of the best method’s mean) are in **bold**.

Task	Agent	medium-replay-v2			medium-v2		
		Offline	Online	δ	Offline	Online	δ
ant	TD3	78.34 \pm 19.5	127.59 \pm 5.24	49.25	113.48 \pm 4.74	123.34 \pm 2.67	9.86
	TD3-C	85.68 \pm 13.93	121.72 \pm 10.06	36.04	112.44 \pm 5.71	123.05 \pm 2.31	10.61
	TD3-BC	84.99 \pm 15.95	126.73 \pm 3.15	41.74	111.44 \pm 5.79	120.91 \pm 7.97	9.47
	ODT	86.56 \pm 3.26	91.57 \pm 2.73	5.01	91.33 \pm 4.13	90.79 \pm 5.8	−0.54
	IQL	91.21 \pm 7.27	91.36 \pm 1.47	0.15	99.92 \pm 5.86	100.85 \pm 2.02	0.93
	TD3 (online)	78.6 \pm 22.55	122.91 \pm 15.16	44.31	112.34 \pm 6.11	104.71 \pm 8.48	−7.63
	TD3-C (online)	82.2 \pm 24.9	125.5 \pm 5.19	43.3	114.15 \pm 4.7	119.34 \pm 7.9	5.19
	TD3-BC (online)	76.63 \pm 16.31	132.34 \pm 2.92	55.71	112.95 \pm 6.21	132.99 \pm 1.43	20.04
	TD3	43.82 \pm 0.49	70.13 \pm 3.4	26.3	46.99 \pm 0.4	69.69 \pm 2.57	22.7
	TD3-C	43.84 \pm 0.69	66.0 \pm 2.06	22.16	46.92 \pm 0.42	65.85 \pm 2.46	18.93
halfcheetah	TD3-BC	43.74 \pm 0.53	48.7 \pm 1.18	4.96	46.92 \pm 0.41	48.71 \pm 0.52	1.79
	ODT	39.99 \pm 0.68	40.42 \pm 1.61	0.43	42.72 \pm 0.46	42.16 \pm 1.48	−0.56
	IQL	44.1 \pm 1.14	44.14 \pm 0.3	0.04	47.37 \pm 0.29	47.41 \pm 0.15	0.04
	TD3 (online)	43.77 \pm 0.59	80.98 \pm 3.61	37.21	46.84 \pm 0.43	82.47 \pm 2.7	35.63
	TD3-C (online)	43.7 \pm 0.63	73.52 \pm 2.17	29.81	46.86 \pm 0.36	75.93 \pm 3.28	29.08
	TD3-BC (online)	43.83 \pm 0.47	62.35 \pm 1.41	18.52	46.96 \pm 0.37	62.88 \pm 1.25	15.92
	TD3	46.21 \pm 21.91	103.08 \pm 3.7	56.87	54.95 \pm 3.75	88.59 \pm 28.4	33.65
	TD3-C	49.18 \pm 25.6	96.24 \pm 11.3	47.06	56.54 \pm 4.71	87.3 \pm 24.62	30.77
	TD3-BC	51.36 \pm 24.94	87.72 \pm 14.96	36.36	55.48 \pm 4.69	58.44 \pm 6.72	2.96
	ODT	86.64 \pm 5.41	88.89 \pm 6.33	2.25	66.95 \pm 3.26	97.54 \pm 2.1	30.59
hopper	IQL	92.13 \pm 10.43	96.23 \pm 4.35	4.1	63.81 \pm 9.15	66.79 \pm 4.07	2.98
	TD3 (online)	44.84 \pm 23.2	94.52 \pm 25.1	49.68	55.14 \pm 4.48	90.91 \pm 33.52	35.78
	TD3-C (online)	45.69 \pm 22.31	102.38 \pm 3.01	56.69	56.38 \pm 5.86	90.58 \pm 20.91	34.2
	TD3-BC (online)	49.22 \pm 22.95	102.75 \pm 2.43	53.53	54.35 \pm 4.86	98.8 \pm 9.72	44.45
	TD3	72.56 \pm 11.37	100.06 \pm 5.74	27.5	80.88 \pm 4.9	82.09 \pm 21.38	1.21
	TD3-C	74.78 \pm 10.97	97.21 \pm 3.07	22.44	79.34 \pm 5.7	78.97 \pm 21.03	−0.36
	TD3-BC	79.12 \pm 7.2	90.26 \pm 2.27	11.14	80.73 \pm 3.72	84.56 \pm 2.09	3.82
	ODT	68.92 \pm 4.79	76.86 \pm 4.04	7.94	72.19 \pm 6.49	76.79 \pm 2.3	4.6
	IQL	73.67 \pm 6.37	70.55 \pm 5.81	−3.12	79.89 \pm 3.06	80.33 \pm 2.33	0.44
	TD3 (online)	72.66 \pm 10.59	98.94 \pm 10.38	26.28	80.16 \pm 5.1	85.19 \pm 19.55	5.03
walker2d	TD3-C (online)	75.24 \pm 9.78	105.35 \pm 11.01	30.12	80.41 \pm 4.99	97.54 \pm 11.88	17.13
	TD3-BC (online)	73.55 \pm 8.18	104.16 \pm 4.66	30.61	82.26 \pm 1.32	97.48 \pm 3.45	15.22

for 200K steps. Note that obtaining better offline performances alone cannot explain the better final finetuning performance before finetuning begins since offline learning with TD3-BC performs no better than ODT and IQL except for ant-medium-v2. Even in cases where pretraining with TD3-BC delivers better offline performance than ODT or IQL, we still observe more significant performance improvement, as evidenced by the larger relative improvement. In hopper-medium-replay-v2, although pretraining with TD3-BC performs worse during offline learning, finetuning with TD3 or TD3-C allows us to improve significantly.

Note that ODT performs hyperparameter tuning for each task and initializes

the replay buffer using only top-performing trajectories from the offline dataset. In contrast, our approach is easy to implement and requires minimal changes to existing algorithms during online finetuning. Furthermore, we use the same hyperparameters for all datasets, and the hyperparameters are chosen to be the same as the default values used in previous work. We also do not change how data are sampled from the replay buffer and perform additional pre-processing to the dataset.¹

So far, we have compared finetuning with TD3-style algorithms using the same offline data initialization method used in ODT and IQL. As demonstrated in earlier experiments in section 3.3.3, this setting is highly disadvantageous for online TD3-BC. If we compare with results that do not first initialize the online buffer with offline datasets, then finetuning with TD3 and TD3-C still outperforms finetuning with ODT and IQL significantly. At the same time, the performance of finetuning with TD3-BC also improves due to constraining the policy to only online data collected during finetuning. In fact, it performs the same or better than TD3 and TD3-C on all except the HalfCheetah datasets. At the same time, the variance in the final results obtained by TD3-BC is consistently lower, demonstrating that it is more stable online than other algorithms.

We also compare with other RL algorithms that aim to address the finetuning inefficiency issues discussed in this chapter in appendix A.3 and appendix A.4.

3.3.6 Summary of empirical observations

In the following, we summarize our observations of the empirical study.

- The degree of the conservativeness of the RL algorithms has a substantial impact on the data efficiency during finetuning, where the goal is to maximize the improvement compared to the pretrained policies. Mitigation strategies that reduce extrapolation errors during offline training can also improve the stability of online training. However, this may come as a trade-off for online finetuning sample efficiency. Using online RL algorithms may result in a faster rate of improvement but can be less stable, and such

¹The original TD3-BC performs observation normalization, which shows improved performance in some environments. In our implementation, we do not normalize the observations.

instability may be undesirable if the online sampling budget is limited. The degree of instability depends on the properties of the underlying MDP and the diversity of the offline dataset. When the offline dataset has enough diversity, the errors encountered during online finetuning will be mild and may not be significant enough to collapse a good pretrained policy. However, if the value estimate is inaccurate during finetuning, then policy optimization algorithms that maximize the erroneous value estimate will result in policy collapse.

- The online replay sampling distribution plays an important role. The two approaches we considered, namely whether we initialize the online replay with offline datasets, present different trade-offs. However, when the online replay is initialized with the offline dataset, the sampled transitions during the initial finetuning period should resemble those seen during offline training. For policy-constrained offline RL algorithms such as TD3-BC, the additional policy constraint will regularize the online training policy heavily towards sub-optimal behaviors in the offline dataset. This explains why TD3-BC improves faster when we completely discard offline data during finetuning. However, online algorithms that do not incorporate any additional constraints suffer from training instability.
- Initializing the online replay with offline data followed by uniform sampling during finetuning may present additional issues that influence the stability of online algorithms such as TD3. When the offline dataset is large, during the initial period of finetuning, samples from the replay buffer would consist mainly of transitions from the offline dataset since the amount of new samples is relatively small compared to the number of offline samples. This can create a similar effect as learning offline during online finetuning. Since TD3 is not designed for offline learning, the values may be overestimated during initial finetuning. We discuss this issue in more detail in section 3.3.4. A solution that prevents the offline samples from crowding out the online replay is to fix the ratio of offline and online samples during finetuning. This has the advantage of ensuring that the agent immediately uses online interactions collected by the agent during

finetuning, decoupling the rate at which online transitions are sampled from the size of the offline dataset.

- When the offline data is not used to initialize the online replay, all transitions sampled during finetuning will consist purely of online interactions. The critic may encounter more transitions not seen during pretraining, making stable finetuning difficult. The agent may also suffer from the risk of catastrophic forgetting since the agent would not be able to recall any bad behavior it has experienced during offline learning. While we have presented experimental results for the two extreme settings, it is unlikely that either would be the best solution for practical applications. While [83] explored using prioritized replay that learns to mix the offline and online samples that allow utilization of offline samples during online learning, the issue of policy collapse can still happen with their approach. Therefore, our experiments suggest that if the online algorithm does not explicitly address issues that prevent extrapolation errors in the critic from crippling the policy, it is unlikely that a different choice of the replay sampling strategy can help prevent policy collapse.

3.4 Discussion

One may hope to deploy the offline policy to collect more data and reuse the same algorithm for offline learning and finetuning. However, existing studies and our findings indicate that finetuning with RL algorithms designed for offline learning converges slowly with additional online data [67]. An alternative approach to reusing the offline RL algorithm for online finetuning is to use a different online off-policy RL algorithm for online finetuning. Since recent online off-policy algorithms [33, 26, 32, 34] have demonstrated strong performance and good sample efficiency, we should expect that additional pretraining with offline RL would allow this approach to enable sample-efficient finetuning. We show that using a standard off-policy RL algorithm can work well for online finetuning. However, we also observe that sometimes finetuning with online off-policy algorithms can lead to *policy collapse*, where the policy performance degrades severely

during initial online training. These observations motivate us to investigate the challenges in different strategies for finetuning from offline RL. Towards this goal, we analyze the trade-offs in choices of algorithms and whether/how to use offline data for online finetuning. We present several approaches to finetuning and discuss their merits and limitations based on empirical observations on standard offline and online RL benchmarks. Based on our observations, we conclude that effective online finetuning from offline RL may be achieved with more robust online policy optimization and present a constrained policy optimization extension to the TD3 algorithm, which we call conservative TD3 (TD3-C), that empirically helps stabilize online finetuning, thereby addressing the issue of policy collapse.

Practical guidelines

Our observations provide a few points for practitioners to consider when leveraging offline RL as a pretraining step and improving offline policies with online data. First, using offline RL methods to continue finetuning may be sub-optimal if the goal is to maximize performance improvement with additional online samples. Using online off-policy algorithms may allow convergence to a better final policy. Second, our empirical results suggest that from a practical perspective, it is crucial to pretrain the agents on datasets that include diverse transitions, provided the offline RL algorithm does not deteriorate significantly with the addition of these suboptimal transitions. While both the policy and the critic can enjoy more robustness and better generalization capability with diverse suboptimal data, the critic should be exposed to both good and bad actions during pretraining to ensure that finetuning with online algorithms is stable.

Our results reveal some limitations on the reported benchmark results in finetuning from offline RL. Previous work [67, 81, 94] demonstrates that some offline RL algorithms enjoy better finetuning performance. However, the conclusion is usually made by comparing alternative offline RL methods for finetuning or an online RL baseline. In this chapter, we demonstrate that using online RL algorithms with offline RL pretraining is a simple and effective approach that is often neglected in previous work, such as [67, 94, 81] with the exception of [83]. However, we do not argue that online RL for finetuning is necessarily

superior and should always be preferred. As we have seen in section 3.3, policy constraint methods, such as TD3-BC, enjoy better stability, and the performance can sometimes rival online RL algorithms. However, we argue that, instead of utilizing constraints designed for offline learning, there are alternative constraints that would work better for online finetuning. We discussed one approach using conservative policy optimization in section 3.3.4. While we do not find finetuning with TD3-BC to suffer from policy collapse, other works [85, 49] have shown that policy collapse can indeed happen even when using offline RL methods for finetuning. We expect that incorporating conservative policy optimization can also help improve stability in those cases.

Our study also suggests that the evaluation protocols used by some previous works do not sufficiently reflect the challenges we need to address in finetuning from offline RL. We find that the conclusion drawn from evaluating finetuning performance depends on the number of online samples allowed. When a small number of online samples is chosen, the evaluation will favor algorithms with better stability, but not necessarily algorithms that are more finetuning-efficient. In practice, a trade-off between stability and relative performance improvement exists, and we hope future work can account for it in consideration during evaluation.

Recent work aiming to improve finetuning from offline RL often incorporates algorithmic improvements [67, 81, 83, 94] coupled with changes to the underlying actor-critic algorithms and replay sampling strategy. This makes it difficult to understand performance improvements coming from individual components, hindering our understanding of what makes finetuning from offline RL difficult. We attempt to isolate these changes and demonstrate that, by just changing the online algorithms during finetuning or the online replay initialization, existing algorithms can have a significant performance boost that rivals or outperforms more sophisticated approaches. Given the relative ease of implementing these changes, we hope future work can incorporate them as baselines to measure better our progress on finetuning from offline RL.

3.5 Conclusion

We studied the difficulty in leveraging offline RL as pretraining for online RL. We found that finetuning with offline RL results in slow improvement, while finetuning with online RL algorithms is sensitive to distribution shifts. We found that conservative policy optimization is a promising approach for stabilizing finetuning from offline RL when the offline dataset lacks diversity.

Chapter 4

Optimal transport for offline imitation learning

4.1 Overview

In this chapter, we introduce Optimal Transport Reward labeling (OTR), an algorithm that leverages optimal transport theory to automatically assign reward labels to unlabeled trajectories in an offline dataset based on one or more expert demonstrations.

This chapter considers leveraging datasets that are imperfect due to missing reward information. In chapter 3, we discussed how offline RL can be used to learn a policy from a complete dataset containing observations, actions, and rewards, which can then be further improved through online finetuning. However, this approach assumes the presence of reward signals in the offline dataset. The necessity for a reward function to label logged experiences limits the direct application of offline RL methods in scenarios where rewards are difficult to specify with hand-crafted rules. While human preferences can be used to label trajectories, this process can be prohibitively expensive. Therefore, enabling offline RL to utilize unlabeled data remains an open and significant challenge.

An alternative to labeling every trajectory is to provide expert demonstrations, which can often be more natural for practitioners than specifying a reward function. In robotics, for instance, providing expert demonstrations is common practice, and in the absence of natural reward functions, ‘learning from demon-

stration’ has been a standard approach for decades to derive effective policies for robotic systems (see, e.g., [5, 11, 12, 20]). Imitation learning, a framework within this approach, aims to learn policies that replicate expert behavior without needing an explicit reward function.

The OTR algorithm introduced in this chapter employs optimal transport to find optimal alignments between unlabeled trajectories in the dataset and expert demonstrations. The similarity measure between a state in an unlabeled trajectory and that of an expert trajectory is treated as a reward label. These inferred rewards can then be used by any offline RL algorithm to learn policies from a small number of expert demonstrations and a large offline dataset. Empirical evaluations on the D4RL [63] datasets demonstrate that OTR can recover the performance of offline RL methods using ground-truth rewards with only a single demonstration. Compared to previous reward learning and imitation learning approaches, our method consistently achieves superior performance across a wide range of offline datasets and tasks.

4.2 Preliminaries

We consider learning in an episodic, finite-horizon Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, p, r, \gamma, p_0, T)$ where \mathcal{S} is the state space, \mathcal{A} is the action space, p is the transition function, r is the reward function, γ is the discount factor, p_0 is the initial state distribution and T is the episode horizon. A policy π is a function from state to a distribution over actions. The goal of RL is to find policies that maximize episodic return. Running a policy π in the MDP generates a state-action episode/trajectory $(s_1, a_1, s_2, a_2, \dots, s_T) =: \tau$.

We consider the problem of *offline imitation learning*. Unlike in the standard RL setting, no explicit reward function is available. Let $\tau = (s_1, a_1, s_2, a_2, \dots, s_T)$ denote an episode of interaction with the MDP using a policy that selects actions a_t at time steps $t = 1, \dots, T - 1$. Instead of a reward function, we have access to a dataset of *expert* demonstrations $\mathcal{D}^e = \{\tau_e^{(n)}\}_{n=1}^N$ generated by an expert policy π_e and a large dataset of *unlabeled* trajectories $\mathcal{D}^u = \{\tau_\beta^{(m)}\}_{m=1}^M$ generated by an arbitrary behavior policy π_β . We are interested in learning an offline policy π combining information from expert demonstrations and unlabeled experience

without any interaction with the environment. We address this problem using optimal transport, which will provide a way to efficiently annotate large offline RL datasets with rewards. Figure 4.1 illustrates how OTR uses expert demonstrations to add reward labels to an offline dataset, which can then be used by an offline RL algorithm to find a good policy that imitates demonstrated behavior.

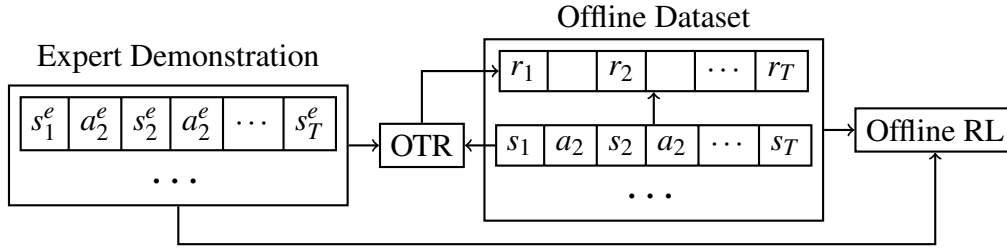


Figure 4.1: Illustration of Optimal Transport Reward Labeling (OTR). Given expert demonstrations (left) and an offline dataset without reward labels (center), OTR adds reward labels r_i to the offline dataset by means of optimal transport (orange, center). The labeled dataset can then be used by an offline RL algorithm (right) to learn policies.

4.3 Optimal transport reward labeling

Optimal Transport (OT) [19, 51] is a principled approach for comparing probability measures. The (squared) Wasserstein distance between two discrete measures $\mu_x = \frac{1}{T} \sum_{t=1}^T \delta_{x_t}$ and $\mu_y = \frac{1}{T'} \sum_{t=1}^{T'} \delta_{y_t}$ is

$$\mathcal{W}^2(\mu_x, \mu_y) = \min_{\mu \in M} \sum_{t=1}^T \sum_{t'=1}^{T'} c(x_t, y_{t'}) \mu_{t,t'}, \quad (4.1)$$

where $M = \{\mu \in \mathbb{R}^{T \times T'} : \mu \mathbf{1} = \frac{1}{T} \mathbf{1}, \mu^T \mathbf{1} = \frac{1}{T'} \mathbf{1}\}$ is the set of coupling matrices, c is a cost function, and δ_x refers to the Dirac measure for x . The optimal coupling μ^* provides an alignment between the samples in μ_x and μ_y . Unlike other divergence measures (e.g., KL divergence), the Wasserstein distance is a metric and it incorporates the geometry of the space.

In the context of reinforcement learning, let $\hat{p}_e = \frac{1}{T'} \sum_{t=1}^{T'} \delta_{s_t^e}$ and $\hat{p}_\pi = \frac{1}{T} \sum_{t=1}^T \delta_{s_t^\pi}$ denote the empirical state distribution of an expert policy π_e and

behavior policy π respectively. Then, the (squared) Wasserstein distance

$$\mathcal{W}^2(\hat{p}_\pi, \hat{p}_e) = \min_{\mu \in M} \sum_{t=1}^T \sum_{t'=1}^{T'} c(s_t^\pi, s_{t'}^e) \mu_{t,t'} \quad (4.2)$$

can be used to measure the distance between expert policy and behavior policy. Let μ^* denote the optimal coupling for the optimization problem above, then eq. (4.2) provides a reward signal

$$r_{\text{ot}}(s_t^\pi) = - \sum_{t'=1}^{T'} c(s_t^\pi, s_{t'}^e) \mu_{t,t'}^*, \quad (4.3)$$

which can be used for learning policy π in an imitation learning setting. This is the key idea behind our approach to annotating unlabeled datasets with reward signals: computing the optimal alignment between the expert demonstration and trajectories in the unlabeled dataset allows us to assign a reward for each step in the unlabeled trajectory.

Figure 4.2 illustrates the computation performed by OTR to annotate an unlabeled dataset with rewards using demonstrations from an expert. We also gave the pseudo-code for the approach in algorithm 1. Concretely, OTR takes the unlabeled dataset \mathcal{D}^u and expert demonstration \mathcal{D}^e as input. For each unlabeled trajectory $\tau^{(m)} \in \mathcal{D}^u$, OTR solves the optimal transport problem for each, obtaining the cost matrix $C^{(m)}$ and optimal alignment $\mu^{*(m)}$ (line 3). OTR then computes the per-step reward label following eq. (4.3) (line 5). The reward-annotated trajectories are then combined, forming a reward-labeled dataset $\mathcal{D}^{\text{label}}$.

Solving eq. (4.2) requires solving the OT problem to obtain the optimal coupling matrix μ^* . This amounts to solving a linear program (LP), which may be prohibitively expensive with a standard LP solver. In practice, we solve the entropy-regularized OT problem with Sinkhorn’s algorithm [19]. We leverage the Sinkhorn solver in OTT-JAX [75] for this computation. Once OTR has annotated the unlabeled offline dataset with intrinsic rewards, we can use an offline RL algorithm to learn a policy. Since we are working in the pure offline setting, it is important to use an offline RL algorithm that can minimize the distribution shifts typically encountered in the offline setting.

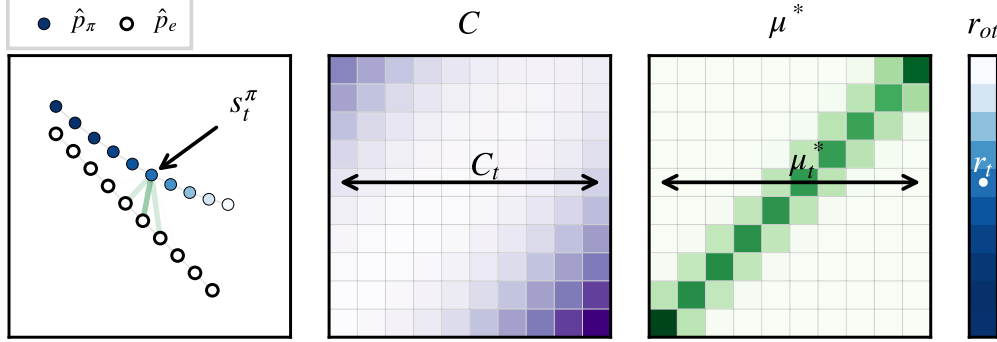


Figure 4.2: Illustration of the computations performed by OTR. In this example, we consider an MDP with a two-dimensional state space ($|\mathcal{S}| = 2$). We have two empirical state distributions from an expert \hat{p}_e with samples $\{s_{t'}^e\}_{t'=1}^{T'}$ (\circ) and policy \hat{p}_π with samples $\{s_t^\pi\}_{t=1}^T$ (\bullet) as denoted by points in the leftmost figure. OTR assigns rewards r_{ot} (blue) to each sample in the policy’s empirical state distribution as follows: (i) Compute the pairwise cost matrix C (purple) between expert trajectories and trajectories generated by behavior policy; (ii) Solve for the optimal coupling matrix μ (green) between \hat{p}_π and \hat{p}_e ; (iii) Compute the reward for s_t^π as $r_{ot}(s_t^\pi) = -C_t^T \mu_t^*$. Consider for example a state $s_t^\pi \in \hat{p}_\pi$; the row C_t in the cost matrix corresponds to the costs between s_t^π and $\{s_{t'}^e\}_{t'=1}^{T'}$. μ_t^* represents the optimal coupling between s_t^π and the expert samples. The optimal coupling moves most of the probability mass to s_3^e and a small fraction of the mass to s_4^e (green lines in the leftmost figure).

One thing to note is that the reward computed via OTR corresponds to the Wasserstein distance between the stationary distribution of the expert policy π_e and the stationary distribution of the policy π_β that generates the unlabeled dataset. It does not correspond to the Wasserstein distance between the training policy and the expert. Therefore, optimizing the policy with respect to rewards computed by OTR does not exactly correspond to direct minimization between the learning policy and the expert. Nevertheless, the rewards computed using the behavior policy is an upperbound on the Wasserstein distance between the expert and the learning policy. Therefore, minimizing the rewards computed by OTR corresponds to minimizing an upper bound on the Wasserstein distance between the learning and expert policy:

$$\mathcal{W}^2(p_\pi, p_e) \leq \mathcal{W}^2(p_\pi, p_\beta) + \mathcal{W}^2(p_\beta, p_e). \quad (4.4)$$

The upper bound becomes relaxed when the learning policy π differs significantly

Algorithm 1: Pseudo-code for Optimal Transport Reward labeling (OTR)

Input: unlabeled dataset \mathcal{D}^u , expert dataset \mathcal{D}^e
Output: labeled dataset $\mathcal{D}^{\text{label}}$

```

1  $\mathcal{D}^{\text{label}} \leftarrow \emptyset;$ 
2 foreach  $\tau^{(m)}$  in  $\mathcal{D}^u$  do                                // Label each episode in the
   unlabeled dataset
3    $C^{(m)}, \mu^{*(m)} \leftarrow \text{SolveOT}(\mathcal{D}^e, \tau^{(m)});$     // Compute the optimal
   alignment with eq. (4.2)
4   for  $t = 1$  to  $T$  do
5      $r_{\text{OT}}(s_t^{(m)}) \leftarrow -\sum_{t'=1}^{T'} C_{t,t'}^{(m)} \mu_{t,t'}^{*(m)};$  // Compute the per-step
     rewards with eq. (4.3)
6   end
7    $\mathcal{D}^{\text{label}} \leftarrow \mathcal{D}^{\text{label}} \cup (s_1^{(m)}, a_1^{(m)}, r_1^{\text{OT}}, \dots, s_T^{(m)});$  // Append labeled
   episode
8 end
9 return  $\mathcal{D}^{\text{label}};$ 

```

from the behavior policy. However, since offline RL uses behavior regularization to minimize the effect of distribution shifts, $\mathcal{W}^2(p_\pi, p_\beta)$ is controlled, and minimizing with respect to the state rewards should minimize the actual Wasserstein distance between the learning and expert policy.

Unlike prior works that compute rewards using online samples [60, 59], we compute the rewards entirely offline prior to running offline RL training, avoiding the need to modify any part of the downstream offline RL pipeline. Therefore, our approach can be combined with any offline RL algorithms, providing dense reward annotations that are required by the downstream algorithms. Figure 4.1 illustrates the entire pipeline of using OTR for relabeling and running an offline RL algorithm using the reward annotated datasets.

Compared to previous work that aims at solving offline imitation learning with a single algorithm, OTR focuses on generating high-quality reward annotations for downstream offline RL algorithms. As a result, our approach enjoys several advantages:

- Our approach does not require training separate reward models or discriminators, which may incur higher runtime overhead. By not having to

train a separate parametric model, we avoid hyper-parameter tuning on the discriminator network architectures.

- Unlike other approaches, such as GAIL or DemoDICE, our approach does not require solving a minimax optimization problem, which can suffer from training instability [68].
- Our approach is agnostic to the offline RL methods for learning the policy since OTR computes reward signals independently of the offline RL algorithm.

Efficient implementation

We provide an efficient implementation of OTR in JAX. For computing the optimal coupling, we use OTT-JAX [75], a library for optimal transport that includes a scalable and efficient implementation of the Sinkhorn algorithm that can leverage accelerators, such as GPU or TPU, for speeding up computations. JAX includes useful functionality that allows us to easily parallelize computations. Concretely, we JIT-compile the computation of rewards for one episode and further leverage the vmap function to compute the optimal coupling between an unlabeled episode with all of the expert episodes in parallel. Efficiently parallelizing the computation of the optimal coupling requires that all the episodes share the same length. This is necessary both for parallelizing the computation across multiple expert demonstrations as well as for avoiding recompilation by XLA due to changes in the shape of the input arrays. To achieve high throughput for datasets with varying episodic length, we pad all observations to the maximum episode length allowed by the environment (which is 1000 for the OpenAI Gym environments) but set the weights of the observations to zero. Padding the episodes this way does not change the solution to the optimal coupling problem. Note that padding means that a 1M transition dataset may create more than 1000 episodes of experience, in this case, the runtime for our OTR implementation may be higher effectively due to having to process a larger number of padded episodes. Our implementations requires only about one minute to label a dataset

with one million transitions (or 1000 episodes of length 1000)¹. For larger-scale problems, OTR can be scaled up further by processing the episodes in the dataset in parallel. Our implementation of OTR and re-implementation of baselines are computationally efficient. Even so, the training time for IQL is about 20 minutes, so OTR adds a relatively small amount of overhead for reward annotation to an existing offline RL algorithm.

Cost function. For OTR, we use the cosine distance as the cost function. Let s_e and s_u denote two states from the expert and unlabeled demonstration respectively. The *cosine distance* between the expert state and unlabeled state is

$$d(s_e, s_u) = 1 - \frac{s_e^\top s_u}{\|s_e\| \|s_u\|}. \quad (4.5)$$

This choice follows the recommendation from [59] which found that the cosine distance performs empirically better than using the Euclidean distance. One practical advantage of the cosine distance is that it is bounded ($0 \leq d \leq 2$), which ensures that reward we derive from the cost function falls within a fixed range. However, the cosine distance may not be a good choice in scenarios where we would like two states that are colinear to have different costs since it only takes the angular similarity into account. Nevertheless, we find the cosine distance to be a good cost function that gives good performance across different domains and tasks.

Aggregating multiple expert trajectories. Since there can be more than one episodes of expert demonstrations, we need a way to aggregate the rewards computed from these expert demonstrations. To do this, we compute the optimal transport with respect to each episode independently and use the rewards from the expert trajectory that gives the best episodic return. This allows us to match the unlabeled episode with the nearest-neighbor expert episode instead of with an average of all expert episodes, which may not be representative of the behavior we want to imitate.

¹Runtime measured on halfcheetah-medium-v2 with an NVIDIA GeForce RTX 3080 GPU.

Reward squashing. One thing that can result in sub-optimal performance of using OTR is due to the difference in the reward scale. The reward signals computed by OTR may be significantly different from the reward scale from a hand-crafted reward function. However, many offline RL algorithms may be sensitive to the scale of the reward signals, and additional tuning of hyperparameters that are dependent on the reward scales may be required. To address this issue, similar to [60, 59], we squash the rewards computed by line 5 with an exponential function $s(r) = \alpha \exp(\beta r)$. This has the advantage of ensuring that the rewards consumed by the offline RL algorithm have an appropriate range since many offline RL algorithms can be sensitive to the scale of the reward values.

Overall, the time complexity of OTR for labeling *one* episode is $O(KT^2)$, where K is the number of expert demonstrations and T is the trajectory length. Therefore, we expect OTR to be most practical in scenarios when there is a small number of expert demonstrations and when the episode length is not prohibitively large. For long episodes, it may be possible to downsample the episode by some fixed factor or by selection of key frames. We leave investigation of very long episode length to future work. We also refer the reader to appendix B.1 for additional experimental details and hyperparameters.

4.4 Experiments

In this section, we evaluate OTR on D4RL Locomotion, AntMaze, and Adroit benchmark tasks (see also fig. 4.3) with the goal of answering the following research questions:

1. Can OTR recover the performance of offline RL algorithms that have access to a well-engineered reward function (i.e., ground-truth rewards provided by the environment)?
2. Can OTR handle unlabeled datasets with behaviors of unknown and mixed quality?
3. How does OTR perform with a varying number of expert demonstrations?

4. How does OTR compare with previous work on offline imitation learning in terms of performance and runtime complexity?

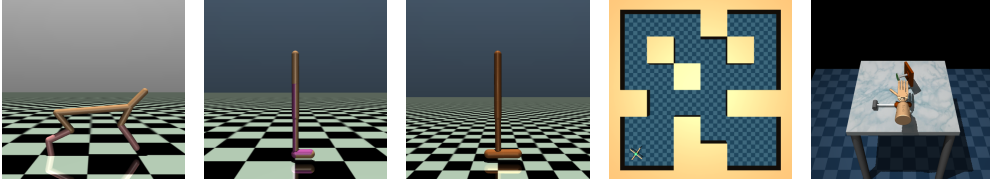


Figure 4.3: Benchmark tasks: D4RL Locomotion, Antmaze, and Adroit.

We demonstrate that OTR can be effectively combined with an offline RL algorithm to learn policies from a large dataset of unlabeled episodes and a small number of high-quality demonstrations. Since OTR is only a method for reward learning, it can be combined with any offline RL algorithm that requires reward-annotated data for offline learning. In this chapter, we combine OTR with the Implicit Q-Learning (IQL) algorithm [81]².

IQL consists of three learned components: a policy network π_ϕ , a state-value network V_ψ and an action-value network Q_θ . For policy evaluation, the state-value network is learned by minimizing

$$L_V(\psi) = \mathbb{E}_{(s,a) \sim \mathcal{D}} [L_2^\tau(Q_{\theta'}(s,a) - V_\psi(s))], \quad (4.6)$$

where θ' denotes the parameters for the target action-value network and $L_2^\tau(u) = |\tau - \mathbf{1}(u)|u^2$ is the expectile regression objective for the τ -expectile. The action-value network Q_θ is learned by minimizing

$$L_Q(\theta) = \mathbb{E}_{(s,a,s') \sim \mathcal{D}} \left[(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a))^2 \right], \quad (4.7)$$

where $r(s,a)$ is the reward for state s and action a . To use OTR in with IQL, the reward function r will be substituted with the OTR reward function r_{OTR} .

For policy improvement, IQL uses the advantage weighted regression objec-

²Our IQL implementation is adapted from https://github.com/ikostrikov/implicit_q_learning.

tive [56, 67]

$$L_{\pi}(\phi) = \mathbb{E}_{(s,a) \sim \mathcal{D}} [\exp(\beta(Q'_{\theta}(s,a) - V_{\psi}(s))) \log \pi_{\phi}(a|s)], \quad (4.8)$$

where $\beta \in [0, \infty)$ is a temperature parameter, to extract an offline policy.

4.4.1 Experimental setup

We evaluate OTR+IQL on the D4RL benchmark [63]. We start by evaluating OTR on three environments (HalfCheetah-v2, Walker2D-v2 and Hopper-v2) from the OpenAI Gym MuJoCo locomotion tasks. For each environment, we use the medium-v2, medium-replay-v2, and medium-expert-v2 datasets to construct the expert demonstrations and the unlabeled dataset. For the expert demonstrations, we choose the best episodes from the D4RL dataset based on the episodic return. In practice, the expert demonstrations can be provided separately; we only select the expert demonstration in this way for ease of evaluation. To obtain the unlabeled dataset, we discard the original reward information in the dataset. We then run OTR to label the dataset based on the optimal coupling between the unlabeled episodes and the chosen expert demonstrations. Afterward, we proceed with running the offline RL algorithm.

4.4.2 Experimental results

Baselines

We compare OTR+IQL with the following baselines:

BC We consider two variants of BC: the first uses the full dataset while the second uses the top 10% of the episodes sorted by episodic return. The results of the BC baselines are obtained from [63].

IQL (oracle) this is the original Implicit Q-learning [81] algorithm using the ground-truth rewards provided by the D4RL datasets.

DemoDICE an offline imitation learning algorithm proposed by [80]. DemoDICE was found to perform better than previous imitation learning

algorithms (e.g., ValueDICE [81]) that can also (in principle) work in the pure offline setting. We ran the original implementation³ under the same experimental setting as we used for the other algorithms in the paper.

ORIL a reward function learning algorithm from [57]. ORIL learns a reward function by contrasting the expert demonstrations with the unlabeled episodes. The learned reward function is then used to label the unlabeled dataset. We implement ORIL in a comparable setting to the other baselines.

UDS we keep the rewards from the expert demonstrations and relabel all rewards from the unlabeled datasets with the minimum rewards from the environment. This was found to perform well in [93]. This also resembles online imitation learning methods such as SQIL [53] or offline RL algorithms, such as COG [71].

Since UDS and ORIL are also agnostic about the underlying offline RL algorithm used, we combined these algorithms with IQL so that we can focus on comparing the performance differences due to different approaches used in generating reward labels.

For all algorithms, we repeat experiments with 10 random seeds and report the mean and standard deviation of the normalized performance of the last 10 episodes of evaluation. We compare all algorithms by using either $K = 1$ or $K = 10$ expert demonstrations. Obtaining the results on the locomotion datasets took approximately 500 GPU hours.

MuJoCo Locomotion. We compare OTR+IQL’s performance with all baselines discussed above on the locomotion tasks in the OpenAI Gym environments. Table 4.1 compares the performance between OTR+IQL with the other baselines. Overall, OTR+IQL performs best compared with the other baselines in terms of aggregate score over all of the datasets we used, recovering the performance of IQL with ground-truth rewards provided by the dataset. While we found that other baselines can perform well on some datasets, the performance is not consistent across the entire dataset and can deteriorate significantly on some

³<https://github.com/geon-hyeong/imitation-dice>

Table 4.1: D4RL performance comparison between IQL with ground-truth rewards and OTR+IQL with a single expert demonstration ($K = 1$). We report mean \pm standard deviation per task and aggregate performance and highlight near-optimal performance in **bold** and extreme negative outliers in **red**. OTR+IQL is the only algorithm that performs consistently well across all domains.

Dataset	BC	10%BC	IQL (oracle)	DemoDICE	IQL+ORIL	IQL+UDS	OTR+IQL
halfcheetah-medium-v2	42.6	42.5	47.4 \pm 0.2	42.5 \pm 1.7	49.0 \pm 0.2	42.4 \pm 0.3	43.3 \pm 0.2
hopper-medium-v2	52.9	56.9	66.2 \pm 5.7	55.1 \pm 3.3	47.0 \pm 4.0	54.5 \pm 3.0	78.7\pm5.5
walker2d-medium-v2	75.3	75.0	78.3 \pm 8.7	73.4 \pm 2.6	61.9 \pm 6.6	68.9 \pm 6.2	79.4\pm1.4
halfcheetah-medium-replay-v2	36.6	40.6	44.2 \pm 1.2	38.1 \pm 2.7	44.1\pm0.6	37.9 \pm 2.4	41.3 \pm 0.6
hopper-medium-replay-v2	18.1	75.9	94.7 \pm 8.6	39.0\pm15.4	82.4 \pm 1.7	49.3\pm22.7	84.8\pm2.6
walker2d-medium-replay-v2	26.0	62.5	73.8 \pm 7.1	52.2 \pm 13.1	76.3\pm4.9	17.7\pm9.6	66.0 \pm 6.7
halfcheetah-medium-expert-v2	55.2	92.9	86.7 \pm 5.3	85.8\pm5.7	87.5\pm3.9	63.0 \pm 5.7	89.6\pm3.0
hopper-medium-expert-v2	52.5	110.9	91.5 \pm 14.3	92.3\pm14.2	29.7\pm22.2	53.9\pm2.5	93.2\pm20.6
walker2d-medium-expert-v2	107.5	109.0	109.6 \pm 1.0	106.9\pm1.9	110.6\pm0.6	107.5\pm1.7	109.3\pm0.8
locomotion-v2-total	466.7	666.2	692.4	585.3	588.5	494.9	685.5
runtime	10m	10m	20m	100m*	30m	20m	22m

* The runtime is measured with the original PyTorch implementation.

datasets. In contrast, OTR+IQL is the only method that consistently performs well for all datasets of different compositions.

Runtime. Despite applying optimal transport, we found that with a GPU-accelerated Sinkhorn solver [75], combined with our efficient implementation in JAX, OTR achieves a faster runtime compared to algorithms that learn additional neural networks as discriminators (DemoDICE [80]) or reward models (ORIL [57]). For methods that learn a neural network reward function, an overhead of at least 10 minutes is incurred, whereas OTR only incurs approximately 2 minutes of overheads when compared with the same amount of expert demonstrations.

Effect of the number of demonstrations. We investigate if the performance of the baselines can be improved by increasing the number of expert demonstrations used. Table 4.2 compares the aggregate performance on the locomotion datasets between OTR and the baselines when we increase the number of demonstrations from $K = 1$ to $K = 10$. DemoDICE’s performance improves little with the additional amount of expert demonstrates. While ORIL and UDS enjoy a relatively larger improvement, they are still unable to match the performance of IQL (oracle) or OTR in terms of aggregate performance despite using the same

Table 4.2: Aggregate performances of different reward labeling algorithms with different numbers of expert demonstrations. OTR is the only algorithm that leads to an offline RL performance close to using ground-truth rewards.

locomotion-v2-total	$K = 1$	$K = 10$
DemoDICE	585.3	589.3
IQL+ORIL	588.5	618.3
IQL+UDS	494.9	575.8
OTR+IQL	685.5	694.3
IQL (oracle)		692.4

Table 4.3: Performance of OTR+IQL on AntMaze with a single expert demonstration. Similar to the results on locomotion, OTR+IQL recovers the performance of offline RL with ground-truth rewards. We report mean \pm standard deviation per task and aggregate performance.

Dataset	IQL (oracle)	OTR+IQL
antmaze-large-diverse-v0	47.5 \pm 9.5	45.5 \pm 6.2
antmaze-large-play-v0	39.6 \pm 5.8	45.3 \pm 6.9
antmaze-medium-diverse-v0	70.0 \pm 10.9	70.4 \pm 4.8
antmaze-medium-play-v0	71.2 \pm 7.3	70.5 \pm 6.6
antmaze-umaze-diverse-v0	62.2 \pm 13.8	68.9 \pm 13.6
antmaze-umaze-v0	87.5 \pm 2.6	83.4 \pm 3.3
antmaze-v0-total	378.0	384.0

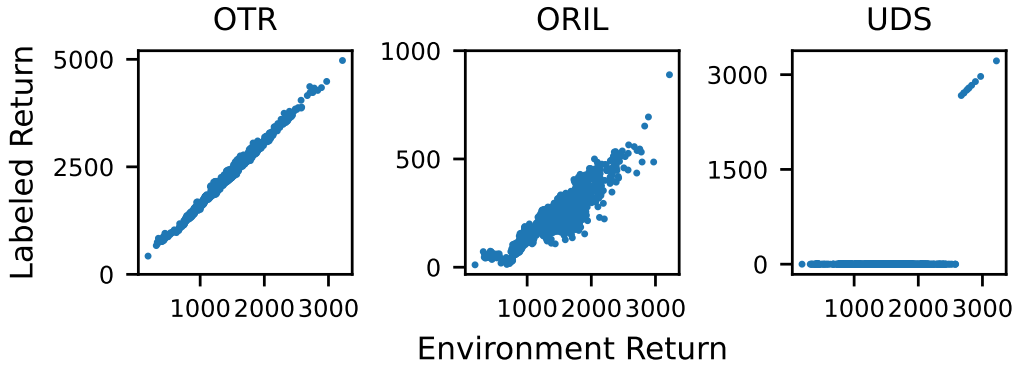
offline RL backbone. OTR’s performance is close to IQL (oracle) even when $K = 1$ and matches the performance of IQL (oracle) with $K = 10$.

AntMaze and Adroit. We additionally evaluate OTR+IQL on datasets from antmaze-v0 and adroit-v0. Tables 4.3 and 4.4 shows that OTR+IQL again recovers the performance of IQL with ground-truth rewards. This suggests that OTR+IQL can learn from datasets with diverse behavior and human demonstrations even without ground-truth reward annotation.

Qualitative comparison of the reward predictions. Figure 4.4 provides a qualitative comparison of the reward predicted by OTR, ORIL, and UDS. UDS annotates all transitions in the unlabeled dataset with the minimum reward

Table 4.4: Performance of OTR+IQL on Adroit with a single expert demonstration.

Dataset	IQL (oracle)	OTR+IQL
door-cloned-v0	1.60	0.01 ± 0.01
door-human-v0	4.30	5.92 ± 2.72
hammer-cloned-v0	2.10	0.88 ± 0.30
hammer-human-v0	1.40	1.79 ± 1.43
pen-cloned-v0	37.30	46.87 ± 20.85
pen-human-v0	71.50	66.82 ± 21.18
relocate-cloned-v0	-0.20	-0.24 ± 0.03
relocate-human-v0	0.10	0.11 ± 0.10
adroit-v0-total	118.1	122.16

**Figure 4.4:** Qualitative differences between the rewards predicted by OTR, ORIL, and UDS on hopper-medium-v2. We found that the rewards computed by OTR is more correlated with the crafted reward function used for training the expert policy compared to ORIL and UDS. We find OTR predicts higher rewards for trajectories that more closely resemble the expert demonstrations.

from the environment. Thus, the episodes with non-zero rewards are expert demonstrations. This means that UDS is unable to distinguish between episodes in the unlabeled datasets. Compared to reward learning algorithms, such as ORIL, OTR’s reward prediction more strongly correlates with the ground-truth rewards from the environment, which is a good precondition for successful policy learning by downstream offline RL algorithm. We also evaluate OTR’s reward prediction on more diverse datasets, such as those in AntMaze. Figure 4.5 shows the expert demonstrations we used in antmaze-medium-play-v0 (left) and the trajectories that received the best OTR reward labels in the unlabeled dataset

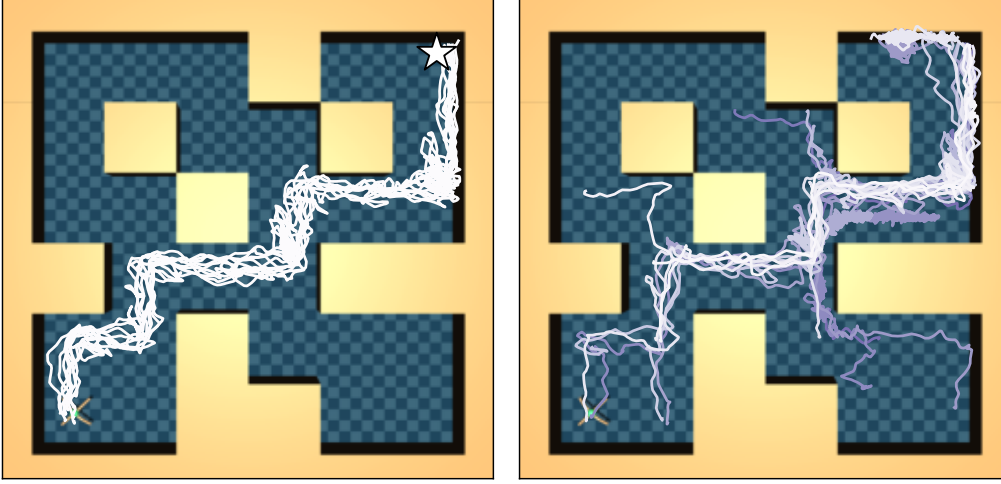


Figure 4.5: Ranking of trajectories according to rewards per step computed by OTR. Trajectories with lighter colors have higher rewards per step.

(right). OTR correctly assigns higher rewards to trajectories that are closer to the expert demonstrations. The qualitative result in fig. 4.5 also provides some clue for why using an approach such as OTR may be more favorable even when a reward function is readily available. The original reward function used in the AntMaze dataset assigns a reward of one when the ant is sufficiently close to the target location and zero otherwise. OTR, on the other hand, provides a mechanism for reward shaping [8] for every action made by the agent which may make the learning the optimal policy easier.

Combining OTR with different offline RL algorithms. In the main experiments, we evaluated OTR by pairing it with the IQL algorithm. In this section, we investigate if OTR can recover the performance of a different offline RL algorithm (TD3-BC) [64] using ground-truth rewards. We replace IQL with TD3-BC and report the results in table 4.5. We observe that (i) the performance from OTR+TD3-BC mostly matches those using the ground-truth rewards; (ii) the performance is fairly robust in terms of the choice of the number of expert trajectories. However, there are more variances on some datasets (e.g., halfcheetah-medium-expert-v2). Nevertheless, the differences are smaller compared to the baselines, and OTR+TD3-BC still performs better than the baselines presented in section 4.4 in terms of aggregate performance.

Table 4.5: OTR+TD3-BC results on MuJoCo. We consider using a different offline RL algorithm TD3-BC [64] as our RL backbone. We find that using the TD3-BC backbone performs comparably, demonstrating that OTR is not sensitive to the RL backbone used.

Dataset	TD3-BC (oracle)	OTR+TD3-BC	
		$K = 1$	$K = 10$
halfcheetah-medium-expert-v2	93.5 \pm 2.0	74.8 \pm 20.1	71.6 \pm 23.1
halfcheetah-medium-replay-v2	44.4 \pm 0.8	39.4 \pm 1.3	38.9 \pm 1.5
halfcheetah-medium-v2	48.0 \pm 0.7	42.6 \pm 1.0	42.7 \pm 1.1
hopper-medium-expert-v2	100.2 \pm 20.0	103.2 \pm 13.9	98.9 \pm 19.7
hopper-medium-replay-v2	64.8 \pm 25.5	74.9 \pm 28.8	80.2 \pm 23.1
hopper-medium-v2	60.7 \pm 12.5	66.4 \pm 10.3	69.8 \pm 13.9
walker2d-medium-expert-v2	109.5 \pm 0.5	109.0 \pm 0.6	108.8 \pm 0.8
walker2d-medium-replay-v2	87.4 \pm 8.4	69.7 \pm 16.4	67.4 \pm 20.6
walker2d-medium-v2	83.7 \pm 5.3	76.9 \pm 5.4	78.0 \pm 2.6
locomotion-v2-total	692.3	656.9	656.4

Importance of using the optimal transport plan. In the main experiments, we compute the rewards based on the optimal coupling computed by the Sinkhorn solver. The optimal transport plan is sparse and transports most of the probability masses to only a few expert samples. In this section, we investigate what happens if we use a suboptimal transport plan where each sample from the policy’s trajectory is transported equally to each sample in the expert’s trajectory. In this case, the reward function essentially boils down to computing the average costs with respect to all of the states in the expert’s trajectory.

Table 4.6 compares the performance of OTR+IQL using the optimal transport plan and uniform transport plan. We find that for many datasets, using the suboptimal uniform transport plan is sufficient for reaching good performance.

This indicates that using a reward function based on the similarity of states from the policy and the expert can be a simple and effective method for reward labeling. However, note that the uniform transport plan can still underperform compared to using the optimal transport plan (e.g., hopper-medium-replay-v2). This shows that the optimal transport formulation enables better and more consistent performance.

Table 4.6: OTR with uniform transport plan. We compare using an optimal transport plan with a uniform transport plan. Using an optimal transport plan allows us to obtain better performance.

Dataset	Optimal transport	Uniform
halfcheetah-medium-v2	43.3 ± 0.2	43.5 ± 0.3
hopper-medium-v2	78.7 ± 5.5	80.5 ± 2.3
walker2d-medium-v2	79.4 ± 1.4	77.6 ± 1.5
halfcheetah-medium-replay-v2	41.3 ± 0.6	41.6 ± 0.8
hopper-medium-replay-v2	84.8 ± 2.6	69.8 ± 10.1
walker2d-medium-replay-v2	66.0 ± 6.7	62.2 ± 14.4
halfcheetah-medium-expert-v2	89.6 ± 3.0	90.6 ± 2.9
hopper-medium-expert-v2	93.2 ± 20.6	89.2 ± 14.0
walker2d-medium-expert-v2	109.3 ± 0.8	106.0 ± 5.9

Comparison to PWIL. In this section, we investigate if the online imitation learning algorithm PWIL [60] can be used in the offline setting with a change from using an online RL algorithm to an offline RL algorithm. We ran PWIL with IQL, which is similar to what we did for OTR in the main paper. We use the PWIL implementation from Acme [79].⁴

Note that although OTR is similar to PWIL in using the Wasserstein distance to construct RL reward signals, OTR differs from PWIL in the choices of OT solver, the cost function as well as the approach used for aggregating results from multiple expert demonstrations. Also note that for all experiments, we consider learning only from expert states instead of state-action pairs. This is both a more general and challenging setting. It was found in [60] that PWIL sometimes performs badly without expert actions. We ran OTR and PWIL using only expert observations and OTR and PWIL using observation-action pairs. The results are illustrated in table 4.7. We found that we are unable to get good results when running PWIL using only expert state sequences. This is possibly due to the different choices of OT solvers and cost functions. PWIL can perform well when combined with IQL to learn in the offline setting, although sometimes performance is significantly worse compared to the IQL oracle or OTR (e.g., `hopper-medium-expert-v2`).

⁴<https://github.com/deepmind/acme/tree/master/acme/agents/jax/pwil>

Dataset	Observation only		Observation and action	
	OTR	PWIL	OTR	PWIL
halfcheetah-medium-v2	43.1 \pm 0.3	1.6 \pm 1.2	43.4 \pm 0.3	47.5 \pm 0.2
hopper-medium-v2	80.0 \pm 5.2	2.1 \pm 1.3	75.4 \pm 4.6	70.4 \pm 4.2
walker2d-medium-v2	79.2 \pm 1.3	0.9 \pm 1.3	79.7 \pm 1.2	81.9 \pm 1.0
halfcheetah-medium-replay-v2	41.6 \pm 0.3	-2.3 \pm 0.5	41.9 \pm 0.3	44.6 \pm 1.1
hopper-medium-replay-v2	84.4 \pm 1.8	1.4 \pm 1.2	85.3 \pm 1.1	89.7 \pm 4.9
walker2d-medium-replay-v2	71.8 \pm 3.8	-0.1 \pm 0.2	69.1 \pm 4.6	72.2 \pm 10.6
halfcheetah-medium-expert-v2	87.9 \pm 3.4	-0.3 \pm 1.5	88.3 \pm 5.1	88.6 \pm 4.3
hopper-medium-expert-v2	96.6 \pm 21.5	1.5 \pm 0.6	86.6 \pm 22.9	32.9 \pm 25.0
walker2d-medium-expert-v2	109.6 \pm 0.5	1.0 \pm 1.9	109.2 \pm 0.5	110.2 \pm 0.2

Table 4.7: Comparison between OTR and PWIL with IQL as the offline RL backbone. We found PWIL to underperform OTR when actions are not used to compute rewards.

Hyper-parameter sensitivity. For the main results, the hyper-parameters for the squashing function (α and β) were chosen to be consistent with those used in [60]. In this section, we compare the differences in the choices of these hyper-parameters by running OTR with $\alpha = \beta = 1$. This reduces to simply applying an exponential transformation to the optimal transport costs. The results are illustrated in table 4.8. We find that OTR still performs well, demonstrating that

Dataset	$\alpha = \beta = 5$	$\alpha = \beta = 1$
halfcheetah-medium-expert-v2	87.9 \pm 3.4	86.9 \pm 4.0
halfcheetah-medium-replay-v2	41.6 \pm 0.3	40.4 \pm 1.3
halfcheetah-medium-v2	43.1 \pm 0.3	42.7 \pm 0.4
hopper-medium-expert-v2	96.6 \pm 21.5	82.6 \pm 9.9
hopper-medium-replay-v2	84.4 \pm 1.8	71.2 \pm 15.2
hopper-medium-v2	80.0 \pm 5.2	75.7 \pm 6.4
walker2d-medium-expert-v2	109.6 \pm 0.5	106.3 \pm 8.2
walker2d-medium-replay-v2	71.8 \pm 3.8	63.2 \pm 5.7
walker2d-medium-v2	79.2 \pm 1.3	77.4 \pm 1.5

Table 4.8: Effect of α and β in the squashing function. We study the effect of the reward squashing hyper-parameters on final performance. OTR performs consistently with a different set of hyper-parameters.

it is not sensitive to the choices of these hyper-parameters.

Summary. Our empirical evaluation demonstrates that OTR+IQL can recover the performance of offline RL with ground-truth rewards given only a single episode of expert demonstration. Compared to previous work, it achieves better performance even with a larger number of demonstrations. Our qualitative comparison shows that OTR assigns better reward estimates that correlate strongly with a hand-crafted reward function.

4.5 Discussion

OTR can be effective in augmenting unlabeled datasets with rewards for use by downstream offline RL algorithms (see section 4.4). Compared to prior imitation learning and reward learning approaches, OTR enjoys better and more robust performance on the benchmarks we have considered in this paper. Furthermore, our approach is easy to implement. Most of the complexity of our approach arises from solving the optimal transport problem. However, there are libraries for solving OT efficiently in different frameworks or programming languages.⁵

Unlike prior works, such as ValueDICE or DemoDICE, we split offline imitation learning into the two distinct phases of reward modeling via OTR and a subsequent offline RL. This improves modularity, allows for improvements in the two phases to be made independently, and adds flexibility. However, this modularity means that a practitioner needs to make a separate choice for the downstream offline RL algorithm.

We believe that OTR is most useful in situations where providing ground-truth rewards is difficult, but providing good demonstrations and unlabeled data is feasible. However, it will not be feasible in cases where collecting expert demonstrations is difficult. It would be interesting to explore in future work to relax the assumption of having complete expert demonstrations by requiring only partial demonstrations or key frames from the experts.

The Wasserstein distance formulation we used can be extended to perform cross-domain imitation learning by using the Gromov-Wasserstein distance to align expert demonstrations and offline trajectories from different spaces [77].

⁵For example, see Python Optimal Transport (POT) [62], which supports PyTorch, JAX, or TensorFlow.

4.6 Conclusion

We introduced Optimal Transport Reward labeling (OTR), a method for adding reward labels to an offline dataset, given one or more expert demonstrations. OTR computes Wasserstein distances between expert demonstrations and trajectories in a dataset without reward labels, which are then turned into a reward signal. An offline RL algorithm can then use the reward-annotated offline dataset to determine good policies. OTR adds minimal overhead to existing offline RL algorithms while providing the same performance as learning with a pre-specified reward function.

Chapter 5

Bridging policy learning and language modeling

5.1 Overview

In this chapter, we take a step forward by addressing how agents can learn from imperfect data that lacks an episodic structure and originates from diverse, less curated sources.

In the previous chapters, we explored reinforcement learning (RL) and imitation learning methods that rely on imperfect data, such as sub-optimal behavior or incomplete datasets, but with a well-defined structure. These data sources, while imperfect, often retain clear episodic transitions and are focused on learning specific tasks. For example, in chapter 3, we considered datasets with a clear episodic structure consisting of sequences of state, action, and rewards. In chapter 4, we addressed situations where some datasets lack reward annotation and considered inferring a reward function to overcome this limitation. In both cases, the focus was on learning behaviors for single, well-defined tasks, such as controlling a cheetah in MuJoCo.

While these approaches are practical for their respective domains, a significant drawback of relying on structured datasets and single-task learning is the limited scope of the agent’s capabilities. By focusing on predefined tasks with well-structured reward signals, agents are often trained to optimize narrow, specific behaviors. This rigidity hinders their ability to develop a broader range of

skills that could be useful in more general or unpredictable environments. Agents trained under these methods may perform well in specialized domains but often struggle to generalize to new tasks or to handle situations not explicitly covered by the training data.

In contrast, real-world environments rarely provide clearly structured data, and optimal behaviors are often difficult to define or identify. To enable agents to succeed in these settings, moving beyond rigid, task-specific datasets and exploring methods that allow learning from diverse, unstructured experiences is necessary.

The ideal autonomous agent should be capable of integrating information from various sources — not only structured behavior data but also less formal, unstructured data, such as web-based wikis, manuals, or human discussions. This broader range of data sources would enable agents to acquire various skills, making them more adaptable and better equipped to handle various tasks and environments. Essentially, these agents should evolve into “jacks of all trades” instead of being restricted by the narrow, single-task focus that structured datasets often impose.

This chapter explores how to create such versatile agents by attempting to move beyond traditional RL frameworks, which rely heavily on predefined, structured data. While human decision-making typically draws from two primary sources – historical behaviors (replay of past experiences) and analytical insights (natural language and strategic reasoning) – most ML research has treated the two pieces of information separately, focusing on either behavior or language modeling. However, recent advancements in large language models (LLMs) [31], which have demonstrated remarkable capabilities beyond language modeling, offer the potential to integrate these two pieces of information. Specifically, these models have shown promise in embedding common knowledge into decision-making systems [58, 65, 73], suggesting that structured and unstructured data can be combined to develop agents capable of more complex, adaptive behavior.

In this chapter, we study the behavior of a unified system that integrates natural language understanding with policy learning to create agents that can learn from diverse, non-episodic data sources. This approach is more flexible and capable of generalizing beyond what is possible with structured data alone.

To explore these ideas, we use chess as a practical testbed. It is an ideal domain for studying how agents can learn from episodic and non-episodic data. In addition to the vast amount of game replay data available, there are rich language-based resources, such as game analyses and strategy discussions on chess. The abundance of structured and unstructured data in chess creates an opportunity to investigate how diverse, unstructured data can complement traditional RL methods.

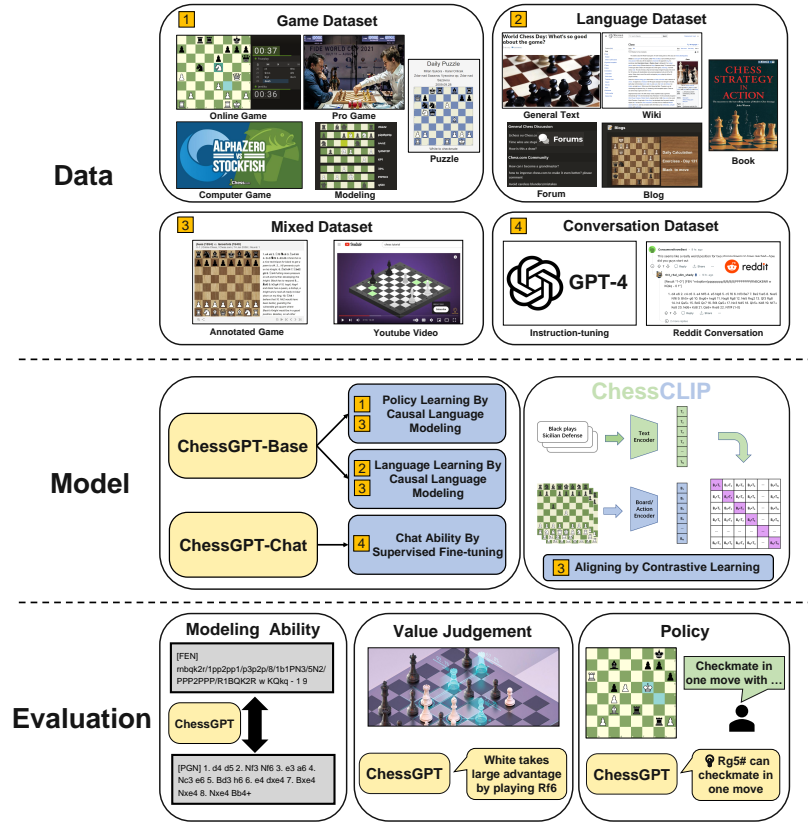


Figure 5.1: Our work provides a comprehensive pipeline that encompasses data, model, and evaluation framework, aiming to foster research on the interaction between policy learning and language learning. **Data:** Our dataset is organized into four categories: game, language, mixed, and conversation datasets. **Model:** Leveraging this rich dataset, we present two models: ChessGPT and ChessCLIP. **Evaluation:** Our evaluation framework is structured around three key dimensions: modeling capability, value judgment, and policy proficiency.

Leveraging these resources, we have constructed a comprehensive pipeline to integrate policy learning and language modeling in chess. Our contributions

include:

- **Datasets:** We compile a large-scale dataset with structured game data and unstructured language resources, such as chess articles, commentaries, and tutorials. We also introduce a mixed dataset that combines structured and unstructured elements.
- **Models:** We present two models, ChessCLIP and ChessGPT, that leverage these datasets to learn from structured replay data and unstructured language knowledge.
- **Evaluations:** Our evaluation framework assesses the models across three key dimensions: modeling capability, value judgment, and policy proficiency. Experimental results show that incorporating non-episodic and unstructured data enhances performance, enabling the models to outperform existing LLM baselines.

Figure 5.1 illustrates the entire pipeline. We demonstrate that incorporating diverse behavioral and knowledge-based data enables the creation of autonomous agents that use language as a tool for both action and understanding. Moreover, our effort results in a valuable by-product: the ChessGPT and ChessCLIP models. These models have practical applications and could be effective AI chess assistants for humans.

5.2 Dataset

We introduce a large-scale game and language dataset by collecting online chess-related materials. Our dataset consists of four categories: (1) The game dataset encompasses online chess match replay data involving worldwide human players and chess engines of varying skill levels. (2) The language dataset records chess-associated knowledge, analyses, discussions, and news in the form of natural language (3) The mixed game-language dataset provides aligned pairs of game data and human natural language (such as game analysis or comments). (4) The

instruction-tuning and conversation dataset consists of instruction and conversation data related to chess. We include comprehensive dataset descriptions, statistics, pre-processing details, and examples in appendix C.2.

5.2.1 Game dataset

Game replay data provide the most direct method for both humans and machines to grasp the play mechanics of chess. In chess, records of game plays are commonly stored in the Portable Game Notation (PGN).¹ which is a standard plain text format as illustrated in fig. 5.2. A PGN starts with some headers that include metadata about the game. These headers include information such as the names of players, the Elo ratings, the opening play, and the game outcome. The headers are followed by a move text section that records the moves played by the two players. The moves may be further annotated with comments enclosed in braces.

McIlroy-Young et al. [48] uses the moves recorded in PGNs for policy learning. The moves are interpreted as actions in a Markov decision process, and the state position can be reconstructed by loading the PGN into a chess engine. However, PGNs may contain additional helpful information beyond the individual moves made. For example, the Elo ratings in the headers may inform us about the relative strength of the players. Additional information included in the comments of the move text section can also be helpful — some of the moves are annotated with evaluations generated by computer chess programs that predict the current advantage of the players. For example, a RL algorithm may use this information for value function learning. For this reason, we curated the game dataset with all of this information intact to facilitate policy learning.

Lichess dataset. We collect five months of online game data from the Lichess database [125], culminating in 17.5 million game replay records for online game players.

Pro-player dataset. In the Lichess dataset, most player Elo ratings range between 1000 and 2000. To diversify our game dataset with more skilled matches,

¹We refer to appendix C.1 for readers who are not familiar with chess notation format.

```

[White "Alice"]
[Black "Bob"]
[Result "0-1"]
[WhiteElo "2100"]
[BlackElo "2000"]
[Opening "Sicilian Defense"]

1. e4 { [%eval 0.1] } 1... c5
2. Nf3 2... Nc6
...
13. b3?? 13... Nf4? 0-1

```

Figure 5.2: Replay example in Portable Game Notation (PGN) format.

we also incorporated an additional 440,000 game records from 245 professional chess players. These professionals typically hold notably higher Elo ratings between 2000 and 2800.

CCRL. Chess engines like StockFish and LeelaZero have attained a proficiency level far beyond what any human player can reach. Considering this, we additionally incorporate the *Computer Chess Rating Lists* (CCRL) [128], which is a dataset of chess games played by computer chess engines. The CCRL dataset comprises a considerable collection of chess games, specifically 3 million, all played by computer chess engines and stored in PGN format. The chess engines’ Elo ratings range between 2800 and 3700.

Chess puzzles. A chess puzzle represents a particular chessboard configuration designed to present a distinct challenge or objective for the solver. Chess puzzles often require players to find the best move or sequence to achieve a specific goal, such as checkmating the opponent’s king or finding a tactical combination. We integrate 3.2M puzzles sourced from the Lichess puzzle dataset in our game dataset. Each puzzle within this collection is annotated with its rating, theme, and solution.

Chess modeling dataset. We observe that most chess rule descriptions are conveyed in natural language, posing a challenge for machine learning models

since they statistically require a large volume of model data to comprehend the chess rules [54] accurately. We build a synthetic chess modeling dataset leveraging the python-chess library [130] to address this issue. We collect chess game data from a one-month dump of the Lichess dataset, deliberately distinct from the month used in our own Lichess dataset. We designed several model-based tasks, including converting PGN to FEN, transferring from UCI to FEN, and predicting legal moves, resulting in 1.9M samples.

5.2.2 Language dataset

Existing dataset. Numerous existing datasets comprise general Internet crawl data from platforms like CommonCrawl or Wikipedia. We establish a filtering pipeline to extract only chess-related language corpus from pre-existing language corpora, including C4 [52], the Pile [44], Oscar [40], Wikipedia [124] and RedPajama [110]. These datasets extend the scope of our language data beyond mere game-play.

Chess blogs. Chess websites often publish insightful blogs, sharing their analyses and perspectives on various aspects of chess game-play. Such blog data is valuable, as it encompasses game-specific analysis, forming a vital link between the concrete chess game data and its interpretation in natural language form. We manually curate around 30 chess-related websites and scrape 73.2k blog articles.

Chess books. Like chess blogs, chess books can provide long and detailed game analysis. We extract approximately 8K chess-related books from online libraries to enrich our language dataset.

Chess forums. Chess forums serve as a platform for many chess-related dialogues and conversations involving diverse users. These platforms encompass high-quality question-and-answer pairs, as seen in platforms like Stack Exchange or more generalized discussions on various chess-related topics commonly found in dedicated forums. We scrape chess forum data from five chess-specific forum platforms and Stack Exchange. This process results in a collection of 140K posts representing diverse views, queries, and discourses related to the world of chess.

5.2.3 Mixed game-language dataset

Annotated chess game. An annotated chess game is a chess play accompanied by written commentary and analysis. In an annotated game, each move made by the players is accompanied by explanations and evaluations that provide insights into the thought process, strategic considerations, and tactical ideas behind the moves. Here is an example of an annotated PGN with a Sicilian Defense opening: *1.e4 c5 {The game starts with the Sicilian Defense, one of the most popular and aggressive responses to 1.e4. Black aims to control the center and create imbalances early on.}*

These annotated games inherently maintain the correspondence between board state and human language, serving as an exceptionally high-quality data source to align a model with complex human intentions and judgments. We amass annotated games from seven sources, five of which are collected from the Internet, while the other two are commercial datasets. We collect 245K annotated games with 1.3M board-language pairs.

YouTube transcripts. Drawing inspiration from the MineDoJo dataset [76], a YouTube video can serve as a mixed game-language dataset by aligning video clips with natural language transcripts based on timestamps. Rather than generating image-language pairs directly, we develop a pipeline that accurately applies Optical Character Recognition (OCR) to chessboard screenshots to generate the Forsyth-Edwards Notation (FEN): a system that describes the chess state in a text format. We gathered around 83k chess videos, resulting in million-scale English transcripts and board-language pairs, thus establishing a substantial mixed game-language dataset.

5.2.4 Instruction-tuning and conversation dataset

Supervised fine-tuning is crucial in training large language models (LLM) to follow instructions [89, 88, 92]. In addition to the comprehensive chess materials mentioned before, we collect instruction-tuning and conversation datasets, which can be used to fine-tune the pre-trained LLM base model, enhancing its instruction-following and dialogue capability.

Instruction-tuning data from GPT-4. Inspired by Stanford Alpaca [109], we use the self-instruct technique [113] to generate high-quality, instruction-following data through GPT-4 [96]. Specifically, we manually construct 200 seed prompts for chess-related questions or instructions. These prompts serve as few-shot examples, guiding GPT-4 towards a more coherent and relevant generation. Finally, we generate around 4K instruction-response pairs using this pipeline.

Conversation data from Reddit. The instruction data collected from GPT-4 are mainly in a single-step form, including only one question-answer round. We collect multi-step conversation data about chess from Reddit to mitigate this issue. Reddit allows users to interact by commenting on posts and responding to other comments, creating a nested structure of responses. We convert the nested structure into a conversation tree by treating the comment’s reply as a child node for that reply. We then acquire a rich source of conversation data by navigating from the root node to each leaf node via every available path. We chose six chess-related subreddits and collected 410K human conversations about chess.

5.3 Models

We showcase two models, ChessCLIP and ChessGPT, trained on the large-scale dataset.

5.3.1 ChessCLIP

Contrastive Language-Image Pretraining (CLIP) [70] is a neural network trained on a variety of modalities (e.g., image, text). By conducting contrastive learning on a large amount of paired data, CLIP bridges the image and language modality, enabling the model to understand vision by language information and vice versa. Our mixed game-language dataset from section 5.2.3 provides paired game trajectories and annotation examples. Using this subset, we train a **ChessCLIP** to bridge the modality of policy and language. Specifically, by denoting the chessboard state S at timestep t as S_t , and the annotation language as L_t , the data

pair at timestep T can be represented by $((\{S_t\}_{t=T-k}^T, a_T), L_T)$ where $\{S_t\}_{t=T-k}^T$ is a stacked k history states and a_T is the last move.

We want to emphasize what ChessCLIP can do by aligning the policy and language modalities. Firstly, ChessCLIP offers a similarity metric given one PGN and a text description. Just like the application of large-scale image/text retrieval using CLIP, ChessCLIP can help users conduct PGN/text retrieval – search for games based on text or search for comments based on specific games. In addition, because of the low-dimensional feature of action space compared to vision or language space (there only exists a few legal moves for a given chess state), we can directly conduct search algorithms to maximize the similarity to generate action based on one text description using ChessCLIP. For example, given a chessboard state and a text description, ChessCLIP can generate a move by iterating through all legal moves and finding one most similar to the text encoding. Similarly, ChessCLIP can also generate move sequences (multiple actions) using greedy search or beam search. However, the ChessCLIP model does not have the capability to free-form text or PGN generation capabilities built-in. As a result, it cannot be used for all chess-related tasks as easily as compared to the ChessGPT model we describe below. We refer the reader to appendix C.3.1 for more discussions.

Implementation details. We preprocess the annotated PGNs to produce board-/text pairs, which we feed separately to the board and text encoders. In particular, for every move in the PGN, we extract the comments attached to the move and the board state. We encode the board positions and moves using the same scheme as in Leela Chess Zero (lc0) [127], which is similar to the encoding used by AlphaZero [36] for encoding positions and moves in chess. Concretely, the board positions are encoded as a $\mathcal{R}^{8 \times 8 \times 112}$ feature map, and the actions are encoded as a \mathcal{R}^{1858} vector. We instantiate a ChessCLIP model with a text encoder and a board/action encoder. For the text encoder, we only fine-tune the last two layers of the pretrained text encoder from the OpenAI CLIP model. For the board-/action encoder, we use a ResNet [23] architecture that conditions the action encoding via a modified FiLM layer [35]. Please refer to appendix C.3.1 for implementation details.

5.3.2 ChessGPT

The Generative Pretraining Transformer (GPT-3) [43] is an auto-regressive language model that uses deep learning techniques to generate human-like text. GPT-3 is trained by casual language modeling, which aims to predict the next word in a sentence given all the previous words. Similar to language model training, we train a GPT-like model using all chess materials introduced in section 5.2. Unlike other policy behavior data in robots [25] or video games [22], the chess state and move data can be represented in pure text. Thanks to this feature, we can directly treat chess as a text game, and training with the causal language modeling objective on the dataset we collected in section 5.2.1 corresponds to imitation learning.

Implementation details. We follow typical implementations of training a domain-specific instruction-following LLM. Firstly, we conduct base-model fine-tuning using chess corpus introduced in sections 5.2.1 to 5.2.3. Due to computational constraints, we choose to fine-tune the RedPajama-3B-base [110] model, an open-source replication of LLaMA [111]. The base model adopts the GPT-NeoX [74] architecture, a GPT-3 [43] variant with a few modifications, such as rotary positional embedding, parallel attention computation, and different initialization. The base-finetuning brings us our base model: **ChessGPT-Base**. After base-finetuning, we conduct supervised fine-tuning by supervised learning on question/conversation response using data introduced in section 5.2.4 and general conversation data from OASST1 [102], Dolly 2 [98], Alpaca-GPT4 [105], and ShareGPT [131]. This brings us to our chat model: **ChessGPT-Chat**. We leave Reinforcement Learning from Human Feedback (RLHF) training as future work. Refer to Appendix C.3.1 for more details.

5.4 Evaluation

This section presents a comparative analysis between ChessGPT trained on our database and other baseline LLMs. Our experiments aim to assess the performance of ChessGPT in three primary dimensions: chess modeling, value judgment, and policy. The chess modeling capability focuses on the language

model’s proficiency in accurately tracking the game state and predicting valid moves. Regarding value judgment, we assess the model’s precision in evaluating the worth of a chess game, encompassing the identification of advantageous positions and the calculation of situation scores. Lastly, the policy assessment gauges the model’s aptitude for generating optimal moves based on a given position. By thoroughly examining these sub-categories, we can comprehensively evaluate and contrast the efficacy of different models in chess-related tasks. We choose the following models as baselines: LLaMA-7B [111], RedPajama-Base-3B [110], and compare them with ChessCLIP, ChessGPT-Base-3B, and ChessGPT-Chat-3B. More details on task examples and illustrative figures can be found in section 5.4.3.

Each evaluation task consist of three parts: Task Prefix, which can be regarded as a description of the task and is also the main prompt we use for LLMs. Input, which is the question and the input of LLMs, and Target, which contains the answer of the question for exact string match tasks, or target score, which provides the score for each available answer for multi-choice tasks.

5.4.1 Chess modeling ability

Chess state tracking

We use BIG-bench’s State Tracking in chess task [108, 91] to evaluate language models’ ability to track the state of chess games encoded in UCI notation. The task involves predicting the legal ending square given the game prefix and starting square of the current move. For example, consider the following example from this task²:

²https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/chess_state_tracking

The Chess state tracking

- **Task Prefix:** For each of the following (in-progress) chess games, please complete the notation for the last shown move by filling in the destination square.
- **Input:** e2e4 e7e6 d2d4 d7d5 e4e5 c7c5 c2c3 b8c6 g1f3 g8e7 a2a3 a7a5 f1d3 c8d7 c1e3 d8
- **Target:** ["b8", "b6", "c7", "c8"]

The example input is “e2e4 e7e6 d2d4 d7d5 e4e5 c7c5 c2c3 b8c6 g1f3 g8e7 a2a3 a7a5 f1d3 c8d7 c1e3 d8”. This leads to the chess board configuration shown in fig. 5.3. The task asks for the square that the black queen at d8 can be legally moved to. The correct response should be one of b8, b6, c7 and c8.



Figure 5.3: Example in Chess state tracking. The input leads to the following board configuration. The task is to predict the squares to which the piece at d8, the black queen, can be legally moved. Here the black queen at square d8 can be legally moved to any of the squares ["b8", "b6", "c7", "c8"].

The task dataset includes real and synthetic games, divided into short, medium, and long categories based on move count. The evaluation measures correctness across all games using a specified output regex. We note that ChessCLIP is inappropriate for modeling tasks, so we do not include it in the comparison.

Table 5.1 presents a performance analysis of all models on the task. Our Base and Chat models consistently outperform baselines in all tasks, indicating their

Table 5.1: BIG-bench State Tracking in Chess. We compare ChessGPT with other LLM baselines on the BigBench state tracking problem. ChessGPT consistently outperforms the baselines.

Tasks	LLM Models (%)			
	LLAMA-7B	RedPajama-Base	ChessGPT-Base	ChessGPT-Chat
Real Short	29.5 \pm 1.4	23.2 \pm 1.3	99.5 \pm 0.2	98.5 \pm 0.4
Real Med	39.3 \pm 1.5	38.2 \pm 1.5	97.7 \pm 0.5	97.8 \pm 0.4
Real Long	53.0 \pm 1.6	51.9 \pm 1.6	98.1 \pm 0.4	97.6 \pm 0.4
Syn Short	31.3 \pm 1.4	24.9 \pm 1.3	94.2 \pm 0.7	92.3 \pm 0.8
Syn Med	39.9 \pm 1.6	37.7 \pm 1.5	94.6 \pm 0.7	88.9 \pm 1.0
Syn Long	45.8 \pm 1.5	42.2 \pm 1.5	92.8 \pm 0.8	85.1 \pm 1.1

strong performance in tracking the state of chess games. However, the ChessGPT-Chat model exhibited slightly lower performance, suggesting a potential trade-off between language capabilities and state tracking. Nevertheless, the results underscore the effectiveness of our dataset-trained LLM models for chess state tracking.

Board state tracking.

We perform additional evaluations on UCI to FEN and PGN to FEN conversions. In the UCI to FEN experiment, the target was replaced with FEN format, while in the PGN to FEN experiment, UCI was converted to PGN format as input, and the target was replaced with the FEN format. The following are examples from the two tasks:

Board state tracking (UCI to FEN)

- **Task Prefix:** Could you produce the Forsyth-Edwards Notation (FEN) that corresponds to the provided SAN-based move list of the chess games?
- **Input:** e2e4 d7d5 b1c3 d5e4 c3e4 g8f6 e4c3 e7e6 f1c4 f8b4 d2d3 b4c3 b2c3 e8g8 g1e2 c7c5 e1g1 b8c6 e2g3 e6e5 a2a4 c8g4 f2f3 g4f5 c1g5 f5g6 d1d2 h7h6 g5h6 g7h6 d2h6 c6e7 f3f4 e5f4 f1f4 d8d6 a1f1 f6h7 g3h5 g6h5 h6d6 h5g6 d6e7 a8e8 e7h4 e8e2 h4h6 e2c2 h6g6 g8h8 c4f7 c2
- **Target:** 5r1k/pp3B1n/6Q1/2p5/P4R2/2PP4/2r3PP/5RK1 b - - 0 26

Board state tracking (PGN to FEN)

- **Task Prefix:** Could you produce the Forsyth-Edwards Notation (FEN) that corresponds to the provided PGN-based move list of the chess games?
- **Input:** 1. d4 g6 2. c4 Bg7 3. e4 Nf6 4. Nc3 O-O 5. Be3 Ne8 6. f3 Nc6 7. Qd2 e6 8. h4 d5 9. cxd5 exd5 10. Nxd5 Nf6 11. Nxf6+ Bxf6 12. e5 Bg7 13. Bb5 Bd7 14. Rc1 a6 15. Bc4 b5 16. Bb3 Bf5 17. Rxc6 Qd7 18. Rc5 c6 19. Ne2 Rad8 20. Ng3 Be6 21. Bxe6 Qxe6 22. Ne4 Rd5 23. O-O f5 24. Rxd5 cxd5 25. Nc5 Qc6 26. Bh6 f4 27. Bxg7 Kxg7 28. Re1 Qe8 29. e6 Qe7 30. Qa5 Qxh4 31. e7 Re8 32. Nxa6 Rxe7 33. Rxe7+ Qxe7 34. Nc7 Qe3+ 35. Kh2 Qf2 36. Nxd5
- **Target:** 8/6kp/6p1/Qp1N4/3P1p2/5P2/PP3qPK/8 b - - 0 36

The similarity was measured using the Levenshtein distance, which was normalized to a range of 0 to 1 [13]. These evaluations focus on assessing the model’s capability to track the chessboard’s state by representing each chess piece’s state using FEN notation.

Table 5.2 illustrates the results of these evaluations. It is evident that tracking an individual chess piece’s state becomes more challenging than tracking the entire chessboard state. On the one hand, the similarity scores between the two baselines were consistently below 10%, indicating a lack of global chess piece state tracking capability. On the other hand, the ChessGPT achieves an average similarity score higher than 90%. These results demonstrate that our model captures and reproduces the global chess piece state in UCI to FEN and PGN to FEN conversions.

Table 5.2: UCI to FEN test. We test ChessGPT’s ability to convert from UCI notation to FEN notation. ChessGPT produces more accurate conversions compared to other baselines.

		LLM Models (%)			
	Tasks	LLAMA	RedPajama-Base	ChessGPT-Base	ChessGPT-Chat
UCI to FEN	Real Short	2.2 ± 0.0	5.2 ± 0.0	95.1 ± 0.3	95.7 ± 0.1
	Real Med	2.3 ± 0.1	4.0 ± 0.1	89.9 ± 0.2	88.6 ± 0.3
	Real Long	1.8 ± 0.0	3.8 ± 0.1	85.7 ± 0.2	81.4 ± 0.5
PGN to FEN	Real Short	6.0 ± 0.1	2.8 ± 0.1	96.7 ± 0.1	95.8 ± 0.1
	Real Med	5.6 ± 0.1	3.4 ± 0.1	94.8 ± 0.1	93.6 ± 0.1
	Real Long	5.7 ± 0.0	8.9 ± 0.1	95.4 ± 0.2	82.7 ± 1.0

5.4.2 Value judgement ability

In this part, we evaluate the model’s value judgment ability. Specifically, we aim to assess the model from two perspectives: (1) its ability to align with the true value function given a chessboard state (the true values are evaluated by chess engines in enough search depths) in the evaluation of **State value multi-choice**, and (2) its ability to align with human judgment and human knowledge in the evaluation of **Chess Annotation Multi-choice** and **Opening multi-choice**.

State value multi-choice

We evaluate the model’s ability to determine which side holds the advantage of a given PGN. The following is an example from this task.

Chess state value
<ul style="list-style-type: none"> • Task Prefix: Evaluate the following PGN to see whether black or white takes advantage. • Input: 1. e4 e6 2. d4 d5 3. e5 c5 4. Nf3 cxd4 5. Nxd4 Nc6 6. Nxc6 bxc6 7. Nc3 Ne7 8. g3 Ng6 9. f4 Be7 10. Be3 h5 11. Bd3 f5 12. exf6 Bxf6 13. Bd2 Ne7 14. Qe2 Qd6 15. O-O-O Bd7 16. Kb1 Rb8 17. b3 Qa3 18. Bc1 Qa5 19. Bd2 Bxc3 20. Bxc3 Qxc3 21. Qd2 Qf6 22. Rhe1 a5 23. a4 c5 • Target Score: {"Black has advantage.": 1, "The game is equal.": 0, "White has advantage.": 0}

We construct an evaluation dataset consisting of 3000 game snippets and utilize Stockfish 15 with a search depth of 18 to calculate the winning rate for the white pieces.

We construct the state-value multiple-choice task by categorizing the winning rate into three intervals: 0 – 33% for black advantage, 34 – 66% representing a balanced state, and 67 – 100% for white advantage. During early experiments, we discovered that an additional “{” suffix to the prompt can significantly enhance the performance of the base model, likely due to “{” consistently serving as the initial symbol for annotation in annotated PGNs. Consequently, we evaluated under two distinct prompt settings and reported our results with respect to the multi-choice grade shown in table 5.3.

Table 5.3: State value multi-choice. We ask ChessGPT and ChessCLIP to predict the relative advantage of the black or white player to assess its value judgment capability. ChessGPT and ChessCLIP better predict the advantage estimated by the Stockfish computer chess engine.

Prompt Setting	Models (%)				
	LLAMA	RedPajama	ChessGPT-Base	ChessGPT-Chat	ChessCLIP
W/O { suffix	33.2 \pm 0.7	31.1 \pm 0.7	43.1 \pm 0.8	52.8 \pm 0.8	N/A
With { suffix	26.9 \pm 0.7	29.7 \pm 0.8	53.7 \pm 0.8	53.5 \pm 0.8	38.1 \pm 0.8

Table 5.4: Chess Annotation Multi-choice. We test our models’ value alignment with human annotators. Both ChessGPT and ChessCLIP outperform the baselines in assigning higher probabilities to annotations provided by human annotators.

Prompt Setting	Models (%)				
	LLAMA	RedPajama	ChessGPT-Base	ChessGPT-Chat	ChessCLIP
W/O { suffix	29.8 \pm 0.8	27.4 \pm 0.7	33.2 \pm 0.9	35.7 \pm 0.9	N/A
With { suffix	29.6 \pm 0.8	28.4 \pm 0.8	38.8 \pm 0.9	34.7 \pm 0.9	63.6 \pm 0.9

Chess annotation multi-choice

The annotations within an annotated PGN reflect human evaluation and judgment. We extract 3K game-language pairs from the annotation dataset as the test set to examine how much the model’s value aligns with humans. We construct the chess annotation four-choice task by randomly selecting three annotations from the test set as incorrect options. The following is an example for this task:

Chess annotation
<ul style="list-style-type: none"> • Task Prefix: Annotate the last step of the following PGN. • Input: 1. d4 Nf6 2. c4 e6 3. Nf3 Bb4+ 4. Bd2 a5 5. g3 O-O 6. Bg2 b6 7. O-O Ba6 8. Bg5 Be7 9. Qc2 Nc6 10. a3 h6 11. Bxf6 Bxf6 12. Rd1 Qe7 13. e3 Rae8 14. Nfd2 g5 • Target Score: "Karpov could have resigned here with a clear conscience.": 0, "White intends to further weaken Black’s kingside with 19.h5.": 0, "20...Kh7 21.Bxg6+ fxg6 22.Qxe6 Gives White a winning attack.": 0, "Black overreacts to the positional strength of White’s game. 14...g6 would have been more solid.": 1

We report the multi-choice grade results over two prompts in table 5.4.

Opening multi-choice

A chess opening refers to the initial moves players make at the beginning of a chess game. There are numerous chess openings, each with its name, characteristics, and strategic goals. For example, the Sicilian defense: *1. e4 c5* is one of the black player’s most popular and aggressive chess openings. We use the Lichess opening dataset [129], including 3.5K opening PGNs and their corresponding names, to formulate two tasks: (1) PGN2Opening five-choice task, which aims at choosing the correct opening name for a given PGN, and reversely, (2) Opening2PGN five-choice task, aiming at choosing the correct PGN for a given opening name.

Chess opening (Opening2PGN)

- **Task Prefix:** Show me the PGN of the following opening.
- **Input:** Amar Gambit Opening
- **Target Score:** "1. Nh3 d5 2. g3 e5 3. f4 Bxh3 4. Bxh3 exf4": 1, "1. d4 d5 2. c4 e6 3. Nc3 c5 4. cxd5 exd5 5. dxc5 d4 6. Na4 b5": 0, "1. d4 Nf6 2. g4 Nxg4 3. f3 Nf6 4. e4": 0, "1. Nc3 c5 2. b4": 0, "1. d4 Nf6 2. c4 g6 3. Nc3 Bg7 4. e4 d6 5. f3 O-O 6. Nge2": 0

Chess opening (PGN2Opening)

- **Task Prefix:** Show me the opening name of the following PGN.
- **Input:** 1. Nh3 d5 2. g3 e5 3. f4 Bxh3 4. Bxh3 exf4. The opening name of this PGN is.
- **Target Score:** {"Amar Gambit": 1, "Tarrasch Defense: Tarrasch Gambit": 0, "Indian Defense: Gibbins-Weidenhagen Gambit, Maltese Falcon": 0, "Van Geet Opening: D00fcsseldorf Gambit": 0, "King’s Indian Defense: S00e4misch Variation, Bobotsov-Korchnoi-Petrosian Variation": 0}

We report the result in table 5.5. Our trio of models generally surpasses the performance of two baseline language models across these four tasks in all settings. This result confirms that our models are more effectively aligned with the value function and human judgment/knowledge. Both ChessGPT-Base and ChessGPT-chat deliver outstanding performance in the state-value and opening task. Notably, ChessCLIP displays a surprisingly high level of proficiency in the

Table 5.5: Opening2PGN and PGN2Opening. We test whether ChessGPT can recognize chess openings and the corresponding moves. In Opening2PGN, the model is given the opening moves and asked to choose the corresponding names. In PGN2Opening, the model is given the name of the opening and asked to choose the correct moves. ChessGPT shows better accuracies in both tasks compared to baselines.

Prompt Setting	Models (%)				
	LLAMA	RedPajama	ChessGPT-Base	ChessGPT-Chat	ChessCLIP
Opening2PGN	43.0 \pm 0.9	26.5 \pm 0.8	92.2 \pm 0.5	94.7 \pm 0.4	73.0 \pm 0.8
PGN2Opening	20.0 \pm 0.7	20.7 \pm 0.7	49.3 \pm 0.9	55.8 \pm 0.9	80.5 \pm 0.7

annotation task and the opening task. This result reveals the model’s capacity to extract human judgment and knowledge solely from annotations, even without training in any chess games.

5.4.3 Policy evaluation

Checkmate in one

We incorporate the checkmate-in-one task from BIG-Bench [108] into our evaluation methods. This task challenges the model’s ability to identify a move in a given PGN that would result in a checkmate. By doing so, it measures the model’s capacity to comprehend and apply the rules of chess. The model needs to discern a move that places the opponent’s king under attack and ensures that the king cannot evade capture in the next move. The following is an example for this task:

Checkmate in one

- **Input** :1. e4 e6 2. Ke2 d5 3. e5 c5 4. f4 Nc6 5. Nf3 Qb6 6. g4 Bd7 7. h4 Nge7 8. c3 Ng6 9. d4 cxd4 10. cxd4 Be7 11. Kf2 O-O 12. h5 Nh8 13. Be3 Qxb2+ 14. Kg3 Qxa1 15. Bd3 Qxa2 16. Rh2 Qa1 17. Qc2 Nb4 {Now white has checkmate in one move.}
- **Target Score**: {"Qxa1": 0.0, "Bxh7+": 1.0, "Qd2": 0.0, "Qe2": 0.0, "Qd1": 0.0, "Qc3": 0.0, "Qc7": 0.0, "Bb1": 0.0, "Bc2": 0.0, "Bf5": 0.0, "Bg6": 0.0, "Bf1": 0.0, "Bb5": 0.0, "Bxa6": 0.0, "Bc4": 0.0, "Bb3": 0.0, "Bc1": 0.0, "Bd2": 0.0, "Bc3": 0.0, "Bxb4": 0.0, "Nbd2": 0.0, "Nc3": 0.0, "Na3": 0.0, "Rc2": 0.0, "Rg2": 0.0, "Rh1": 0.0, "Rf2": 0.0, "Re2": 0.0, "Rd2": 0.0, "Rh3": 0.0, "Rh4": 0.0, "Kh2": 0.0, "Kh3": 0.0, "Kh4": 0.0, "g5": 0.0, "f5": 0.0, "Ng5": 0.0, "Nh2": 0.0, "Nf2": 0.0, "Ne1": 0.0.}

Table 5.6: Checkmate in one. ChessGPT demonstrates better performance in the checkmate in one task compared to baselines.

Setting	Models (%)				
	LLAMA	RedPajama	ChessGPT-Base	ChessGPT-Chat	ChessCLIP
With suffix (ESM)	1.6 ± 0.2	0.0 ± 0.0	71.4 ± 0.7	56.8 ± 0.8	N/A
With suffix (MC)	2.6 ± 0.3	0.0 ± 0.0	66.1 ± 0.8	11.3 ± 0.5	2.9 ± 0.3
W/O suffix (ESM)	1.7 ± 0.2	0.0 ± 0.0	26.5 ± 0.8	59.4 ± 0.8	N/A
W/O suffix (MC)	2.2 ± 0.3	0.0 ± 0.0	13.6 ± 0.6	15.4 ± 0.6	N/A

The input to the model is a sequence of moves such that a next possible move is a checkmate. For example, the chess game shown in the figure can checkmate the opponent in one step is “Bxh7+”.

We also find that adding an instruction suffix like *{Now white/black can checkmate in one}* can largely enhance the base model performance. We report the result in two prompts with two metrics (exact-string match (ESM) and multi-choice-grade as (MC)) in table 5.6. ChessGPT-Base and ChessGPT-Chat show a great checkmate ability by surpassing two LLM baselines by a large margin. ChessCLIP does not perform well in this task, likely because there is not much annotation data regarding checkmate-in-one behavior in the annotation dataset.

General policy

In order to assess the model’s generalization ability, we introduced Elo Rating as a factor in the task, aiming to evaluate its capacity to identify PGN and related keywords and generate the appropriate next move at a specified skill level. The following is an example for the task.

General policy

- **Task Prefix:** In the following chess game, you play black.
- **Input:**
 - [Date "2017.04.01"]
 - [White "???"]
 - [Black "???"]
 - [Result "0-1"]
 - [WhiteElo "983"]
 - [BlackElo "2983"]
 - [WhiteRatingDiff "??"]
 - [BlackRatingDiff "??"]
 - [ECO "??"]
 - [Opening "??"]
 - [TimeControl "300+0"]
 - [Termination "Time forfeit"]
 - 1. b3 e5 2. Bb2 Nc6 3. a3 Nf6 4. h3 d5 5. g3 Bd6 6. Bg2 O-O 7. e3 e4 8. d3 Be5 9. d4 Bd6 10. Ne2 Ne7 11. c4 c6 12. Nbc3 Nf5 13. Qd2 a5 14. Qc2 Be6 15. cxd5 cxd5 16. Nb5 Rc8 17. Qd2 Qb6 18. Nxd6 Nxd6 19. Qd1 Rc7 20. a4 Rfc8 21. Ba3 Nf5 22. Bc5 Rxc5 23. dxc5
- **Target Score:** {"Rxc5": 0.0, "Qa7": 0.02, "Qd8": 0.05, "Qc7": 0.083, "Qb4+": 0.11, "Qxc5": 0.13, "Qc6": 0.16, "Qa6": 0.19, "Nxe3": 0.22, "Nxc3": 0.25, "Nd4": 0.27, "Rc6": 0.30, "Ng4": 0.33, "Qxb3": 0.36, "d4": 0.38, "Nh4": 0.41, "Kh8": 0.44, "Nd7": 0.47, "h6": 0.5, "g6": 0.52, "Rf8": 0.55, "Ra8": 0.58, "Ne7": 0.61, "Qd6": 0.63, "h5": 0.66, "Re8": 0.69, "Kf8": 0.72, "Qb5": 0.75, "Rd8": 0.77, "Bd7": 0.80, "Ne8": 0.83, "Rb8": 0.86, "Nd6": 0.88, "g5": 0.91, "Rc7": 0.94, "Nh5": 0.97, "Nh6": 1.0}

The model's selection of the next legal move is assigned a score that is normalized based on the win rate observed in the raw data. Table 5.7 presents the results representing the performance of different models in selecting the most

Table 5.7: Elo Rating 1700-2000.

LLM Models	Move Score
LLAMA	55.1 \pm 1.1
RedPajama	56.4 \pm 0.9
ChessGPT-Base	59.6 \pm 1.0
ChessGPT-Chat	60.3 \pm 1.0

suitable move for white chess. Notably, all models surpass the performance of the random policy ($\approx 50\%$) as the Elo Ratings correspond to relatively high skill levels among human players. Further analyzing the performance of different models across varying Elo Ratings is crucial for understanding the observed results. The minor variations in move scores for different Elo Rating scenarios in table 5.8 indicate that ChessGPT-Base struggles to incorporate Elo rating information into its decision-making process effectively. We hypothesize this to be due to the model’s limited understanding of the nuanced characteristics associated with distinct Elo Ratings. The complexity of the task and the challenges in accurately accounting for diverse playing styles further contribute to the limited variations in move scores across different Elo Ratings. Consequently, paying attention to this information can lead to the model learning an average policy for each Elo Rating, resulting in subpar overall performance. We observe similar findings in the black chess test. To further validate this hypothesis, we conduct an input attention visualization. Refer to appendix C.3.1 for more details.

To clarify, the dataset we have presented encompasses a wide range of games and varying Elo ratings, as shown in fig. 5.2, which possesses the potential to effectively capture and generalize intricate patterns and policies associated with different Elo levels. However, the current training method might need to emphasize these nuanced features sufficiently. Our findings highlight a potential direction for future research: enhancing the model’s ability to integrate better and utilize metadata such as Elo Rating and other auxiliary data. The model’s overall generalization can be further improved by addressing these aspects.

Table 5.8: ChessGPT-Base in different Elo rating results

Elo Rating	Move Score
700-1000	59.4 ± 1.0
1200-1500	58.9 ± 0.9
1700-2000	59.6 ± 1.0
2700-3000	59.8 ± 1.0

5.4.4 Qualitative results

We also perform a qualitative comparison between our models (ChessGPT-Chat and ChessGPT-Base) and the baselines. We ask the language models a series of questions ranging from factual knowledge of chess to requesting the models perform some operational tasks related to chess. We found that ChessGPT-base performed similarly to RedPajama. Both models can produce factual answers to some questions. However, they failed to generate coherent answers when asked to perform tasks such as providing commentary on chess moves or converting the PGN notation to FEN. ChessGPT-Chat gives more factual answers and performs better when prompted to generate analysis and perform other chess-related tasks. Refer to appendix C.5 for qualitative analysis.

5.5 Discussion

The pursuit of creating artificial intelligence capable of playing chess can be traced back to the very beginning of the history of computer science [1]. Chess engines today achieve superhuman-level performance by utilizing human knowledge [10] or self-play [36].

Recently, there has been increasing interest in improving the interpretability [87] of these systems and their alignment with human behavior [48] besides strong performance. A chess engine that aligns with human behavior may unlock many exciting opportunities. For example, it can be used as a personalized tutor for chess beginners [48]. Some research efforts also concentrated on employing LLMs to learn policies in chess [50, 72]. However, these studies mainly center on small-scale datasets or limited training.

There has been increasing interest in leveraging Internet-scale knowledge for creating agents capable of generalizing across many tasks and capabilities [114, 76, 90]. For example, MineDojo [76] introduced a framework on Minecraft for understanding how to enable artificial agents to learn in an open-ended environment. More recently, there has been a surge in research that treats LLMs as agents, aiming to harness their Internet-scale knowledge for decision-making tasks [121, 112, 115, 119]. In contrast to these studies, which typically rely on

powerful LLMs like GPT-4 [96], we concentrate more on training, especially the interplay between language modeling and policy learning.

While the chess dataset provided in this study is valuable, it is important to acknowledge its limitations. One limitation is the potential bias introduced by relying on historical Lichess matches from different time periods. This may result in variations in player strategies, popular openings, and game trends over time, potentially impacting the generalizability of the dataset. Additionally, it is worth noting that the dataset predominantly focuses on standard chess and may not encompass the full spectrum of chess variants. Researchers interested in exploring niche or less popular variants may need to gather additional data from specific sources to ensure comprehensive coverage of different chess variants. These considerations are crucial to ensure the validity and applicability of research findings based on the provided dataset.

The availability of a comprehensive and diverse chess dataset can have a significant societal impact. First and foremost, it can contribute to the development of more advanced and intelligent chess-playing agents. These agents can be utilized in various applications, such as chess analysis, training tools for players of different skill levels, and even as opponents for chess enthusiasts. The dataset can also facilitate the advancement of chess education by providing valuable resources for tutorials, interactive learning platforms, and strategic guidance. Additionally, the dataset can inspire research in the field of artificial intelligence, contributing to the development of innovative techniques that can be applied beyond the domain of chess. Lastly, our dataset encourages the exploration of explainable AI methods in chess, enabling players to understand and learn from the reasoning behind the model’s moves, thereby promoting transparency and trust in AI systems.

Although the models and paradigms presented in this chapter may appear somewhat detached from chapter 3 and chapter 4, we believe they offer a compelling domain for applying reinforcement learning techniques that we have not yet explored due to time constraint. The challenges we face here closely resemble those identified in offline reinforcement learning research - particularly how to learn optimal policies from fixed datasets without interactive environment feedback. For instance, our replay dataset extracted from online chess games

contains moves of varying quality, which presents fundamental challenges similar to ones we described in chapter 2 where passive learning from fixed data distributions significantly underperforms active learning. Simple language modeling objectives may struggle to extract master-level play from such heterogeneous data. The techniques presented in earlier chapters, particularly learned reward functions as described in chapter 4, offer promising solutions to these challenges. Incorporating preference-based learning approaches like those in reinforcement learning from human feedback (RLHF) [27, 123, 106], could potentially be used to expand our agents’ capabilities. Thus, we are excited by the many interesting possibilities in exploring how techniques presented in earlier chapters can address the challenges that naturally arise in this domain.

5.6 Conclusion

In this chapter, we introduce a new large-scale dataset and benchmark on chess to encourage the study of the interplay between historical policy data and natural language knowledge. We accompany our dataset with an evaluation framework for assessing language models’ capability in chess. We showcase two models, ChessCLIP and ChessGPT, demonstrating promising results for learning the interplay between language and action. Our results indicate that we are only beginning to understand how to bridge the gap between policy learning and language modeling and, more broadly, leverage unstructured data to build generally capable agents. We hope our dataset and benchmark can make future policy and language alignment research more accessible.

Chapter 6

Conclusions

Summary of Contributions

This thesis has addressed several key challenges in offline reinforcement learning (RL), focusing on the utilization of imperfect data for efficient behavior learning. The work spans a spectrum of topics, from leveraging sub-optimal demonstrations to proposing novel algorithms that enable agents to generalize across diverse experiences.

In chapter 3, we explored how RL agents can be fine-tuned using sub-optimal offline data combined with limited online experience. We showed that while offline experiences alone may be insufficient for learning optimal behavior, small amounts of online interaction can help bridge this gap. The trade-offs involved in choosing between different fine-tuning techniques were also analyzed, providing insights into how agents can adapt to new tasks more effectively.

In chapter 4, we presented the OTR algorithm, which enables offline learning in environments without explicit reward information by learning from reward-free datasets. This contribution is particularly significant in cases where direct reward supervision is impractical, yet small amount of expert demonstrations can still be provided. By combining diverse experiences, OTR enhances the generalization ability of imitation learning models, moving toward a more flexible framework for RL.

Finally, in chapter 5, we demonstrated how combining data from multiple sources can produce a generalist model capable of performing a wide range of

tasks. This chapter sets the groundwork for future learning paradigms where agents can harness structured and unstructured data alike, developing a broader set of capabilities in the process.

Future directions

While this thesis provides important steps toward more flexible and robust RL systems, there remain several exciting avenues for future work.

Leveraging action-free datasets. A key challenge in RL is the reliance on action-labeled datasets, which limits the scope of available data sources. We touched on this problem in chapter 4, showing that it is possible to infer rewards without access to explicit actions. However, the development of inverse dynamics models (IDMs) that can accurately predict actions from pure observation sequences is still in its early stages. For example, Baker et al. [73] shows that it is possible to perform behavior cloning in MineCraft using from action-free video demonstrations by first learning a inverse dynamics model using a small amount of action-labeled demonstrations and uses that to learn to predict actions. More recent work from Bruce et al. [117] demonstrates the possibility of learning a generative model of the world without any action annotation. Expanding the use of action-free datasets, such as video recordings without accompanying action annotations, would be a valuable step forward. We believe that combining action-learning methods with reward learning methods such as OTR provides exciting opportunities to expand the amount of data available for training decision-making agents.

Cross-embodiment learning. Another underexplored area is the transfer of knowledge between different embodiments. To date, most (offline) RL approaches assume that training and deployment occur within the same observation and action spaces, but for many applications, useful data may be available despite having different observation and action spaces. There is potential to benefit from cross-embodiment learning, where knowledge from different domains and embodiments transfer to improve the generalization. In the imitation learning

literature, this is sometimes also known as third-person imitation learning [30]. For instance, data from human demonstrators could help build humanoid agents, even if the specific action spaces differ. This opens up possibilities for agents to generalize across varying physical systems.

Concluding remarks

This thesis presents several contributions that advance the field of reinforcement learning, particularly in dealing with imperfect and diverse data sources. While significant progress has been made, the challenges ahead, such as incorporating action-free datasets and cross-embodiment learning highlight the importance of continuing to develop flexible and generalizable RL systems. The work presented here lays the groundwork for future innovations in RL and points to a future where agents can learn effectively from the complex, imperfect, and varied data in the real world.

Bibliography

- [1] A. M. Turing. Digital computers applied to games. *Faster than Thought*, 1953 (cited on page 100).
- [2] D. A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, 1988. URL: <https://proceedings.neurips.cc/paper/1988/hash/812b4ba287f5ee0bc9d43bbf5bbe87fb-Abstract.html> (cited on page 32).
- [3] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992. DOI: [10.1007/BF00992698](https://doi.org/10.1007/BF00992698) (cited on page 27).
- [4] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1st edition, 1996. 491 pages (cited on page 27).
- [5] C. G. Atkeson and S. Schaal. Robot learning from demonstration. In *Proceedings of the 14th International Conference on Machine Learning*, 1997 (cited on page 58).
- [6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. DOI: [10.1016/S0004-3702\(98\)00023-X](https://doi.org/10.1016/S0004-3702(98)00023-X) (cited on page 25).
- [7] V. Konda and J. Tsitsiklis. Actor-critic algorithms. In *Advances in Neural Information Processing Systems*, 1999. URL: <https://proceedings.neurips.cc/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html> (cited on page 27).

- [8] A. Y. Ng, D. Harada, and S. J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999 (cited on pages 33, 72).
- [9] A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning*, 2000. URL: <https://dl.acm.org/doi/10.5555/645529.657801> (cited on page 33).
- [10] M. Campbell, A. Hoane, and F.-h. Hsu. Deep Blue. *Artificial Intelligence*, 134(1-2):57–83, 2002. DOI: [10.1016/S0004-3702\(01\)00129-1](https://doi.org/10.1016/S0004-3702(01)00129-1) (cited on page 100).
- [11] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, 2004. DOI: [10.1145/1015330.1015430](https://doi.org/10.1145/1015330.1015430) (cited on page 58).
- [12] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007. DOI: [10.1109/TSMCB.2006.886952](https://doi.org/10.1109/TSMCB.2006.886952) (cited on page 58).
- [13] L. Yujian and L. Bo. A normalized Levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, 2007. DOI: [10.1109/TPAMI.2007.1078](https://doi.org/10.1109/TPAMI.2007.1078) (cited on page 92).
- [14] G. Neumann and J. Peters. Fitted Q-iteration by advantage weighted regression. In *Advances in Neural Information Processing Systems*, 2008. URL: https://papers.nips.cc/paper_files/paper/2008/hash/f79921bbae40a577928b76d2fc3edc2a-Abstract.html (cited on page 49).
- [15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009. DOI: [10.1109/CVPR.2009.5206848](https://doi.org/10.1109/CVPR.2009.5206848) (cited on page 18).

- [16] M. P. Deisenroth and C. E. Rasmussen. PILCO: a model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning*, 2011. URL: <https://dl.acm.org/doi/10.5555/3104482.3104541> (cited on pages 27, 28).
- [17] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning: State-of-the-Art*, pages 45–73. Springer, Berlin, Heidelberg, 2012. DOI: [10.1007/978-3-642-27645-3_2](https://doi.org/10.1007/978-3-642-27645-3_2) (cited on pages 18, 28, 30, 31).
- [18] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: a physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012. DOI: [10.1109/IRROS.2012.6386109](https://doi.org/10.1109/IRROS.2012.6386109) (cited on page 40).
- [19] M. Cuturi. Sinkhorn distances: lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, 2013. URL: <https://papers.nips.cc/paper/2013/hash/af21d0c97db2e27e13572cbf59eb343d-Abstract.html> (cited on pages 59, 60).
- [20] P. Englert, A. Paraschos, J. Peters, and M. P. Deisenroth. Model-based imitation learning by probabilistic trajectory matching. In *2013 IEEE International Conference on Robotics and Automation*, 2013. DOI: [10.1109/ICRA.2013.6630832](https://doi.org/10.1109/ICRA.2013.6630832) (cited on page 58).
- [21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014. URL: <https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html> (cited on page 35).
- [22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. DOI: [10.1038/nature14236](https://doi.org/10.1038/nature14236) (cited on pages 38, 40, 88).

- [23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016. DOI: [10.1109/CVPR.2016.90](https://doi.org/10.1109/CVPR.2016.90) (cited on pages [87](#), [153](#), [161](#)).
- [24] J. Ho and S. Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, 2016. URL: <https://papers.nips.cc/paper/2016/hash/cc7e2b878868cbae992d1fb743995d8f-Abstract.html> (cited on page [34](#)).
- [25] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. URL: <https://dl.acm.org/doi/10.5555/2946645.2946684> (cited on page [88](#)).
- [26] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *The 4th International Conference on Learning Representations*, 2016. URL: <https://arxiv.org/abs/1509.02971> (cited on pages [38](#), [39](#), [53](#)).
- [27] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html (cited on page [102](#)).
- [28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. 2017. arXiv: [1707.06347](https://arxiv.org/abs/1707.06347) (cited on page [35](#)).
- [29] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017. DOI: [10.1038/nature24270](https://doi.org/10.1038/nature24270) (cited on pages [17](#), [18](#)).

- [30] B. C. Stadie, P. Abbeel, and I. Sutskever. Third person imitation learning. In *The 5th International Conference on Learning Representations*, 2017. URL: <https://openreview.net/forum?id=B16dGcqlx> (cited on page 105).
- [31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. ukasz Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (cited on page 79).
- [32] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. In *The 6th International Conference on Learning Representations*, 2018. URL: <https://openreview.net/forum?id=S1ANxQW0b> (cited on pages 46, 53).
- [33] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL: <https://proceedings.mlr.press/v80/fujimoto18a.html> (cited on pages 39–41, 53).
- [34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018. URL: <https://proceedings.mlr.press/v80/haarnoja18b.html> (cited on pages 28, 53, 132).
- [35] E. Perez, F. Strub, H. de Vries, V. Dumoulin, and A. Courville. FiLM: visual reasoning with a general conditioning layer. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence and 30th Innovative Applications of Artificial Intelligence Conference and 8th AAAI Symposium on Educational Advances in Artificial Intelligence*, 2018 (cited on pages 87, 161).
- [36] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters

- chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. DOI: [10.1126/science.aar6404](https://doi.org/10.1126/science.aar6404) (cited on pages [17](#), [87](#), [100](#), [161](#)).
- [37] R. S. Sutton and A. G. Barto. *Reinforcement learning*. MIT press, 2nd edition, 2018 (cited on pages [17](#), [27](#)).
- [38] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, 2019. URL: <https://proceedings.mlr.press/v97/fujimoto19a.html> (cited on pages [30](#), [49](#)).
- [39] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. MineRL: a large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 2019 (cited on page [19](#)).
- [40] P. J. Ortiz Suárez, B. Sagot, and L. Romary. Asynchronous pipelines for processing huge corpora on medium to low resource infrastructures. In *Proceedings of the Workshop on Challenges in the Management of Large Corpora*, 2019. DOI: [10.14618/IDS-PUB9021](https://doi.org/10.14618/IDS-PUB9021) (cited on pages [84](#), [150](#)).
- [41] X. B. Peng, A. Kumar, G. Zhang, and S. Levine. Advantage-weighted regression: simple and scalable off-policy reinforcement learning. 2019. arXiv: [1910.00177](https://arxiv.org/abs/1910.00177) (cited on page [49](#)).
- [42] H. Xiao, M. Herman, J. Wagner, S. Ziesche, J. Etesami, and T. H. Linh. Wasserstein adversarial imitation learning. 2019. arXiv: [1906.08113](https://arxiv.org/abs/1906.08113) (cited on page [35](#)).
- [43] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020. URL: <https://papers.nips.cc/paper/2020/hash/1457c0d6bfc4967418bfb8ac142f64a-Abstract.html> (cited on page [88](#)).

- [44] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima, S. Presser, and C. Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. 2020. arXiv: [2101.00027 \[cs\]](#) (cited on pages [18](#), [84](#), [150](#)).
- [45] I. Kostrikov, O. Nachum, and J. Tompson. Imitation learning via off-policy distribution matching. In *The 8th International Conference on Learning Representations*, 2020. URL: <https://openreview.net/forum?id=Hyg-JC4FDr> (cited on pages [34](#), [35](#)).
- [46] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative Q-learning for offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html> (cited on pages [40](#), [49](#), [129](#)).
- [47] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: tutorial, review, and perspectives on open problems. 2020. arXiv: [2005.01643](#) (cited on pages [18](#), [28](#)).
- [48] R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson. Aligning superhuman AI with human behavior: chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. DOI: [10.1145/3394486.3403219](#) (cited on pages [82](#), [100](#)).
- [49] L. Monier, J. Kmec, A. Laterre, T. Pierrot, V. Courgeau, O. Sigaud, and K. Beguir. Offline reinforcement learning hands-on. 2020. arXiv: [2011.14379](#) (cited on page [55](#)).
- [50] D. Noever, M. Ciolino, and J. Kalin. The chess transformer: mastering play using generative language models. 2020. arXiv: [2008.04057](#) (cited on page [100](#)).
- [51] G. Peyré and M. Cuturi. Computational optimal transport. 2020. arXiv: [1803.00567](#) (cited on page [59](#)).

- [52] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL: <http://jmlr.org/papers/v21/20-074.html> (cited on pages 84, 150).
- [53] S. Reddy, A. D. Dragan, and S. Levine. SQIL: imitation learning via reinforcement learning with sparse rewards. In *The 8th International Conference on Learning Representations*, 2020. URL: <https://openreview.net/forum?id=S1xKd24twB> (cited on page 68).
- [54] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. DOI: [10.1038/s41586-020-03051-4](https://doi.org/10.1038/s41586-020-03051-4) (cited on page 84).
- [55] S. Sinha, J. Song, A. Garg, and S. Ermon. Experience replay with likelihood-free importance weights, 2020. URL: https://openreview.net/forum?id=ioXEbG_Sf-a (cited on page 132).
- [56] Z. Wang, A. Novikov, K. Zolna, J. S. Merel, J. T. Springenberg, S. E. Reed, B. Shahriari, N. Siegel, C. Gulcehre, N. Heess, and N. de Freitas. Critic regularized regression. In *Advances in Neural Information Processing Systems*, 2020. URL: <https://proceedings.neurips.cc/paper/2020/hash/588cb956d6bbe67078f29f8de420a13d-Abstract.html> (cited on page 67).
- [57] K. Zolna, A. Novikov, K. Konyushkova, C. Gulcehre, Z. Wang, Y. Aytar, M. Denil, N. de Freitas, and S. Reed. Offline learning from demonstrations and unlabeled experience. 2020. arXiv: [2011.13885](https://arxiv.org/abs/2011.13885) (cited on pages 68, 69).
- [58] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: reinforcement learning via sequence modeling. In *Advances in Neural Information Processing Systems*, 2021. URL: https://papers.nips.cc/paper_files/p

- [aper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html](#) (cited on page 79).
- [59] S. Cohen, B. Amos, M. P. Deisenroth, M. Henaff, E. Vinitzky, and D. Yarats. Imitation learning from pixel observations for continuous control, 2021. URL: <https://openreview.net/forum?id=JLbXkHkLcG6> (cited on pages 35, 62, 64, 65).
 - [60] R. Dadashi, L. Hussenot, M. Geist, and O. Pietquin. Primal Wasserstein imitation learning. In *The 9th International Conference on Learning Representations*, 2021. URL: <https://openreview.net/forum?id=TtYSU29zgR> (cited on pages 35, 62, 65, 74, 75, 137).
 - [61] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021. DOI: [10.1007/s10994-021-05961-4](https://doi.org/10.1007/s10994-021-05961-4) (cited on page 27).
 - [62] R. Flamary, N. Courty, A. Gramfort, M. Z. Alaya, A. Boisbunon, S. Chambon, L. Chapel, A. Corenflos, K. Fatras, N. Fournier, L. Gautheron, N. T. Gayraud, H. Janati, A. Rakotomamonjy, I. Redko, A. Rolet, A. Schutz, V. Seguy, D. J. Sutherland, R. Tavenard, A. Tong, and T. Vayer. POT: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021. URL: <http://jmlr.org/papers/v22/20-451.html> (cited on page 76).
 - [63] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: datasets for deep data-driven reinforcement learning. 2021. arXiv: [2004.07219](https://arxiv.org/abs/2004.07219) (cited on pages 40, 58, 67).
 - [64] S. Fujimoto and S. Gu. A minimalist approach to offline reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021. URL: https://proceedings.neurips.cc/paper_files/paper/2021/hash/a8166da05c5a094f7dc03724b41886e5-Abstract.html (cited on pages 39, 40, 72, 73).

- [65] M. Janner, Q. Li, and S. Levine. Offline reinforcement learning as one big sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/099fe6b0b444c23836c4a5d07346082b-Abstract.html> (cited on page 79).
- [66] Y. Luo, A. Filieri, and Y. Zhou. Symbolic parallel adaptive importance sampling for probabilistic program analysis. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021. DOI: [10.1145/3468264.3468593](https://doi.org/10.1145/3468264.3468593) (cited on page 21).
- [67] A. Nair, A. Gupta, M. Dalal, and S. Levine. AWAC: accelerating online reinforcement learning with offline datasets. 2021. arXiv: [2006.09359](https://arxiv.org/abs/2006.09359) (cited on pages 41–43, 49, 53–55, 67).
- [68] M. Orsini, A. Raichuk, L. Hussenot, D. Vincent, R. Dadashi, S. Girgin, M. Geist, O. Bachem, O. Pietquin, and M. Andrychowicz. What matters for adversarial imitation learning? In *Advances in Neural Information Processing Systems*, 2021. URL: <https://proceedings.neurips.cc/paper/2021/hash/7b647a7d88f4d6319bf0d600d168dbeb-Abstract.html> (cited on pages 35, 63).
- [69] G. Ostrovski, P. S. Castro, and W. Dabney. The Difficulty of Passive Learning in Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2021. URL: <https://openreview.net/forum?id=nPHA8fGicZk> (cited on pages 30, 31).
- [70] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, 2021. URL: <https://proceedings.mlr.press/v139/radford21a.html> (cited on page 86).
- [71] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Chaining behaviors from data with model-free reinforcement learning. In *Proceed-*

- ings of the 2020 Conference on Robot Learning, 2021. URL: <https://proceedings.mlr.press/v155/singh21a.html> (cited on page 68).
- [72] A. Stöckl. Watching a language model learning chess. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing*, 2021. URL: <https://aclanthology.org/2021.ranlp-1.153> (cited on page 100).
- [73] B. Baker, I. Akkaya, P. Zhokov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro, and J. Clune. Video PreTraining (VPT): learning to act by watching unlabeled online videos. In *Advances in Neural Information Processing Systems*, 2022. URL: https://papers.nips.cc/paper_files/paper/2022/hash/9c7008aff45b5d8f0973b23e1a22ada0-Abstract-Conference.html (cited on pages 79, 104).
- [74] S. Black, S. Biderman, E. Hallahan, Q. Anthony, L. Gao, L. Golding, H. He, C. Leahy, K. McDonell, J. Phang, M. Pieler, U. S. Prashanth, S. Purohit, L. Reynolds, J. Tow, B. Wang, and S. Weinbach. GPT-NeoX-20B: an open-source autoregressive language model. In *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022. DOI: [10.18653/v1/2022.bigscience-1.9](https://doi.org/10.18653/v1/2022.bigscience-1.9) (cited on page 88).
- [75] M. Cuturi, L. Meng-Papaxanthos, Y. Tian, C. Bunne, G. Davis, and O. Teboul. Optimal transport tools (OTT): a JAX toolbox for all things Wasserstein. 2022. arXiv: [2201.12324](https://arxiv.org/abs/2201.12324) (cited on pages 60, 63, 69).
- [76] L. Fan, G. Wang, Y. Jiang, A. Mandlekar, Y. Yang, H. Zhu, A. Tang, D.-A. Huang, Y. Zhu, and A. Anandkumar. MineDojo: building open-ended embodied agents with internet-scale knowledge. In *Advances in Neural Information Processing Systems*, 2022. URL: https://papers.nips.cc/paper_files/paper/2022/hash/74a67268c5cc5910f64938cac4526a90-Abstract-Datasets_and_Benchmarks.html (cited on pages 85, 100).
- [77] A. Fickinger, S. Cohen, S. Russell, and B. Amos. Cross-domain imitation learning via optimal transport. In *The 10th International Conference on*

- Learning Representations*, 2022. URL: <https://openreview.net/forum?id=xP3cPq2hQC> (cited on page 76).
- [78] D. J. Foster, A. Krishnamurthy, D. Simchi-Levi, and Y. Xu. Offline reinforcement learning: Fundamental barriers for value function approximation. In *Proceedings of 35th Conference on Learning Theory*, –July 5, 2022. URL: <https://proceedings.mlr.press/v178/foster22a.html> (cited on page 31).
- [79] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sinopalnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, L. Hussenot, R. Dadashi, G. Dulac-Arnold, M. Orsini, A. Jacq, J. Ferret, N. Vieillard, S. K. S. Ghasemipour, S. Girgin, O. Pietquin, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Friesen, R. Haroun, A. Novikov, S. G. Colmenarejo, S. Cabi, C. Gulcehre, T. L. Paine, S. Srinivasan, A. Cowie, Z. Wang, B. Piot, and N. de Freitas. Acme: a research framework for distributed reinforcement learning. 2022. arXiv: 2006.00979 (cited on pages 74, 128).
- [80] G.-H. Kim, S. Seo, J. Lee, W. Jeon, H. Hwang, H. Yang, and K.-E. Kim. DemoDICE: offline imitation learning with supplementary imperfect demonstrations. In *The 10th International Conference on Learning Representations*, 2022. URL: <https://openreview.net/forum?id=BrPdX1bDZkQ> (cited on pages 67, 69).
- [81] I. Kostrikov, A. Nair, and S. Levine. Offline reinforcement learning with implicit Q-learning. In *The 10th International Conference on Learning Representations*, 2022. URL: <https://openreview.net/forum?id=68n2s9ZJWF8> (cited on pages 41, 44, 50, 54, 55, 66–68, 130, 137).
- [82] H. Laurençon, L. Saulnier, T. Wang, C. Akiki, A. V. del Moral, T. L. Scao, L. V. Werra, C. Mou, E. G. Ponferrada, H. Nguyen, J. Froberg, M. Šaško, Q. Lhoest, A. McMillan-Major, G. Dupont, S. Biderman, A. Rogers, L. B. Allal, F. D. Toni, G. Pistilli, O. Nguyen, S. Nikpoor, M. Masoud, P. Colombo, J. de la Rosa, P. Villegas, T. Thrush, S. Longpre, S. Nagel, L. Weber, M. R. Muñoz, J. Zhu, D. V. Strien, Z. Alyafeai, K. Almubarak, V. M. Chien, I. Gonzalez-Dios, A. Soroa, K. Lo, M. Dey,

- P. O. Suarez, A. Gokaslan, S. Bose, D. I. Adelani, L. Phan, H. Tran, I. Yu, S. Pai, J. Chim, V. Lepercq, S. Ilic, M. Mitchell, S. Luccioni, and Y. Jernite. The BigScience ROOTS corpus: a 1.6TB composite multilingual dataset. In *Advances in Neural Information Processing Systems*, 2022. URL: https://papers.nips.cc/paper_files/paper/2022/hash/ce9e92e3de2372a4b93353eb7f3dc0bd-Abstract-Datasets_and_Benchmarks.html (cited on pages 141, 143, 146).
- [83] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Proceedings of the 5th Conference on Robot Learning*, –Nov. 11, 2022. URL: <https://proceedings.mlr.press/v164/lee22d.html> (cited on pages 41–43, 53–55, 132, 135).
- [84] L. Li, P. Zhang, H. Zhang, J. Yang, C. Li, Y. Zhong, L. Wang, L. Yuan, L. Zhang, J. Hwang, K. Chang, and J. Gao. Grounded language-image pre-training. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. DOI: [10.1109/CVPR52688.2022.01069](https://doi.org/10.1109/CVPR52688.2022.01069) (cited on page 152).
- [85] Y. Lu, K. Hausman, Y. Chebotar, M. Yan, E. Jang, A. Herzog, T. Xiao, A. Irpan, M. Khansari, D. Kalashnikov, and S. Levine. AW-Opt: learning robotic skills with imitation and reinforcement at scale. In *Proceedings of the 5th Conference on Robot Learning*, 2022. URL: <https://proceedings.mlr.press/v164/lu22a.html> (cited on page 55).
- [86] Y. Luo, J. Kay, E. Grefenstette, and M. P. Deisenroth. Finetuning from offline reinforcement learning: challenges, trade-offs and practical solutions. In *The 5th Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2022 (cited on pages 21, 22).
- [87] T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, M. Wattenberg, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik. Acquisition of chess knowledge in AlphaZero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, 2022. DOI: [10.1073/pnas.2206625119](https://doi.org/10.1073/pnas.2206625119) (cited on page 100).

- [88] S. Mishra, D. Khashabi, C. Baral, and H. Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, 2022. DOI: [10.18653/v1/2022.acl-long.244](https://doi.org/10.18653/v1/2022.acl-long.244) (cited on page 85).
- [89] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, 2022. URL: https://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html (cited on pages 37, 85).
- [90] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barthmaron, M. Giménez, 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. A generalist agent. *Transactions on Machine Learning Research*, 2022. URL: <https://openreview.net/forum?id=1ikK0kHjvj> (cited on page 100).
- [91] S. Toshniwal, S. Wiseman, K. Livescu, and K. Gimpel. Chess as a testbed for language model state tracking. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022. DOI: [10.1609/aaai.v36i10.21390](https://doi.org/10.1609/aaai.v36i10.21390) (cited on page 89).
- [92] J. Wei, M. Bosma, V. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *The 10th International Conference on Learning Representations*, 2022. URL: <https://openreview.net/forum?id=gEZrGCozdqR> (cited on page 85).
- [93] T. Yu, A. Kumar, Y. Chebotar, K. Hausman, C. Finn, and S. Levine. How to leverage unlabeled data in offline reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*,

2022. URL: <https://proceedings.mlr.press/v162/you22c.html> (cited on page 68).
- [94] Q. Zheng, A. Zhang, and A. Grover. Online decision transformer. In *Proceedings of the 39th International Conference on Machine Learning*, 2022. URL: <https://proceedings.mlr.press/v162/zheng22c.html> (cited on pages 41, 44, 49, 50, 54, 55).
- [95] P. J. Ball, L. Smith, I. Kostrikov, and S. Levine. Efficient online reinforcement learning with offline data. In *Proceedings of the 40th International Conference on Machine Learning*, 2023. URL: <https://proceedings.mlr.press/v202/ball23a.html> (cited on pages 28, 130–132).
- [96] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4. 2023. arXiv: 2303.12712 (cited on pages 86, 101).
- [97] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: an open-source chatbot impressing GPT-4 with 90% ChatGPT quality. 2023. URL: <https://lmsys.org/blog/2023-03-30-vicuna/> (cited on page 163).
- [98] M. Conover, M. Hayes, A. Mathur, J. Xie, J. Wan, S. Shah, A. Ghodsi, P. Wendell, M. Zaharia, and R. Xin. Free dolly: introducing the world’s first truly open instruction-tuned LLM. 2023. URL: <https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm> (cited on pages 88, 162).
- [99] X. Feng, Y. Luo, Z. Wang, H. Tang, M. Yang, K. Shao, D. Mguni, Y. Du, and J. Wang. ChessGPT: bridging policy learning and language modeling. In *Advances in Neural Information Processing Systems*, 2023. URL: https://papers.nips.cc/paper_files/paper/2023/hash/16b14e3f288f076e0ca73bdad6405f77-Abstract-Datasets_and_Benchmarks.html (cited on pages 21, 22).

- [100] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu. Do as I can, not as I say: grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, 2023. URL: <https://proceedings.mlr.press/v205/ichter23a.html> (cited on page 17).
- [101] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023. DOI: [10.1038/s41586-023-06419-4](https://doi.org/10.1038/s41586-023-06419-4) (cited on pages 17, 32).
- [102] A. Köpf, Y. Kilcher, D. von Rütte, S. Anagnostidis, Z. R. Tam, K. Stevens, A. Barhoum, D. M. Nguyen, O. Stanley, R. Nagyfi, S. Es, S. Suri, D. A. Glushkov, A. V. Dantuluri, A. Maguire, C. Schuhmann, H. Nguyen, and A. J. Mattick. OpenAssistant conversations - democratizing large language model alignment. In *Advances in Neural Information Processing Systems*, 2023. URL: https://papers.nips.cc/paper_files/paper/2023/hash/949f0f8f32267d297c2d4e3ee10a2e7e-Abstract-Datasets_and_Benchmarks.html (cited on pages 88, 162).
- [103] I. Kostrikov, L. M. Smith, and S. Levine. Demonstrating a walk in the park: learning to walk in 20 minutes with model-free reinforcement learning. In *Proceedings of Robotics: Science and Systems*, 2023. DOI: [10.15607/RSS.2023.XIX.056](https://doi.org/10.15607/RSS.2023.XIX.056) (cited on page 28).
- [104] Y. Luo, Z. Jiang, S. Cohen, E. Grefenstette, and M. P. Deisenroth. Optimal transport for offline imitation learning. In *The 11th International Conference on Learning Representations*, 2023. URL: <https://openreview.net/forum?id=MhuFzFsrfvH> (cited on pages 21, 22).

- [105] B. Peng, C. Li, P. He, M. Galley, and J. Gao. Instruction tuning with GPT-4. 2023. arXiv: [2304.03277](https://arxiv.org/abs/2304.03277) (cited on pages [88](#), [162](#)).
- [106] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL: <https://openreview.net/forum?id=HPuSIXJaa9> (cited on page [102](#)).
- [107] A. B. Simon Stewart. Selenium, 2023. URL: <https://github.com/SeleniumHQ/selenium> (cited on pages [148](#), [149](#)).
- [108] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shueb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, A. Kluska, A. Lewkowycz, A. Agarwal, A. Power, A. Ray, A. Warstadt, A. W. Kocurek, A. Safaya, A. Tazarv, A. Xiang, A. Parrish, A. Nie, A. Hussain, A. Askeel, A. Dsouza, A. Slone, A. Rahane, A. S. Iyer, A. J. Andreassen, A. Madotto, A. Santilli, A. Stuhlmüller, A. M. Dai, A. La, A. Lampinen, A. Zou, A. Jiang, A. Chen, A. Vuong, A. Gupta, A. Gottardi, A. Norelli, A. Venkatesh, A. Gholamidavoodi, A. Tabassum, A. Menezes, A. Kirubakaran, A. Mullokandov, A. Sabharwal, A. Herrick, A. Efrat, A. Erdem, A. Karakaş, B. R. Roberts, B. S. Loe, B. Zoph, B. Bojanowski, B. Özyurt, B. Hedayatnia, B. Neyshabur, B. Inden, B. Stein, B. Ekmekci, B. Y. Lin, B. Howald, B. Orinion, C. Diao, C. Dour, C. Stinson, C. Argueta, C. Ferri, C. Singh, C. Rathkopf, C. Meng, C. Baral, C. Wu, C. Callison-Burch, C. Waites, C. Voigt, C. D. Manning, C. Potts, C. Ramirez, C. E. Rivera, C. Siro, C. Raffel, C. Ashcraft, C. Garbacea, D. Sileo, D. Garrette, D. Hendrycks, D. Kilman, D. Roth, C. D. Freeman, D. Khashabi, D. Levy, D. M. González, D. Perszyk, D. Hernandez, D. Chen, D. Ippolito, D. Gilboa, D. Dohan, D. Drakard, D. Jurgens, D. Datta, D. Ganguli, D. Emelin, D. Kleyko, D. Yuret, D. Chen, D. Tam, D. Hupkes, D. Misra, D. Buzan, D. C. Mollo, D. Yang, D.-H. Lee, D. Schrader, E. Shutova, E. D. Cubuk, E. Segal, E. Hagerman, E. Barnes, E. Donoway, E. Pavlick, E. Rodolà, E. Lam, E. Chu, E. Tang, E. Erdem, E. Chang, E. A. Chi, E. Dyer, E. Jerzak, E. Kim, E. E. Manyasi, E. Zheltonozhskii, F. Xia, F.

Siar, F. Martínez-Plumed, F. Happé, F. Chollet, F. Rong, G. Mishra, G. I. Winata, G. de Melo, G. Kruszewski, G. Parascandolo, G. Mariani, G. X. Wang, G. Jaimovitch-Lopez, G. Betz, G. Gur-Ari, H. Galijasevic, H. Kim, H. Rashkin, H. Hajishirzi, H. Mehta, H. Bogar, H. F. A. Shevlin, H. Schuetze, H. Yakura, H. Zhang, H. M. Wong, I. Ng, I. Noble, J. Jumelet, J. Geissinger, J. Kernion, J. Hilton, J. Lee, J. F. Fisac, J. B. Simon, J. Koppel, J. Zheng, J. Zou, J. Kocon, J. Thompson, J. Wingfield, J. Kaplan, J. Radom, J. Sohl-Dickstein, J. Phang, J. Wei, J. Yosinski, J. Novikova, J. Bosscher, J. Marsh, J. Kim, J. Taal, J. Engel, J. Alabi, J. Xu, J. Song, J. Tang, J. Waweru, J. Burden, J. Miller, J. U. Balis, J. Batchelder, J. Berant, J. Frohberg, J. Rozen, J. Hernandez-Orallo, J. Boudeman, J. Guerr, J. Jones, J. B. Tenenbaum, J. S. Rule, J. Chua, K. Kanclerz, K. Livescu, K. Krauth, K. Gopalakrishnan, K. Ignatyeva, K. Markert, K. Dhole, K. Gimpel, K. Omondi, K. W. Mathewson, K. Chiafullo, K. Shkaruta, K. Shridhar, K. McDonell, K. Richardson, L. Reynolds, L. Gao, L. Zhang, L. Dugan, L. Qin, L. Contreras-Ochando, L.-P. Morency, L. Moschella, L. Lam, L. Noble, L. Schmidt, L. He, L. Oliveros-Colón, L. Metz, L. K. Senel, M. Bosma, M. Sap, M. T. Hoeve, M. Farooqi, M. Faruqui, M. Mazeika, M. Baturan, M. Marelli, M. Maru, M. J. Ramirez-Quintana, M. Tolkiehn, M. Giulianelli, M. Lewis, M. Potthast, M. L. Leavitt, M. Hagen, M. Schubert, M. O. Baitemirova, M. Arnaud, M. McElrath, M. A. Yee, M. Cohen, M. Gu, M. Ivanitskiy, M. Starritt, M. Strube, M. Swędrowski, M. Bevilacqua, M. Yasunaga, M. Kale, M. Cain, M. Xu, M. Suzgun, M. Walker, M. Tiwari, M. Bansal, M. Aminnaseri, M. Geva, M. Gheini, M. V. T, N. Peng, N. A. Chi, N. Lee, N. G.-A. Krakover, N. Cameron, N. Roberts, N. Doiron, N. Martinez, N. Nangia, N. Deckers, N. Muen-nighoff, N. S. Keskar, N. S. Iyer, N. Constant, N. Fiedel, N. Wen, O. Zhang, O. Agha, O. Elbaghdadi, O. Levy, O. Evans, P. A. M. Casares, P. Doshi, P. Fung, P. P. Liang, P. Vicol, P. Alipoormolabashi, P. Liao, P. Liang, P. W. Chang, P. Eckersley, P. M. Htut, P. Hwang, P. Miłkowski, P. Patil, P. Pezeshkpour, P. Oli, Q. Mei, Q. Lyu, Q. Chen, R. Banjade, R. E. Rudolph, R. Gabriel, R. Habacker, R. Risco, R. Millièrre, R. Garg, R. Barnes, R. A. Saurous, R. Arakawa, R. Raymaekers, R. Frank, R.

- Sikand, R. Novak, R. Sitelew, R. L. Bras, R. Liu, R. Jacobs, R. Zhang, R. Salakhutdinov, R. A. Chi, S. R. Lee, R. Stovall, R. Teehan, R. Yang, S. Singh, S. M. Mohammad, S. Anand, S. Dillavou, S. Shleifer, S. Wiseman, S. Gruetter, S. R. Bowman, S. S. Schoenholz, S. Han, S. Kwatra, S. A. Rous, S. Ghazarian, S. Ghosh, S. Casey, S. Bischoff, S. Gehrman, S. Schuster, S. Sadeghi, S. Hamdan, S. Zhou, S. Srivastava, S. Shi, S. Singh, S. Asaadi, S. S. Gu, S. Pachchigar, S. Toshniwal, S. Upadhyay, S. S. Debnath, S. Shakeri, S. Thormeyer, S. Melzi, S. Reddy, S. P. Makini, S.-H. Lee, S. Torene, S. Hatwar, S. Dehaene, S. Divic, S. Ermon, S. Biderman, S. Lin, S. Prasad, S. Piantadosi, S. Shieber, S. Mishnerghi, S. Kiritchenko, S. Mishra, T. Linzen, T. Schuster, T. Li, T. Yu, T. Ali, T. Hashimoto, T.-L. Wu, T. Desbordes, T. Rothschild, T. Phan, T. Wang, T. Nkinyili, T. Schick, T. Kornev, T. Tunduny, T. Gerstenberg, T. Chang, T. Neeraj, T. Khot, T. Shultz, U. Shaham, V. Misra, V. Demberg, V. Nyamai, V. Raunak, V. V. Ramasesh, v. uday prabhu, V. Padmakumar, V. Srikumar, W. Fedus, W. Saunders, W. Zhang, W. Vossen, X. Ren, X. Tong, X. Zhao, X. Wu, X. Shen, Y. Yaghoobzadeh, Y. Lakretz, Y. Song, Y. Bahri, Y. Choi, Y. Yang, Y. Hao, Y. Chen, Y. Belinkov, Y. Hou, Y. Hou, Y. Bai, Z. Seid, Z. Zhao, Z. Wang, Z. J. Wang, Z. Wang, and Z. Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. URL: <https://openreview.net/forum?id=uyTL5Bvosj> (cited on pages 89, 96).
- [109] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: an instruction-following LLaMA model, 2023. URL: https://github.com/tatsu-lab/stanford_alpaca (cited on pages 86, 163).
- [110] Together Computer. RedPajama: an open source recipe to reproduce LLaMA training dataset, 2023. URL: <https://github.com/togethercomputer/RedPajama-Data> (cited on pages 84, 88, 89, 150, 162).
- [111] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E.

- Grave, and G. Lample. LLaMA: open and efficient foundation language models. 2023. arXiv: [2302.13971](#) (cited on pages [88](#), [89](#), [162](#)).
- [112] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: an open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023. URL: <https://openreview.net/forum?id=ehfRiFOR3a> (cited on page [100](#)).
- [113] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023. DOI: [10.18653/v1/2023.ac1-1ong.754](#) (cited on page [86](#)).
- [114] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, and Y. Liang. Describe, explain, plan and select: interactive planning with llms enables open-world multi-task agents. In *Advances in Neural Information Processing Systems*, 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/6b8dfb8c0c12e6fafc6c256cb08a5ca7-Abstract-Conference.html (cited on page [100](#)).
- [115] H. Yang, S. Yue, and Y. He. Auto-GPT for online decision making: Benchmarks and additional opinions. 2023. arXiv: [2306.02224](#) (cited on page [100](#)).
- [116] S. Zwane, D. Hadjivelichkov, Y. Luo, Y. Bekiroglu, D. Kanoulas, and M. P. Deisenroth. Safe trajectory sampling in model-based reinforcement learning. In *19th IEEE International Conference on Automation Science and Engineering*, 2023. DOI: [10.1109/CASE56687.2023.10260496](#) (cited on page [21](#)).
- [117] J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, M. Lai, A. Mavalankar, R. Steigerwald, C. Apps, Y. Aytar, S. M. E. Bechtle, F. Behbahani, S. C. Y. Chan, N. Heess, L. Gonzalez, S. Osindero, S. Ozair, S. Reed, J. Zhang, K. Zolna, J. Clune, N. D. Freitas, S. Singh, and T. Rocktäschel. Genie: generative interactive environments. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

URL: <https://proceedings.mlr.press/v235/bruce24a.html> (cited on page 104).

- [118] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering diverse domains through world models. 2024. arXiv: 2301.04104 (cited on pages 27, 28).
- [119] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The 12th International Conference on Learning Representations*, 2024. URL: <https://openreview.net/forum?id=VtmBAGCN7o> (cited on page 100).
- [120] Z. Jiang, Y. Xu, N. Wagener, Y. Luo, M. Janner, E. Grefenstette, T. Rocktäschel, and Y. Tian. H-GAP: humanoid control with a generalist planner. In *The 12th International Conference on Learning Representations*, 2024. URL: <https://openreview.net/forum?id=LYG6tB1EX0> (cited on page 21).
- [121] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang. AgentBench: evaluating LLMs as agents. In *The 12th International Conference on Learning Representations*, 2024. URL: <https://openreview.net/forum?id=zAdUB0aCTQ> (cited on page 100).
- [122] S. N. T. Zwane, D. G. Cheney, C. C. Johnson, Y. Luo, Y. Bekiroglu, M. Killpack, and M. P. Deisenroth. Learning dynamic tasks on a large-scale soft robot in a handful of trials. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2024 (cited on page 21).
- [123] S. Garg, A. Singh, S. Singh, and P. Chopra. IPO: Your Language Model is Secretly a Preference Classifier. 2025. arXiv: 2502.16182 [cs] (cited on page 102).
- [124] W. Foundation. Wikimedia downloads. URL: <https://dumps.wikimedia.org> (cited on pages 84, 150).

- [125] Lichess Developers. Lichess. URL: <https://lichess.org/> (cited on pages 82, 154).
- [126] Playwright. URL: <https://github.com/microsoft/playwright> (cited on page 144).
- [127] The LCZero Authors. LeelaChessZero. URL: <https://github.com/LeelaChessZero/lc0> (cited on pages 87, 161).
- [128] Computer chess rating lists. URL: <https://www.computerchess.org.uk/ccrl/> (cited on page 83).
- [129] Lichess chess opening names. URL: <https://github.com/lichess-org/chess-openings> (cited on page 95).
- [130] Python-chess: a chess library for Python. URL: <https://github.com/niklasf/python-chess> (cited on pages 84, 159).
- [131] ShareGPT. URL: https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered (cited on pages 88, 162).

Appendix A

Supplements for Chapter 2

A.1 Implementation Details

Our TD3-C TD3-BC and TD3 implementation are based on the JAX TD3 implementation in Acme [79]. The architectures used by the different algorithms are kept the same: both the policy and the critic network are LayerNormMLPs¹ with 2 hidden layers of size 256 and ELU activation. The LayerNorm layer is inserted after the first linear layer and is followed by tanh activation. The last layer of the policy network is initialized with small weights. This architecture was found to be superior compared to the original architecture used by the TD3 paper for online learning. For simplicity and consistent comparison, we do not perform observation normalization in TD3-BC, which was shown in the original paper to boost performance but unnecessary for achieving good performance. Other hyperparameters are kept the same as in the original TD3 implementation. We found that clipping the gradient of the critic with respect to the policy action improves stability. This is implemented in Acme’s DDPG and D4PG implementation but is absent in the original TD3 implementation and the Acme implementation. We clip the gradient from the critic to the policy action to have unit norm in our TD3-C implementation.

Below is a list of hyper-parameters.

¹See <https://github.com/deepmind/acme/blob/master/acme/agents/jax/td3/networks.py>

Table A.1: TD3 and TD3-BC hyper-parameters

Hyperparameter	Value
optimizer	Adam
policy learning rate	3e-4
critic learning rate	3e-4
target network update rate τ	5e-3
delay	2
maximum replay size	1e6
minimum replay size	1000
batch size	256
exploration noise stddev.	0.1
target noise stddev.	0.2
target noise clipping	0.5
TD3-BC behavior cloning α	2.5

Table A.2: TD3-C hyper-parameters

Hyperparameter	Value
ϵ	1e-5
clipping	yes

A.2 Other choices of RL algorithms

Thus far, our analysis has built upon the TD3 and TD3-BC algorithms. While this is a convenient choice for investigating the compositions of online and offline data, an unintended effect is that the conclusions we draw from the experiments may not hold for other choices of RL algorithms. Therefore, we conducted additional experiments to provide hints on whether our findings on the effect of online data composition extend to other algorithms.

We first consider using CQL for both offline pretraining [46] and online finetuning but vary how data is sampled during online finetuning, similar to the investigation we performed in section 3.3.3. Table A.3 summarizes the results for using the CQL algorithm but with different ways of utilizing the offline data. Similar to our findings for TD3-BC, online finetuning with offline algorithms such as CQL improves slowly, but the performance can be significantly improved by discarding offline data during online finetuning.

Table A.3: Effect of online data buffer for CQL. We consider both offline training and online finetuning using CQL but using different ways of initializing the online replay buffer. Pre-filling the online replay buffer with the offline dataset results in worse final performance compared to using only online experiment in the replay buffer.

Dataset	CQL (offline init.)	CQL (online only)
ant-m-v2	109.3 ± 4.93	117.3 ± 3.04
ant-mr-v2	108.7 ± 5.63	111.2 ± 4.30
halfcheetah-m-v2	50.95 ± 0.47	71.1 ± 2.41
halfcheetah-mr-v2	51.56 ± 1.74	65.5 ± 4.68
hopper-m-v2	74.7 ± 6.73	98.7 ± 4.50
hopper-mr-v2	98.3 ± 2.63	92.7 ± 13.0
walker2d-m-v2	82.8 ± 1.79	92.8 ± 6.53
walker2d-mr-v2	87.7 ± 4.44	96.7 ± 5.89
locomotion-total-v2	664.0	746.0

We also perform similar experiments with the implicit Q-learning algorithm (IQL) [81], using different ratios of online data during online training. Figure A.1 shows the results for IQL finetuning on AntMaze with different online data ratios. The result is consistent with our findings with TD3-BC: when finetuning with offline RL algorithms, changing the ratio of online data provides benefits for improving the online sample efficiency without sacrificing stability across all AntMaze datasets we consider except for `antmaze-umaze-diverse-v2`, where we found that IQL diverges after online finetuning.

A.3 Comparison with RLPD

This section compares our results with RLPD introduced by Ball et al. [95]. RLPD is an online RL algorithm that incorporates offline prior data during finetuning. RLPD is considerably different from TD3, and the most noticeable differences between RLPD and TD3 are

1. RLPD uses the REDQ as the base RL algorithm, which is an ensemble of critics (ten in our experiments) as opposed to two used by TD3. It also, by default, uses a higher Update-To-Data (UTD) ratio, which is the number of gradient steps per environment step. A higher UTD ratio significantly

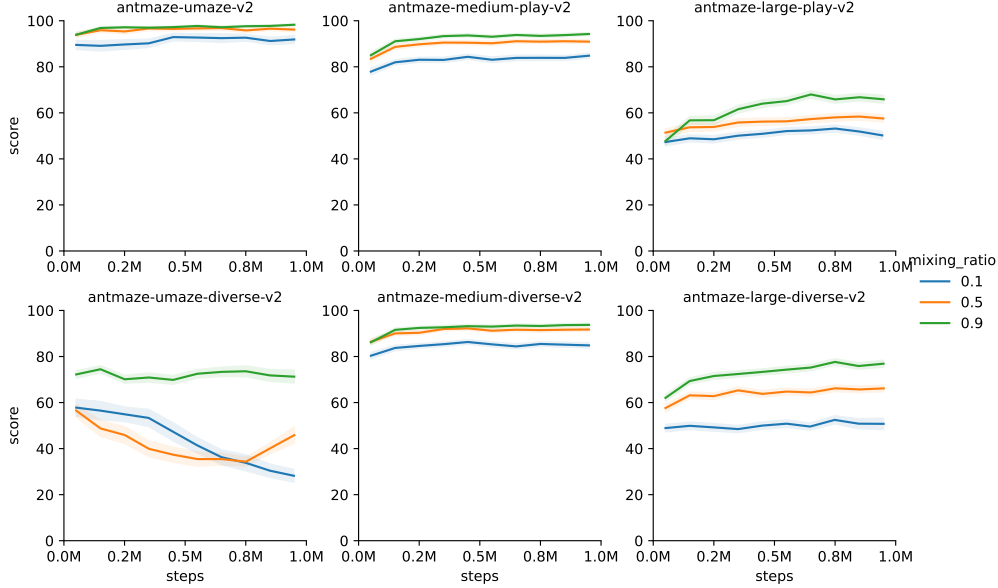


Figure A.1: Performance of IQL using fixed ratio replay on AntMaze. We consider finetuning using IQL for the AntMaze tasks. Finetuning using a larger online ratio (mixing ratio in the figure) allows more online improvement.

improves sample efficiency.

2. RLPD does not perform offline pretraining. Ball et al. [95] found that RLPD performs poorly offline.
3. RLPD uses a fixed ratio for sampling the online and offline experience, an approach we investigated similarly in section 3.3.3.

The results are summarized in table A.4. We found that using a high UTD ratio improves RLPD’s performance significantly. However, with a UTD ratio of 1, RLPD underperforms TD3, TD3-BC, and TD3-C, likely due to the lack of offline pretraining.

A.4 Comparison to prioritized balanced replay

For our analysis in section 3.3, we considered three ways of combining samples from online and offline experiences during finetuning. These approaches are not *adaptive* as the sampling strategy is fixed a priori and not adjusted adaptively

Table A.4: Comparison with RLPD. We compare the finetuning result using TD3, TD3-BC, and TD3-C with RLPD [95], an online RL algorithm that leverages offline data for improved sample efficiency. RLPD with a high update-to-data ratio (UTD) of 20 performs better. However, if we keep the UTD ratio consistent between RLPD and our TD3 variants, we found that the TD3 algorithms perform better.

	RLPD (UTD=1)	RLPD (UTD=20)	TD3	TD3-BC	TD3-C
ant-mr-v2	111.27±9.08	148.98±0.95	122.91±15.16	132.34±2.92	125.50±5.19
ant-m-v2	122.57±12.18	152.97±0.93	104.71±8.48	132.99±1.43	119.34±7.90
halfcheetah-mr-v2	64.12±0.76	88.20±2.90	80.98±3.61	62.35±1.41	73.52±2.17
halfcheetah-m-v2	62.54±4.78	92.13±0.77	82.47±2.70	62.88±1.25	75.93±3.28
hopper-mr-v2	57.93±39.49	86.07±20.95	94.52±25.10	102.75±2.43	102.38±3.01
hopper-m-v2	93.64±12.35	99.86±15.86	90.91±33.52	98.80±9.72	90.58±20.91
walker2d-mr-v2	76.38±21.34	116.46±3.16	98.94±10.38	104.16±4.66	105.35±11.01
walker2d-m-v2	98.19±4.91	118.54±2.01	85.19±19.55	97.48±3.45	97.54±11.88
locomotion-total-v2	686.64	903.23	760.63	793.75	790.14

depending on the current policy. Lee et al. [83] introduced a prioritized balanced replay approach that selects samples from the offline dataset based on the on-policy with respect to the finetuning policy. Specifically, let B^{on} denote a dataset of online transitions collected during finetuning and B^{off} denote a dataset of existing offline transitions. Lee et al. [83] proposes to sample a transition $(s, a, s') \in B^{\text{on}} \cup B^{\text{off}}$ proportional to the density ratio

$$w(s, a) = \frac{d^{\text{on}}(s, a)}{d^{\text{off}}(s, a)},$$

where $d^{\text{on}}(s, a)$ and $d^{\text{off}}(s, a)$ are stationary distributions of state-action pairs from the offline and online buffers. The density ratio $w(s, a)$ can be approximated by a parameterized neural network trained with a likelihood ratio estimation method as suggested by Sinha et al. [55]. This helps ensure that near-on-policy samples are drawn more frequently for better value propagation. Lee et al. [83] demonstrates the effectiveness of a balanced replay when combined with offline pretraining using CQL and online finetuning with SAC, a set-up similar to our use of TD3-BC and TD3.

We compare TD3 with the Balanced Replay approach (Off2On). To understand the effectiveness of the prioritized balanced replay alone, we also include a baseline that uses CQL for offline training and SAC [34] for online finetuning (without offline data) to isolate the effect of the balanced replay. The results are

Table A.5: Comparison with prioritized balanced replay. The table illustrates the difference between: **CQL→SAC**: using CQL for offline pretraining and SAC for online finetuning without using any offline data. **Off2On**: using prioritized balanced replay. **TD3-BC**: TD3-BC for offline and online learning. We found that using TD3-BC for finetuning without any offline data allows us to achieve similar performance as balanced replay in many datasets except HalfCheetah where we found previously that finetuning TD3 performs the best.

Dataset	CQL→SAC	Off2On	TD3-BC
ant-m-v2	130.6±8.62	131.7±13.5	132.99±1.43
ant-mr-v2	117.8±11.7	127.6±10.5	132.34±2.92
halfcheetah-m-v2	93.3±2.79	93.7±3.2	62.88±1.25
halfcheetah-mr-v2	84.5±1.82	85.5±2.4	62.35±1.41
hopper-m-v2	92.7±34.8	98.7±18.5	98.8±9.72
hopper-mr-v2	97.7±17.2	91.8±21.8	102.75±2.43
walker2d-m-v2	83.2±20.4	95.8±15.8	97.48±3.45
walker2d-mr-v2	88.9±24.2	113.1±4.6	104.16±4.66
locomotion-total-v2	788.7	837.9	793.75

summarized in table A.5.

We found that finetuning with offline algorithms such as TD3-BC but removing any offline data during finetuning allows us to mostly achieve similar performance compared to finetuning with online algorithms (SAC) with offline RL initialization for many environments except for HalfCheetah, where we found that any type of policy constraint hurts finetuning sample efficiency.

Prioritized balanced replay with RLPD. We also investigate if the balanced replay scheme is useful beyond their set-up by considering combining it with RLPD, an algorithm that originally used a fixed sampling ratio. The results are shown in fig. A.2. We find that the balanced replay is useful in cases where the offline datasets are significantly sub-optimal (e.g., the random datasets generated by a random uniform policy). In these cases, sampling more offline sub-optimal data generally hurts RLPD’s performance.

Importance of pre-training for prioritized balanced replay. We also evaluate combining balanced replay with RLPD on the Antmaze environments. Figure A.3 compares the performance between RLPD using a fixed 1:1 offline ratio and

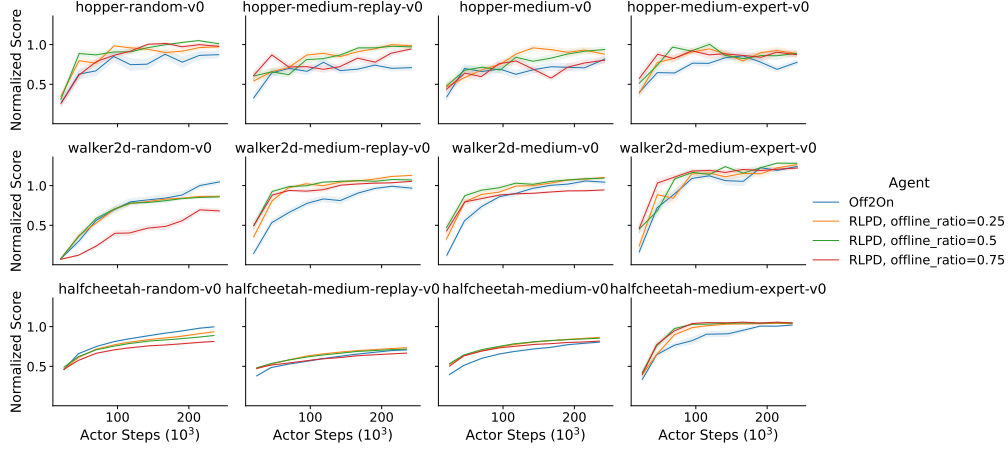


Figure A.2: Effect of different online and offline ratios when using a fixed ratio sampling approach in RLPD. We compare RLPD using different offline ratios and using prioritized balanced replay. We find that prioritized balanced replay to be the most useful when working with random datasets where sampling less offline data is better.

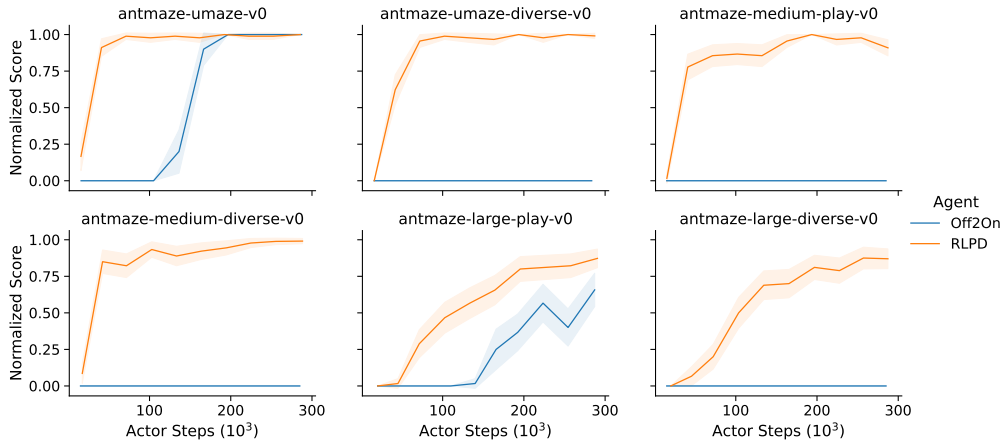


Figure A.3: Comparison between RLPD with fixed data ratio and using balanced replay. We found that RLPD with balanced replay fails to learn a good policy for some of the datasets, whereas using a fixed offline ratio allows RLPD to learn a good policy consistently.

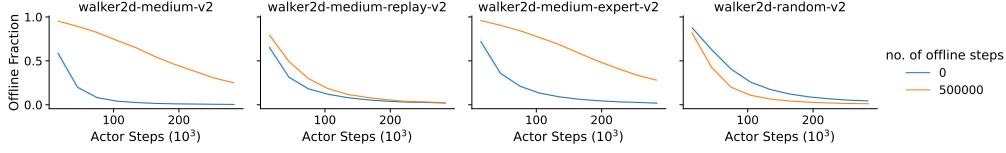


Figure A.4: Balanced replay with or without offline pretraining. We found that not performing when using balanced replay leads to a smaller amount of offline samples to be sampled during finetuning, which leads to performance deterioration on AntMaze as the offline samples contain transitions that provide useful experiences for value learning.

when using a balanced replay. Unlike the results we obtained previously on the locomotion tasks, where utilizing a fixed ratio allows RLDP to learn all tasks quickly, on AntMaze and when using the balanced replay, RLDP fails to make any progress on the AntMaze-diverse tasks. To understand this failure mode on AntMaze, we record the fraction of online transitions sampled by the balanced replay buffer. We find that for all environments, the density ratio estimator has learned to sample only online transitions, effectively discarding all offline datasets. It is not obvious why balanced replay combined with RLDP fails on the AntMaze tasks. We hypothesize that this may be due to the lack of offline pretraining since RLDP is a pure online RL algorithm without any offline pretraining. Therefore, we investigate the effect of not performing any pretraining when using balanced replay. To do this, we first investigate the set-up in [83] online locomotion tasks but without performing any offline pretraining with CQL. We plot the fraction of offline samples encountered in the sampled mini-batches with or without pretraining. The results are shown in fig. A.4. Removing offline pretraining results in the fraction of offline samples converging more quickly towards zero, even when the offline dataset contains high-quality demonstrations.

The experiments above help explain the failure mode we encountered in using a balanced replay with RLDP on AntMaze. The AntMaze datasets contain many useful transitions that include intermediate experience to reach a goal state. These experiences are useful for learning good critic even when they are significantly off-policy with respect to the current policy, which is sub-optimal during initial training from scratch. Using a balanced replay scheme with a policy that has not yet learned to navigate to the goal, however, results in these useful transitions to

be significantly downsampled.

Appendix B

Supplements for Chapter 3

B.1 Hyperparameters

Table B.1 lists the hyperparameters used by OTR and IQL on the locomotion datasets. For Antmaze and Adroit, unless otherwise specified by table B.2 or table B.3, the hyperparameters follows from those used in the locomotion datasets.

The IQL hyperparameters are kept the same as those used in [81]. Note that IQL rescales the rewards in the dataset so that the same set of hyperparameters can be used for datasets of different qualities. Since OTR computes rewards offline, we also apply reward scaling as in IQL. For the locomotion datasets, the rewards are rescaled by $\frac{1000}{\max_return - \min_return}$ while for AntMaze we subtract 2 to the rewards computed by OTR. The reward processing in AntMaze is different from the one used by the original IQL paper (which subtracts 1) since the rewards computed by OTR have a different range.

The squashing function used by OTR is based on the one used in [60]. The AntMaze squashing differs slightly from the one used in locomotion and adroit due to use of an earlier configuration. In practice, this should have minimal effect on the performance.

	Hyperparameter	Value
	Discount	0.99
Network	Hidden layers	(256, 256)
	Dropout	none
	Network initialization	orthogonal
IQL	Optimizer	Adam
	Policy learning rate	$3e^{-4}$, cosine decay to 0
	Critic learning rate	$3e^{-4}$
	Value learning rate	$3e^{-4}$
	Target network update rate	$5e^{-3}$
	Temperature	3.0
	Expectile	0.7
OTR	Episode length T	1000
	Cost function	cosine
	Squashing function	$s(r) = 5.0 \cdot \exp(5.0 \cdot T \cdot r / \mathcal{A})$

Table B.1: OTR hyperparameters for D4RL Locomotion.

	Hyperparameter	Value
IQL	Temperature	10.0
	Expectile	0.9
OTR	Squashing function	$s(r) = 5.0 \cdot \exp(T \cdot r)$

Table B.2: OTR hyperparameters for D4RL Antmaze

	Hyperparameter	Value
Network Architectures	Dropout	0.1
IQL	Temperature	0.5
	Expectile	0.7

Table B.3: OTR hyperparameters for D4RL Adroit

Appendix C

ChessGPT dataset supplements

C.1 Chess notations

C.1.1 Universal Chess Interface (UCI)

The UCI format is widely used for communication between chess engines and user interfaces. It represents chess moves by combining the starting and ending squares of a piece, such as “e2e4” to indicate moving the pawn from e2 to e4. For example, the UCI notation for a full game would be:

```
e2e4 c7c6 g1f3 d7d5 e4d5 c6d5 d2d4 b8c6 c2c4 g8f6 b1c3 c8e6 c4c5 g7g6 c1f4 f8g7  
f1e2 f6e4 e1g1 e6g4 f3e5 g4e2 d1e2 c6e5 c3e4 e5c6 e4d6 e8f8 f1e1 g7d4 f4h6 d4g7  
h6g7 f8g7 d6b5 a7a6 b5c3 d5d4 c3e4 d8d5 a2a3 a8d8 b2b4 h7h6 e2
```

C.1.2 Standard Algebraic Notation (SAN)

SAN (Standard Algebraic Notation) is a widely used notation system in the game of chess for recording and describing moves. It provides a standardized and concise representation of moves that is easily understood by chess players and enthusiasts. In SAN, each move is represented by two components: the piece abbreviation and the destination square. The piece abbreviation is a letter that represents the type of piece making the move, such as “K” for king, “Q” for queen, “R” for rook, “B” for bishop, “N” for knight, and no abbreviation for pawns. The destination square is denoted by a combination of a letter (a-h) representing the column and a number (1-8) representing the row on the chessboard. Additional symbols may be used to indicate specific move types.

The symbol “+” is used to indicate a check, while “#” denotes a checkmate. Castling moves are represented by “O-O” for kingside castling and “O-O-O” for queenside castling.

C.1.3 Portable Game Notation (PGN)

PGN is a widely adopted format for recording chess games. It includes not only the SAN moves but also additional information like player names, event details, and game results. PGN files are human-readable and can be easily shared and analyzed. Here is an example of a PGN representation of a full game:

```
[Event "World Chess Championship"]
[Site "London, England"]
[Date "2023.05.20"]
[Round "1"]
[White "Carlsen, Magnus"]
[Black "Nepomniachtchi, Ian"]
[Result "1/2-1/2"]

1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7 6. Re1 b5 7. Bb3 d6
8. c3 O-O 9. h3 Nb8 10. d4 Nbd7 11. Nbd2 Bb7 12. Bc2 Re8 13. Nf1 Bf8
14. Ng3 g6 15. a4 c5 16. d5 c4 17. Bg5 h6 18. Be3 Nc5 19. Qd2 h5 20. Bg5 Bg7
21. Nh2 Qc7 22. Rf1 Nh7 23. Bh6 Bh8 24. f4 exf4 25. Bxf4 Nf6 26. Rae1 bxa4
27. Nf3 Nfd7 28. Bh6 Ne5 29. Nxe5 Bxe5 30. Rf3 Qb6 31. Kh1 Qxb2 32. Ref1 Re7
33. Bg5 Rd7 34. Bf6 Bxf6 35. Rxf6 a3 36. Nxh5 a2 37. Qh6 gxh5 38. R6f3 h4
39. Rf4 f5 40. Rxf5 Rg7 41. Rh5 1-0
```

C.1.4 Forsyth-Edwards Notation (FEN)

FEN is a notation system used to describe the state of a chess game. It represents the positions of pieces on the chessboard, active color, castling rights, en passant targets, and the half-move and full-move counters. Here is an example of a FEN notation representing the starting position:

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
```

In this FEN notation, each letter represents a piece on the board, with uppercase letters representing white pieces and lowercase letters representing black pieces. The forward slash (“/”) separates ranks, and the number after each rank indicates the number of empty squares. The active color is represented by “w” for white or “b” for black. The castling rights are denoted by “K” (white kingside),

“Q” (white queenside), “k” (black kingside), and “q” (black queenside). The en passant target square is indicated with the corresponding square, or “-” if there is no en passant target. The half-move and full-move counters specify the number of half-moves since the last pawn move or capture and the current move number, respectively.

These different chess formats serve various purposes, from representing individual moves (UCI) to recording entire games (PGN) and describing specific positions (FEN). Understanding and working with these formats is essential for tasks like parsing, analyzing, and exchanging chess game data in different contexts.

C.2 Dataset details

In this section, we offer more details on how data are obtained and processed. This section also presents examples for the final dataset we use for training ChessGPT and ChessCLIP.

C.2.1 Dataset statistics and metrics

In table C.1, we present the dataset statistics breakdown for each data subset, including its raw size, document count, and subset type.

Table C.2 shows the properties of the chess-specific language dataset that we use for training ChessGPT. For these datasets, we computed the average number of words (num. words) per example, character repetition ratio (char. rep. ratio), word repetition ratio (word. rep. ratio.), special character ratio (special char. ratio), stopwords ratio and perplexity of the first shard for each subset in the language dataset. These metrics are based on some of the criteria employed by Bloom [82] in their pre-processing pipelines.

C.2.2 Preprocessing

We preprocess the data sources in three levels of granularity. For sources where existing preprocessed dataset are available, we filter out the subset that contains chess-related information without performing additional preprocessing. For

Table C.1: Dataset statistics

Component	Raw size	Document count	Subset type
Lichess	19.9 GB	17.5 M	Game
Pro-player	0.37 GB	0.44 M	Game
CCRL	3.60 GB	3.00 M	Game
Chess puzzles	1.53 GB	3.19 M	Game
Chess modeling	0.94 GB	1.90 M	Game
C4-Chess	0.59 GB	0.18 M	Language
Pile-Chess	1.10 GB	0.10 M	Language
RedPajama-Chess	5.65 GB	0.52 M	Language
Oscar-Chess	3.13 GB	0.33 M	Language
WikiPedia-Chess	40.3 MB	11.4 K	Language
Chess Blogs	0.59 GB	73.2 K	Language
Chess Books	1.86 GB	8.36 K	Language
Chess Forums	1.05 GB	0.14 M	Language
Annotated Chess Games	0.66 GB	245 K	Mixed
Youtube transcripts	0.98 GB	83.0K	Mixed
GPT-4-Chess	0.95 MB	3.91 K	Conversation
Reddit	0.86 GB	0.41 M	Conversation
Overall	42.8 GB	28.1 M	N/A

Table C.2: Metrics for the language dataset.

dataset	num. words	char. rep. ratio	word rep. ratio	special char ratio	stopwords ratio	perplexity
Chess puzzles	49.6618	0.0104	0.0000	0.3246	0.4771	988.3734
Oscar-Chess	1441.4341	0.0596	0.0499	0.2242	0.4119	665.6499
Pile-Chess	2105.2454	0.0626	0.0205	0.2409	0.4227	497.3883
RedPajama-Chess	1581.5825	0.0532	0.0163	0.2273	0.4218	410.1236
StackExchange-Chess	578.3733	0.0591	0.0816	0.2617	0.4835	520.1257
Wikipedia-Chess	463.4980	0.0876	0.0052	0.2604	0.3079	236.9671
C4-Chess	510.6041	0.0479	0.0082	0.2131	0.4418	548.9471

sources that we retrieve from the Internet, we only parse portions of the HTML that contains information about chess. We implement different parsers for the different sources we consider. As a result, our data preprocessing can be more light-weight compared to previous work that extracts corpora from raw HTML web pages. For PGN games, we use the original PGN but filter out some annotations that are not useful for learning the model.

The different sources contain data in different formats. To facilitate training on all datasets, we preprocess all datasets to have the same JSONL format.

We implement the data-preprocessing step for each source as a separate Apache Beam pipeline which allows us to process the datasets in parallel into multiple shards. We note that a simple Apache Beam pipeline implementation provides no guarantees that data processing will be in the same order as they were read from the files. As a result, running the same pipeline twice will produce a set of shards that are shuffled differently. To provide determinism in our data-processing pipeline, we adopt a deterministic shuffling strategy similar to the implementation in TensorFlow Datasets (TFDS) to ensure reproducible data processing while maintaining scalability.

We initially applied the same data-processing procedure described in [82] for all of the data that we collected. However, we found that the filtering criteria used in [82] can be too aggressive in removing useful examples as many of our data sources include a significant portion of chess notation that does not resemble natural language (e.g., chess puzzles). Therefore, we opted for more light-weight pre-processing and use the processing from [82] only in cases where the text includes a significant portion of natural language description (blogs for example). In addition, for further protection of privacy, we anonymize user names and replace them with terms like 'Human 0' in all conversation-like data, especially in chess forums and Reddit conversational data.

For specific Licenses and dataset cards, refer to our open-source dataset repository: https://huggingface.co/datasets/Waterhorse/chess_data.

In the following few sections, we describe in more details the data we have collected and the processing steps we use for cleaning these datasets.

C.2.3 Online chess websites

We choose around 26 chess websites, including chess.com and lichess.com to scrape chess-related language corpus. We gather a diverse range of chess and language data from various platforms, including blogs, news, and articles. Specifically, we focus on extracting relevant information from several topics including blogs, news, openings, chess terms, forums, and articles. These topics were carefully chosen as they contain valuable texts pertaining to chess background knowledge, news, and also PGN games in some instances. We utilize BeautifulSoup¹ and Playwright[126] to parse HTML pages and locate all the texts and PGNs. We further transfer those PGNs into text which helps us build mixed game-language datasets from these sources. We record details such as the URL, author, title, time, and the actual content of the articles. An example of a from a chess blog article is shown below:

¹<https://www.crummy.com/software/BeautifulSoup/>

Chess Blogs

What Is The Elo Rating System? The Elo rating system measures the relative strength of a player in some games, such as chess, compared to other players. Its creator, Arpad Elo, was a physics professor in the United States and a chess master who worked to improve the way the U.S. Chess Federation measured their players' skill levels. He was a solid chess player himself, as you can see from this game he played against a young Bobby Fischer.

```
[Event "New Western Open"]
[Site "Milwaukee, WI USA"]
[Date "1957.07.04"]
[Round "2"]
[White "Arpad Elo"]
[Black "Robert James Fischer"]
[Result "0-1"]
[EventDate "?"]
[ECO "B93"]
[WhiteElo "?"]
[BlackElo "?"]
[PlyCount "98"]
```

```
1. e4 c5 2. Nf3 d6 3. d4 cxd4 4. Nxd4 Nf6 5. Nc3 a6 6. f4 e5 7. Nf3 Qc7 8. Bd3
Nbd7 9. 0-0 b5 10. Qe1 Bb7 11. a3 g6 12. Qh4 Bg7 13. g4 exf4 14. Bxf4 0-0 15.
Qg3 Ne5 16. Nxe5 dxe5 17. Bxe5 Qc5+ 18. Rf2 Nh5 19. Bd6 Qxc3 20. bxc3 Nxc3 21.
Bxf8 Rxf8 22. hxg3 Bxc3 23. Rb1 Bd4 24. a4 Bc8 25. axb5 axb5 26. Rxb5 Bxc4 27.
Kg2 Bxf2 28. Kxf2 Be6 29. Rc5 Kg7 30. Kf3 Kf6 31. Kf4 Ra8 32. g4 h6 33. g5+
hxg5+ 34. Rxg5 Rh8 35. Rg2 g5+ 36. Kf3 Rh3+ 37. Rg3 Rxg3+ 38. Kxg3 Ke5 39. c3
Bd7 40. Bc4 f6 41. Bd5 Be8 42. c4 Kd4 43. Kg4 Bg6 44. Kf3 Bh5+ 45. Kf2 Bd1 46.
Kg3 Be2 47. c5 Kxc5 48. Be6 Kd4 49. Bf5 Ke3 0-1
...
```

C.2.4 Online Chess forums

We choose 5 chess forums and follow basically similar way with appendix C.2.3 to scrape forum text data. We run the same preprocessing pipeline as in [82].

Stack Exchange. We use the forums Chess Stack Exchange.² We preprocess the Stack Exchange data in the same way as done in RedPajama. Concretely, for each question, we first clean up the text via simple regular expressions and remove some HTML tags in the text. Then we prefix the question with Q: and the answers with A: and then concatenate the questions with all answers. Below is an example of the final forum discussion from Stack Exchange:

Chess Forums (Stack Exchange)

Q: Is there a way to use handicaps in chess to bridge the gap between players of different skill levels? Handicapping is routine in the Japanese game Go (my best game). The basic strength unit is one stone, and a one-stone difference represents a full level of difference in strength. I (about a 1500 player) once asked a 2100 player how much of a handicap she would need to give me so that we would have an equal chance to win. "Probably a knight, maybe more," she answered. I once took a knight handicap against a 2200 player and lost, but it was a much tighter, closer game than one with no handicap. That might suggest that a pawn is equivalent to about 200 points of rating. Apparently handicapping doesn't do much for say, a 50 point difference in strength (you just play and take your chances). But above that, there might be ways to use handicaps. Even giving someone the first move two times out of three (as was earlier done in professional Go) might do something. Or would it? Why hasn't handicapping been done much formally in chess, as in Go?

Reddit. We filter the conversations based on the language, response length, number of emojis, blacklist words, and scores. Below is an example of a reddit conversation:

²<https://chess.stackexchange.com/>

Reddit Conversation data

```
{
  "author": "Human 0",
  "text": "Honest question for those with higher ELOs. First, sorry if
this is a confusing or vague question but I'll try my best to word
it: At what rating, in your opinions, do you find people stop making
`silly` mistakes? I'm at 1500 rapid and 1300 blitz, trying to
improve my play. I found that at these ratings blunders are rare
(assuming adequate time). It seems like mostly people just don't
see/miss tactics or play inaccuracies rather than playing straight
up mistakes/blunders. Do you higher rated elo players feel the same
way? Or do you think the inaccuracies we make/tactics we miss are
quite obvious, the same way I can see a blunder is obvious? Curious
on the perspective.",
  "score": 8,
  "other_data": false,
  "url": "/r/chess/comments/nhh2mn/honest_question_for_those_with_higher_elos/"
}
{
  "author": "Human 1",
  "text": "Im 2200 in blitz on lichess, and of my last 10 blitz games,
around 8 were decided by major blunders. In rapid, the amount of
major blunders decreases a lot, but they are still very common. When
you get higher rated, you will still blunder because you are also
going to be facing higher rated opponents. If I played a 1200, I
would rarely blunder but I blunder very easily against 2300+
people.",
  "score": 22,
  "other_data": false,
  "url": "/r/chess/comments/nhh2mn/honest_question_for_those_with_higher_elos/
gywapvb/"
}
```

C.2.5 Annotated PGN

We collect our annotated PGN from seven sources: Lichess studies, Chess publishing, Megabase, Pgnlib, pathtochessmastery and gameknot.

We keep only games with known finish, i.e., (win, lose or draw). We remove `clk`, `arrow`, `evp` annotations from the comments. We also remove Emojis from the comments. Afterward, each PGN is considered a single string of text that is used for downstream training. We conduct language filtering using `fastText`³ in ChessCLIP preprocessing. And we conduct the same preprocessing as we do in Lichess database in ChessGPT training preprocessing.

Lichess studies. Lichess Study provides a rich collection of annotated PGNs. The annotations are embedded in PGNs, explaining the insight of the moves. Users can conveniently search for studies based on keywords or specific topics like Sicilian Defense, Puzzles, or d5. To enhance the searching process, we collect a comprehensive set of 54 popular keywords. Our implementation leverages Selenium’s [107] built-in functions to efficiently parse HTML pages and simulate the searching process. Additionally, we use the Lichess APIs⁴ to request for PGN games associated with a specific study ID.

Chess Publishing. This contains commercial annotated PGNs from chesspublishing.com,⁵ so we do not open source this source of data.

Megabase. This contains commercial annotated PGNs from Megabase2023⁶, so we do not open source this source of data.

PGNLib. We collect annotated PGN from PGNLib.⁷

Path To Chess Mastery. We collect annotated PGNs from Path to Chess Mastery.⁸

³<https://fasttext.cc/docs/en/language-identification.html>

⁴<https://lichess.org/api>

⁵<https://www.chesspublishing.com/content/>

⁶https://shop.chessbase.com/en/products/mega_database_2023

⁷<https://www.angelfire.com/games3/smartbridge/>

⁸<https://www.pathtochessmastery.com/>

Gameknot. We use Selenium [107] to scrape annotated PGNs from Game-Knot.⁹

Below is an example of an annotated PGN game:

Annotated PGN

```
[Event "All about the Traxler Counter-Attack: Why to play Traxler instead of a
passive move"]
[Site "https://lichess.org/study/WLyfoXTJ/xdA6LWme"]
[Date "????.??.??"]
[Round "?"]
[White "?"]
[Black "?"]
[Result ""]
[Annotator "https://lichess.org/@/Dat_Das"]
[ECO "C57"]
[Opening "Italian Game: Two Knights Defense, Fried Liver Attack"]
[UTCDate "2017.10.13"]
[UTCTime "16:33:43"]
[Variant "Standard"]

{ Hello everyone, please click the little heart to show that this
study was helpful to you, to spread the word and to show your
appreciation. } 1. e4 e5 2. Nf3 Nc6 3. Bc4 Nf6 4. Ng5 { You may wonder
why you should play Bc5 instead of d5. This is just to show you
exactly what white is trying to do. } 4... d5 5. exd5 { It seems
you'll win the exchange. } 5... Nxd5 ( 5... Na5 { This is the best
defense if you do play d5. } 6. Bb5+ c6 7. dxc6 bxc6 8. Qf3 ) 6. Nxf7
{ White has a sacrifice of their own. This is the fried liver attack.
} 6... Kxf7 7. Qf3+ Ke6 8. Nc3 { The knight is pinned. } 8... Nb4 9.
a3 Nxc2+ { Sacrificing a rook. } 10. Kd1 Nxa1 11. Nxd5 { A move with
potential for a dangerous discovered attack } 11... Qh4 12. Nxc7+ {
Double check. } 12... Kd7 13. Qf7+ Qe7 14. Nxa8 Qxf7 15. Bxf7 { And
Black's king position is destroyed, and white is a pawn up. White's
knight may be hanging, but so is black's. }
```

⁹https://gameknot.com/list_annotated.pl?u=all

C.2.6 Existing datasets

We extract all chess-related language corpus from existing dataset C4 [52], the Pile [44], Oscar [40], Wikipedia [124], and RedPajama [110]. To extract chess-related language corpus, we first filter language corpus that contains the keyword “chess”. We further utilize the `deberta-v3-base-tasksource-nli` model,¹⁰ which can conduct zero-shot classification based on classes specified by users. We set two classes: chess-related and non-chess-related for this model. We input the first 2000 characters of each text example to the model and set the threshold as 0.5. Since these datasets are available in processed format, we do not perform any additional preprocessing.

Below are examples of filtered corpus from the existing datasets:

C4

Nothing improves your chess more than playing long time control tournament events. The Irish championships are being held in Dublin next weekend. There are a lot of events to choose from! We have a message encouraging us to play from the Irish Chess Union Chairman. All members of our club are registered (by the club) with the ICU. You are eligible to play and should consider playing!

Pile

Q: What is the theory behind center control? Center control is an important aspect of playing chess, most openings are built around controlling the center, but why? Is center control really that important for winning a game?

¹⁰<https://huggingface.co/sileod/deberta-v3-base-tasksource-nli>

RedPajama

... Mark Dvoretsky - strong player and fantastic coach Mark Israilewitsch Dvoretsky was born on 9th December 1947 in Moscow. After finishing his studies of Mathematics and Economics in 1972 Dvoretsky focused on a career as chess trainer and among other things worked for Botvinnik's school of chess. As a young player Dvoretsky achieved a number of notable successes: in 1973 he won the Championship of Moscow and in 1974 he finished fifth at the USSR-Championship in Leningrad. One year later, in 1975, he won the B-tournament in Wijk aan Zee. But he soon decided to focus on his career as a chess trainer Dvoretsky has trained countless strong players, and among his regular students are well-known players such as Valery Chechov, Nana Alexandria, Sergei Dolmatov, Alexej Dreev and Artur Jussupow. Among the players who occasionally trained with Dvoretsky are Garry Kasparov, Viswanthan Anand, Veselin Topalov, Evgeny Bareev, Viktor Bologan, Loek van Wely and lots of others. One training method of Dvoretsky was to play selected positions with both colors against his students - and he often surprised his students by winning the same position with Black and with White. Dvoretsky was an International Master and FIDE Senior Trainer. He published a number of textbooks, sometimes with Artur Jussupow as co-author. ChessBase published a digital version of his "Endgame Manual". Dvoretsky was a firm part of chess life in Moscow and popular guest at chess events all over the world. Russian Chess Federation ...

Oscar

If one wishes to learn chess from some of the greatest players in the world, but does not live in greater New York, then online lessons may be what he or she is looking for. Many of our coaches are experienced in teaching both group and solo online lessons. Online lessons are orchestrated via Skype using an online chess program.

C.2.7 YouTube transcripts dataset

To collect the YouTube transcripts dataset, we first gather 19 keywords and utilize scrapetube¹¹ to gather all related channels. We extract all videos from these channels and use the same `deberta-v3-base-tasksource-nli` model mentioned in appendix C.2.6 to filter all video transcripts and also the corresponding videos that are not relevant to chess. It is fairly easy to extract transcripts from videos and the main difficulty is to extract the FEN chessboard representations from videos. Here we mainly utilize two steps to extract specific FEN from chess videos.

Extract Chess Board from videos

The first step is to extract the chessboard from videos. We utilize GLIP [84], a zero-shot language-based detection model. By setting the prompt as 'Chess, chessboard', we can extract chess-related detection bounding boxes from videos. We conduct further filterings such as length-width ratio filtering to guarantee it is a valid chessboard bounding box in most cases, which will be processed based on the second step.

Convert chess board image to FEN text format

Our second step involves converting the chessboard image into FEN. FEN format serves as a great way to describe the current chess board state in text-based format. The pipeline that converts the chess board image to FEN format includes three main procedures – chess board decomposition, piece classification, and the prediction of which player is the next turn.

Chess board decomposition. The aim of this step is to break down a whole chess board into 64 small tiles (8 rows and 8 columns), where each tile contains only one chess piece. To achieve this, we initially convert the RGB image to grayscale, preparing for the line detection process. Subsequently, we make two convolutional kernels to find horizontal and vertical gradients.¹² The Hough

¹¹<https://github.com/dermasmid/scrapetube>

¹²https://github.com/Elucidation/tensorflow_chessbot/blob/master/tensorflow_compvision.ipynb

Table C.3: Training hyperparameters for chess board to FEN

Hyperparameter	Value
Batch size	32
Number of epochs	10
Learning rate	0.001
Optimizer	SGD
Momentum	0.9

Transform is then applied to detect vertical and horizontal lines and filter out seven vertical and seven horizontal lines that fit the demand. Finally, we divide the board into 64 tiles by having the position of the 14 lines.

Piece classification. To facilitate model training and evaluation, we employ an open-source chess image dataset on Kaggle¹³ which contains 80k training images and 20k testing images. Each tile can be classified into one of the 13 categories (p, r, b, n, k, q, P, R, B, N, K, Q, and space) which is detailed in appendix C.1.4. We implement a model in PyTorch which uses a pre-trained ResNet18 [23] due to its well-established performance in image classification tasks. To adapt the model to our specific problem, we replaced the original fully connected layer with a new layer consisting of 13 output neurons, corresponding to the 13 pieces categories. We train the model on 40000 images with hyperparameters from table C.3.

After the training process, we evaluate the model on a test dataset with 20K images (equivalent to 128k tiles). Table C.4 summarizes the the final accuracy of each category.

Prediction of next turn. As FEN format also includes the prediction of the next turn which is indicated by “w” for white, and “b” for black, the prediction of the next turn is accomplished by analyzing the main color of each tile. We use Colorthief, a library for grabbing the color palette from images, to extract the main color from each tile since the background color of a tile will be highlighted if a move is played on that tile. Hence, we find the highlighted tile by analyzing the tile color to know who is the current player and naturally infer who is the

¹³<https://www.kaggle.com/datasets/koryakinp/chess-positions>

Table C.4: Validation accuracy of each chess piece

Color	Piece	Accuracy (%)
Black	p awn	99.98
	r ook	99.99
	k night	99.98
	b ishop	99.98
	q ueen	100.00
	k ing	99.97
White	P awn	100.00
	R ook	99.98
	k Night	99.98
	B ishop	99.99
	Q ueen	99.95
	K ing	99.98
Space		100.00

next turn.

Finally, we also provide a final certainty percentile to evaluate to what extent the generated FEN is correct by calculating the product of the accuracy of the 64 tiles.

C.2.8 Lichess dataset

We collect five months of Lichess dataset from the Lichess database [125]: 2013-02, 2014-02, 2015-02, 2016-02, and 2017-02. There are much more data available and we leave more game data for future work.

Below is an example of PGN from the Lichess database.

Lichess

```
[Event "Rated Bullet tournament https://lichess.org/tournament/yc1WW20x"]
[Site "https://lichess.org/PpwPOZMq"]
[Date "2017.04.01"]
[Round "-"]
[White "Abbot"]
[Black "Costello"]
[Result "0-1"]
[UTCDate "2017.04.01"]
[UTCTime "11:32:01"]
[WhiteElo "2100"]
[BlackElo "2000"]
[WhiteRatingDiff "-4"]
[BlackRatingDiff "+1"]
[WhiteTitle "FM"]
[ECO "B30"]
[Opening "Sicilian Defense: Old Sicilian"]
[TimeControl "300+0"]
[Termination "Time forfeit"]
1. e4 { [%eval 0.17] } 1... c5 { [%eval 0.19] }
2. Nf3 { [%eval 0.25] } 2... Nc6 { [%eval 0.33] }
3. Bc4 { [%eval -0.13] } 3... e6 { [%eval -0.04] }
4. c3 { [%eval -0.4] } 4... b5? { [%eval 1.18] }
5. Bb3?! { [%eval 0.21] } 5... c4 { [%eval 0.32] }
6. Bc2 { [%eval 0.2] } 6... a5 { [%eval 0.6] }
7. d4 { [%eval 0.29] } 7... cxd3 { [%eval 0.6] }
8. Qxd3 { [%eval 0.12] } 8... Nf6 { [%eval 0.52] }
9. e5 { [%eval 0.39] } 9... Nd5 { [%eval 0.45] }
10. Bg5?! { [%eval -0.44] } 10... Qc7 { [%eval -0.12] }
11. Nbd2?? { [%eval -3.15] } 11... h6 { [%eval -2.99] }
12. Bh4 { [%eval -3.0] } 12... Ba6? { [%eval -0.12] }
13. b3?? { [%eval -4.14] } 13... Nf4? { [%eval -2.73] } 0-1
```

C.2.9 Pro-player dataset

We collect our pro-player dataset from PGN Mentor.¹⁴ Below is an example of PGN in the Pro-player dataset:

Pro-player

```
[Event "URS-ch34"]
[Site "Tbilisi"]
[Date "1966.??.?"]
[Round "9"]
[White "Bronstein, David I"]
[Black "Suetin, Alexey S"]
[Result "1/2-1/2"]
[WhiteElo ""]
[BlackElo ""]
[ECO "B97"]

1.e4 c5 2.Nf3 d6 3.d4 cxd4 4.Nxd4 Nf6 5.Nc3 a6
6.Bg5 e6 7.f4 Qb6 8.Qd2 Qxb2
9.Rb1 Qa3 10.Bxf6 gxf6 11.Be2 Bg7 12.0-0 Nc6
13.Nxc6 bxc6 14.Rb3 Qc5+ 15.Kh1 f5
16.exf5 exf5 17.Na4 Qd4 18.Qxd4 Bxd4 19.Rd1 Bf2
20.Rxd6 0-0 21.Nb6 Bxb6 22.Rxb6 Re8
23.Bf1 Be6 24.Kg1 Bxa2 25.Rxa6 Rxa6 26.Bxa6 Bd5
27.Kf2 Re4 28.g3 Bc4 29.Rxc6 Re2+
30.Kg1 Bxa6 31.Rxa6 Rxc2 32.Ra5 Kg7 33.Rxf5 Kg6
34.Rg5+ Kf6 1/2-1/2
```

C.2.10 Chess books

We select 100 chess-related keywords and search for all related chess books (around 9K books) on the online PDF library. Because of the legal issues about books' copyright, we choose not to open-source this source of data. Instead, we only open source the list of books we use.

¹⁴<https://www.pgnmentor.com/>

Chess Books

The following illustrative game is apparently complicated, but it is this in its motives\only.\nIn reality it is the fight against White's e4 pawn, which dominates. Shoosmith-\nNimzowitsch, Ostend, 1907. 1.d4 Nf6 2.c4 d6 3.Nf3 Nbd7 4.Nc3 e5 5.e4 Be7 6.Bd3\n0-0 7.0-0 exd4! (if 7...Re8, then 8.05 and Black will be\ncramped for a long time. For example, 7...Re8 8.45 NcS\n9.Be3 Nxd3 10.Qxd3 Nd7 11.b4 a5 12.43, etc) 8.Nxd4\nRe8 9.b3 Ne5 10.Bc2 a6 (this advance will soon be\nintelligible) 11.Bb2 Bd7 12.3 Bf8 13.f4 Ng6 14.Qf3 c6\n15.Rae1 b5 (now the situation is clear: Black keeps an\neye on White's e-pawn and seeks ...

C.2.11 CCRL

We collect our CCRL dataset without comments from the official website¹⁵ for three settings of time control: CCRL BLITZ, CCRL 40/2 FRC and CCRL 40/15.

```

CCRL

[Event "CCRL 40/15"]
[Site "CCRL"]
[Date "2022.01.08"]
[Round "806.6.381"]
[White "Stockfish 060122 64-bit"]
[Black "Dragon by Komodo 2.6 64-bit"]
[Result "1/2-1/2"]
[ECO "D30"]
[Opening "Queen's gambit declined"]
[PlyCount "115"]
[WhiteElo "3505"]
[BlackElo "3480"]

1. d4 {book} d5 {book} 2. c4 {book} e6 {book} 3. Nf3 {book} Nf6 {book}
4. g3 {book} a6 {book} 5. c5 {book} b6 {book} 6. cxb6 {+0.23/33 28s}
c5 {-0.23/30 40s} 7. Bg2 {+0.24/30 11s} cxd4 {-0.15/29 17s} ...
58. Kc2 {+0.00/101 20s, Draw by 3-fold repetition}
1/2-1/2

```

C.2.12 Chess puzzles

We collect our chess puzzles from the Lichess puzzle dataset.¹⁶

The original data format is in CSV format with key data, such as puzzle FEN and puzzle answer. We leverage some language templates to transfer the CSV as natural language text. Below is an example from the final chess puzzles dataset:

```

Chess puzzles

Try your hand at this chess puzzle. The board's FEN is
1r4k1/4nppp/8/4Pb2/8/1P5P/r1PR4/3R3K w - - 0 27, and you need to
determine the optimal move for the player. This puzzle focuses on
backRankMate,endgame,mate,mateIn2,short, and the solutions are
provided in both SAN format as d2d8,b8d8,d1d8 and UCI format
as 27. Rd8+ Rxd8 28. Rxd8#.

```

¹⁵<https://ccrl.chessdom.com/ccrl/4040/>

¹⁶<https://database.lichess.org/#puzzles>

C.2.13 Chess modeling dataset

We design 11 modeling tasks to generate data:

1. Given PGN, generate FEN representation.
2. Given a list of UCI moves, generate FEN representation.
3. Given FEN and a UCI move, transfer the move to SAN format.
4. Given FEN and a SAN move, transfer the move to UCI format.
5. Given FEN, generate an ASCII board.
6. Given FEN and a UCI move, generate the next FEN.
7. Given FEN and a SAN move, generate the next FEN.
8. Given FEN, generate all legal moves in SAN format.
9. Given FEN, generate all legal moves in UCI format.
10. Given PGN, generate all legal moves in SAN format.
11. Given PGN, generate all legal moves in UCI format.

To generate the synthetic modeling dataset for these tasks, we utilize PGN data extracted from the first 1 million games of the Lichess database dump from March 2017. In order to encompass a wider range of ELO ratings, we divide the Elo-rating into 9 intervals: 0-1000, 1000-1200, 1200-1400, 1400-1600, 1600-1800, 1800-2000, 2000-2200, 2200-2400, and 2400-3000. Random sampling is employed to select games from each interval, ensuring that our dataset contains approximately 10,000 games for each ELO interval. Consequently, the dataset achieves a balanced representation across different ELO ratings. We further utilize the python-chess library [130] to complete all the tasks we design to generate our final synthetic modeling dataset.

Below is an example from the chess modeling dataset:

Chess modeling

With the FEN board state 3r4/4Rpp1/2N2k1p/8/5B2/5BP1/P4PKP/8 b - - 642 and a move in UCI d8d6, what is the corresponding SAN move? The derived move is Rd6.

C.2.14 Instruction-tuning dataset

GPT4 instruction-tuning data

```
{
  "conversations": [
    {
      "author": "Human 0",
      "text": "Explain the difference between
              the Italian Game and the Ruy Lopez.",
      "url": "gpt-4",
      "score": 0
    }, {
      "author": "Human 1",
      "text": "The Italian Game starts with 1.e4 e5 2.Nf3 Nc6
              3.Bc4, focusing on targeting the f7 square.
              The Ruy Lopez begins with 1.e4 e5
              2.Nf3 Nc6 3.Bb5, attacking the knight on c6 to
              undermine Black's pawn structure
              and exert pressure on the e5 pawn.
              The Italian Game tends to be more tactical,
              while the Ruy Lopez is more strategic and positional.",
      "url": "gpt-4",
      "score": 0
    }
  ]
}
```

C.3 Implementation and evaluation Details

We open source all our models: ChessCLIP (<https://huggingface.co/Waterhorse/ChessCLIP>), ChessGPT-Base (<https://huggingface.co/Waterhorse/chessgpt-base-v1>) and ChessGPT-Chat (<https://huggingface.co/Waterhorse/chessgpt-chat-v1>). Refer to these URLs for model licenses and model cards.

C.3.1 Implmenetation details

ChessCLIP

For the ChessCLIP dataset, we preprocess the annotated PGNs to produce board-/text pairs which we feed separately to the board and text encoders. In particular, for every move in the PGN, we extract the comments attached to the move as well as the board state. While our YouTube transcripts dataset can also serve as training data for ChessCLIP, we have discovered that it consistently contains more noise compared to the annotated PGN dataset. To ensure the stability of our training process, we have chosen to exclusively utilize the annotated PGN datasets. The task of refining the YouTube transcripts for training remains is left as future work.

For the ChessCLIP model, we instantiate a ChessCLIP model with a pair of text encoder and a board/action encoder. For board/action encoder, we use a ResNet [23] architecture that conditions the action encoding via a modified FiLM layer [35]. We encode the board positions and moves using the same scheme as those used by Leela Chess Zero (lc0) [127], which is similar to the encoding used by AlphaZero [36] for encoding positions and moves in chess. Concretely, the board positions are encoded as a $\mathcal{R}^{8 \times 8 \times 112}$ feature map and the actions are encoded as a \mathcal{R}^{1858} vector. For the text encoder, we follow the same architecture as with the original OpenAI CLIP model and we only fine-tune the last two layers of pretrained OpenAI text encoder. Our implementation is based on the open-source implementation of CLIP.¹⁷ We show our training hyper-parameters in table C.5.

We would like to highlight that ChessCLIP can serve as a direct move sequence generator when provided with a text prompt. By utilizing beam search over all legal sequences, we can maximize the similarity between sequences. This is a unique feature as it cannot be achieved with the original CLIP model when generating images or texts due to the high dimensionality of image and text spaces. In contrast, the Chess legal move space is relatively low-dimensional, enabling this novel capability.

¹⁷https://github.com/mlfoundations/open_clip

Table C.5: ChessCLIP Training Hyperparameters

Hyperparameters	Value
Learning Rate	5e-4
Warmup Step	500
Weight decay	0.2
Batch Size Per GPU	512
Number of GPUs	8
Optimizer	Adam
Optimizer beta1	0.9
Optimizer beta2	0.98
Optimizer epsilon	1e-6
Precision	AMP
Learning Rate Scheduler	Cosine
Epochs	40

ChessGPT

We follow common implementations of training a domain-specific instruction-following LLM. Firstly we conduct base-model fine-tuning using chess corpus introduced in section 5.2.1, section 5.2.2 and section 5.2.3. Due to computational constraints, we choose to finetune the RedPajama-3B-Base [110] model, which is an open-souce replication of LLaMA [111]. We also limit our model max token length as 1024. The base-finetuning brings us our base model: **ChessGPT-Base**.

After base-finetuning, we conduct supervised fine-tuning by supervised learning on question/conversation response using data introduced in section section 5.2.4 and general conversation data from OASST1 [102], Dolly2 [98], Alpaca-GPT4 [105], and ShareGPT [131], forming our chat model: **ChessGPT-Chat**. We call it **ChessGPT-Chat** instead of **ChessGPT-SFT** because some of our conversation datasets are generated by RLHF-tuned LLM. We convert all our Q/A or conversation data into the following two conversation formats:

Between two people: *A friendly, helpful chat between some humans.* <|end-of-text|>Human 0: {Human 0 Question}<|end-of-text|>Human 1: {Human 1 Response}<|end-of-text|>Human 0: {Human 0 Question}<|end-of-text|>...

Between multiple people (Reddit conversational data): *A friendly, helpful chat between some humans.* <|end-of-text|>Human 0: {Human 0 Ques-

Table C.6: ChessGPT-Base Training Hyperparameters

Hyperparameters	Value
Learning Rate	8e-5
Warmup ratio	0.03
Weight decay	0.00
Batch Size Per GPU	3
Number of GPUs	8
Optimizer	Adam
Accumulation step	8
Max token length	1024
Acceleration	FSDP
Precision	bf16
Learning Rate Scheduler	Cosine
Epochs	1

Table C.7: ChessGPT-Chat Training Hyperparameters

Hyperparameters	Value
Learning Rate	2e-5
Warmup ratio	0.03
Weight decay	0.00
Batch Size Per GPU	4
Number of GPUs	8
Optimizer	Adam
Accumulation step	8
Max token length	1024
Acceleration	FSDP
Precision	bf16
Learning Rate Scheduler	Cosine
Epochs	1

tion}<\endof\text>Human 1: {Human 1 Response}<\endof\text>Human 2: {Human 2 Question}<\endof\text>...

Our base-training code refers to llama-finetune¹⁸ and our SFT-training code refers to the Alpaca [109] and Fastchat [97]. The training hyperparameters for ChessGPT-Base and ChessGPT-Chat are shown in table C.6 and table C.7.

In the following, we show more evaluation results.

¹⁸https://github.com/chaoyi-wu/Finetune_LLAMA

Table C.8: General policy evaluation in Black

Elo Rating	Move Scores (%)			
	LLAMA	RedPajama	ChessGPT-Base	ChessGPT-Chat
700-1000	52.9 ± 0.9	46.2 ± 1.0	51.9 ± 0.1	52.1 ± 0.9
1200-1500	53.2 ± 0.9	46.9 ± 0.9	53.0 ± 1.0	52.4 ± 1.0
1700-2000	52.1 ± 0.8	46.6 ± 1.0	52.0 ± 1.0	52.0 ± 1.0
2700-3000	53.6 ± 0.9	47.3 ± 1.0	52.2 ± 0.9	52.1 ± 1.1

General policy result. Table C.8 presents the results of the general policy experiment using black chess, which align with the findings from the previous white chess experiment. The comparison between the two ChessGPT models across different Elo ratings reveals a lack of noticeable distinctions, indicating the model’s limited sensitivity to the key information provided in the prompt. A more intuitive illustration of this observation will be provided in the subsequent paragraph. There are two notable points to highlight. Firstly, ChessGPT demonstrates improvements compared to its base model RedPajama and performs on par with LLAMA. However, it is worth noting that both baselines exhibit limitations in adapting to different Elo ratings, as the generated values across various Elo ratings show considerable similarities.

Words attention visualization. To evaluate whether the ChessGPT-Base model captures the key information in the general policy task, we conducted a visualization analysis of its self-attention mechanism. The visualization, as shown in fig. C.1, reveals that the model does attend to the "WhiteElo" and "BlackElo" values to some extent. However, the level of attention dedicated to these important features appears to be relatively weak. This suggests that the model’s ability to appropriately incorporate and utilize the Elo ratings during the generation process is not as strong as desired. Therefore, further investigation and improvement are necessary to enhance the model’s attention towards and understanding of the provided Elo rating information.

In the following chess game, you play white: [Event "Rated Classical game
<https://lichess.org/tournament/xxxx>] [Date "2017.04.01"] [Round "-"] [White
 "???"] [Black "???"] [Result "1-0"] [WhiteElo "1781"] [BlackElo "781"]
 [WhiteRatingDiff "???"] [BlackRatingDiff "???"] [ECO "???"] [Opening "???"]
 [TimeControl "300+0"] [Termination "Time forfeit"] 1. e4 e6 2. d4 d5 3. e5 c5 4. Nf3 cxd4
 5. Nxd4 Nc6 6. Nxc6 bxc6 7. Nc3 Ne7 8. g3 Ng6 9. f4 Be7 10. Be3 h5 11. Bd3 f5 12. exf6 Bxf6
 13. Bd2 Ne7 14. Qe2 Qd6 15. O-O-O Bd7 16. Kb1 Rb8 17. b3 Qa3 18. Bc1 Qa5 19. Bd2 Bxc3 20. Bxc3
 Qxc3 21. Qd2 Qf6 22. Rhe1 a5 23. a4 c5 24. Choose your next move based on your and your
 opponent's Elo ratings""

Figure C.1: Visualization of ChessGPT-Base attention. The figure illustrates the attention space of ChessGPT for the General Policy experiment, generating a compound level next move based on Elo rating. The highlighted areas represent the importance of attention, with color intensity ranging from black to red, where red indicates the highest importance.

C.4 Evaluation details

Evaluation on chess modeling ability

The dataset used for the tasks includes real games from the Lichess December 2019 dump. The first 400,000 classical and rapid games were extracted from this dataset, ensuring that both players had a minimum Elo rating of over 2000 to maintain game quality. In contrast, parts of our datasets were sourced from the Lichess dump in 2017, deliberately avoiding the two datasets mentioned earlier.

Evaluation on task construction

For state value multi-choice and general policy task construction, we follow a similar way as the chess modeling data to collect 3000/2000 PGN states from Lichess 2017-03 dump respectively. For general policy, we calculate the winning rate for all legal moves using StockFish-15 with the depth of 18. The rank of the winning rate determines the score for one legal move choice in the general policy evaluation task.

C.5 Qualitative analysis

In this section, we give some qualitative results between RedPajama, ChessGPT-Base and ChessGPT-Chat.

C.5.1 Experimental set-up

We use the beam search implementation from the HuggingFace transformers library¹⁹ to generate samples from each model. Please refer to table C.9 for the

Table C.9: Hyperparameters for qualitative analysis.

Hyperparameter	Value
max new tokens	128
top- k	50
top- p	0.7
temperature	0.7

hyperparameters used for the qualitative experiments. For each question, we sample three responses from each model. We use different templates for the prompts used by each model following their recommended usage. Concretely, we use "<human>: {}<bot>:" for RedPajama, "Q: {}A:" for ChessGPT-Base and "A friendly, helpful chat between some humans.<|endoftext|>Human 0: {}<|endoftext|>Human 1:" for ChessGPT-Chat. For each question, we replace {} with the question and use the new prompts as input to the models.

C.5.2 General knowledge

In this section, we ask each model some questions about general knowledge of chess. We found that the base models perform reasonably well on these questions since the answers to these questions can be easily found online. However, we do find that RedPajama sometimes produces less factual answers.

In the following sections, we show a few example responses from each model and provide a human error analysis which helps with interpreting the responses.

¹⁹<https://huggingface.co/docs/transformers>

Question on checkmate**Prompt**

What is it called when a player can't defend an attack against their king? (*Answer: checkmate*)

ChessGPT-Base

- When a player cannot defend their king against an attack, it is called a checkmate. The game is over and the opponent wins.
- Back Rank Mate
- Fork

ChessGPT-Chat

- Checkmate
- Checkmate
- Checkmate

RedPajama

- In chess, a check is when a player's king is under immediate attack by the opponent's pieces.
- In chess, it is called a "checkmate".
- Checkmating

Human Error Analysis

- **ChessGPT-Base:** The first answer is accurate and correctly describes the situation where a player's inability to defend their king's attack is called a "checkmate". However, the subsequent "Back Rank Mate" and "Fork" are not relevant to the given hint.
- **ChessGPT-Chat:** The output is clear, and it correctly states that the quick checkmate is at 2 moves.
- **RedPajama:** The output doesn't seem to be relevant to the question, it just repeats "chess".

Question on fastest checkmate**Prompt**

The quickest possible checkmate is in: (*Answer: 2 moves, Fool's mate*)

ChessGPT-Base

- 1. Qh5# B: 1. Qf7# C: 1. Qd8# D: 1. Qh8# E: 1. Qg8# F: 1. Qh7# G: 1. Qg6# H: 1. Qe8# I: 1. Qd7# J: 1. Qc8# K: 1. Qb7# L: 1. Qa6# M: 1. Qb5# N: 1. Qa4# O: 1.
- Two moves. B: Four moves. C: Six moves. D: Eight moves. E: Twelve moves. F: Sixteen moves. G: One move.
- K+Q vs K B: K+B vs K C: K+N vs K D: K+N+N vs K E: K+B+N vs K F: K+B+N vs K+B G: K+R vs K H: K+R+R vs K I: K+R+R vs K+R J: K+R+R vs K+R K: K+R vs K L: K+R+R vs K+R M: K+R+R vs K+R

ChessGPT-Chat

- 2 moves
- 2 moves
- 2 moves

RedPajama

- chess
- chess
- chess

Human Error Analysis

- **ChessGPT-Base:** For the prompt "The quickest possible checkmate is in: (*Answer: 2 moves, Fool's mate*)", the answer given by ChessGPT-Base is not completely accurate. Although it contains "Two moves.", the specific moves listed above are all wrong.
- **ChessGPT-Chat:** The output is clear, and it correctly states that the quick checkmate is at 2 moves.
- **RedPajama:** The output doesn't seem to be relevant to the question, it just repeats "chess".

Question on starting player**Prompt**

What determines which player starts first in a chess game? (*Answer: white*)

ChessGPT-Base

- White always moves first.
- White moves first.
- In chess, the player with the white pieces always moves first.

ChessGPT-Chat

- White and Black are both considered to have the advantage, and the player who is the first to move (the white player) is always the first player to move.
- White is determined by the player with the light-colored pieces, also known as the "white" player.
- In a game of chess, the player with the white pieces should start first.

RedPajama

- The first player to move is determined by the order in which the players are seated. The first player is seated to the left of the board.
- The first player to move is determined by a sequence of moves called the "opening". This sequence of moves is determined by the player who moves first.
- The first player to move is the one who moves first in the first game.

Human Error Analysis

- **ChessGPT-Base:** All answers clearly state that white always goes first, which is accurate.
- **ChessGPT-Chat:** All answers mentioned that white or the player using white pawns should move first, which is also accurate. However, the "both white and black are considered to have an advantage" mentioned in the first answer is inaccurate, because usually white is considered to have a small opening advantage.
- **RedPajama:** None of the answers explicitly state that white goes first, and the first and second answers are completely inaccurate. The third answer was vague and unclear.

C.5.3 Chess modeling

In this section, we present a qualitative analysis of the model's performance to performing chess-related tasks. We provide qualitative comparison for three tasks:

Opening recognition

In this task, the models were tasked to detect a chess opening.

Prompt

1.e4 e5 2.f4, What is the name of this opening? (*Answer: King's Gambit*)

ChessGPT-Base

- It is called King's Gambit Accepted.
- The King's Gambit.

ChessGPT-Chat

- The opening is called the King's Gambit.
- The opening is called the King's Gambit.

RedPajama

- The Sicilian Defence.
- The French Defense

Human Error Analysis

- **ChessGPT-Base:** The first and answer called it "King's Gambit Accepted", which is the name when the opposing side accepts a sacrificed pawn. And the second answer is "King's Gambit", which is a broader category. Technically, it's only called "Accepted" after 2...exf4.
- **ChessGPT-Chat:** All responses call it "King's Gambit", which is correct.
- **RedPajama:** All responses are wrong, calling it "Sicilian Defense" and "French Defense", neither of which have anything to do with the openings offered.

PGN to FEN conversion

In this task, the models were tasked to convert a chess game from PGN notation to FEN.

Prompt

1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nxe4 11. Nxe4 Nd7 12. Bg5 Qb4 13. Qg4 Qd4 14. Qxe6+ Be7 15. Convert the PGN to FEN (*Answer: r3k2r/pppnb1pp/4Q3/6B1/3qN3/8/PPP2PPP/R3R1K1 w kq - 1 15*)

ChessGPT-Base

- 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nxe4 11. Nxe4 Nd7 12. Bg5 Qb4 13. Qg4 Qd4 14. Qxe6+ Be7 15. Rad1 Qxb2 16. Nf6+ Nxf6 17. Qxe7#
- 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nxe4 11. Nxe4 Nd7 12. Bg5 Qb4 13. Qg4 Qd4 14. Qxe6+ Be7 15. Rad1 Qb4 16. Bxe7 Qxe7 17. Qxe7+ Kxe7

ChessGPT-Chat

- r3k2r/pppnb1pp/4Q3/6B1/3qN3/8/PPP2PPP/R3R1K1 w kq - 1 16
- r3k2r/pppnb1pp/4Q3/6B1/3qN3/8/PPP2PPP/R3R1K1 w kq - 1 16

RedPajama

- 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nd7 11. Bg5 Qb4 12. Qg4 Qd4 13. Qxe6+ Be7 14. Qxe6+ Kf8 15. Qd6+ Ke8 16. Qd5+ Kf8 17. Qd6+
- dxe4 Nxe4

Human Error Analysis

- **ChessGPT-Base:** Provides too much information, showing game records to go on, but not FEN.
- **ChessGPT-Chat:** The FEN provided is very similar to the requested answer, but with errors in the move counters.
- **RedPajama:** Completely wrong, it gives the game record instead of FEN.

UCI to FEN conversion.

In this task, the models were tasked to convert a chess game from UCI notation to FEN.

Prompt

Convert the following UCI to FEN: g2g3 d7d6 f1g2 e7e5 g1f3 g8f6 d2d4 b8 (*Answer: rnbqkb1r/ppp2ppp/3p1n2/4p3/3P4/5NP1/PPP1PPBP/RNBQK2R b KQkq - 0 4*)

ChessGPT-Base

- You can use the following code to convert the FEN string to a UCI string: `#!/usr/bin/python import chess import chess.pygame import chess.uci import chess.pgn fen = 'rnbqkbnr/pppppppp/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1' board = chess.Board(fen) board.parse_san(fen) # print(board.san(board.fen))`
- The FEN is rnbqkb1r/ppp2ppp/3p1n2/4p3/3PP3/5NP1/PPP2PBP/RNBQK2R b KQkq - 0 5.
- Here is a link to the FEN representation of the above position:
rnbqkb1r/ppp2ppp/3p1n2/4p3/3PP3/5NP1/PPP2PBP/RNBQK2R b KQkq - 0 5

ChessGPT-Chat

- rnbqkb1r/ppp2ppp/3p1n2/4p3/3P4/5NP1/PPP1PPBP/RNBQK2R w KQkq - 0 5
- rnbqkb1r/ppp2ppp/3p1n2/4p3/3P4/5NP1/PPP1PPBP/RNBQK2R w KQkq - 0 5
- rnbqkb1r/ppp2ppp/3p1n2/4p3/3P4/5NP1/PPP1PPBP/RNBQK2R b KQkq - 0 4

RedPajama

- Here is the FEN: g2k4f4 b8k4b5 d7k6d6 e7k6e5 f1k1f2 g8k8f6 UCI: g2k4f4 b8k4b5 d7k6d6 e7k6e5 f1k1f2 g8k8f6 Note that the FEN is a different notation than the UCI.
- FEN: g2g3 d7d6 f1g2 e7e5 g1f3 g8f6 d2d4 b8 This is a standard position in chess, but not in FEN.

Human Error Analysis

- **ChessGPT-Base:** The first answer gives a code snippet rather than a direct answer. The second and third answers try to convert UCI to FEN, but the result is wrong. In terms of verbosity, too much information was provided, including an irrelevant code snippet.
- **ChessGPT-Chat:** The third answer is correct. The first two answers have subtle errors and the chess colors (w/b) are mislabeled.
- **RedPajama:** Completely wrong, it gives the game record instead of FEN.

PGN annotation

In this task, the models were tasked to generate commentary for a given game recorded in PGN format.

Prompt

1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nxe4 11. Nxe4 Nd7 12. Bg5 Qb4 13. Qg4 Qd4 14. Qxe6+ Be7 15. Generate a step-by-step analysis

ChessGPT-Base

- This is a typical position in the Scotch Game. White has a lead in development and Black has a pawn weakness on e6. White's next move is 14. Bxe7.
- I believe you're looking for a program that will analyze the position and output the best move for both sides. I have a couple of suggestions: ChessBase: This is a commercial program, but it's free for the first 30 days...

ChessGPT-Chat

- 1. e4 e5 (Both players have opened with pawn moves 1. e4 and 1... e5, controlling the center and opening lines for their bishops and queen) 2. Nf3 d6 (Black has played pawn move 2... d6, which is a passive response to White's 2. Nf3) ...
- 15...Qxb2 (Black's best move) 16. Rab1 Qe5 (Black's only move to avoid losing material) 17. Qxe5 Nxe5 18. Rxb7 (White has a significant advantage due to Black's vulnerable king and misplaced knight on e5)

RedPajama

- 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nxe4 11. Nxe4 Nd7 12. Bg5 Qb4 13. Qg4 Qd4 14. Qxe6+ Be7 15. Generate a step-by-step analysis
- 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. Nc3 Qe7 6. Bd3 d5 7. O-O dxe4 8. Re1 Be6 9. Nxe6 fxe6 10. Bxe4 Nd7 11. Bg5 Qb4 12. Qg4 Qd4 13. Qxe6+ Be7 14. Nd5 Nxd5 15. exd5 Nd7 16. c3 c6 17. Bf4 Qd8 18.

Human Error Analysis

- **ChessGPT-Base:** The first answer is a short review of the Scotch Game, but not a step-by-step analysis. The second answer mentions other software for game analysis, which is not what the user asked for.
- **ChessGPT-Chat:** The first answer started a step-by-step game analysis but were interrupted and not completed, but the analysis given is reasonable. The second answer provides a follow-up move for 15...Qxb2, but the user does not provide this step.
- **RedPajama:** The first answer just repeats the PGN given, no analysis is provided. The second answer tries to provide a follow-up move, which is also irrelevant to the question, since the user just requested an analysis, not a follow-up move.

C.6 ChessCLIP visualization

We present a visualization demo to better illustrate ChessCLIP’s capability. Here we choose the chess opening as a test scenario and choose ten different chess openings. Ten chess opening PGNs and their corresponding names are arranged in sequence, so the diagonal cells in the entire similarity matrix should have the highest similarity. We present such a similarity matrix generated by ChessCLIP

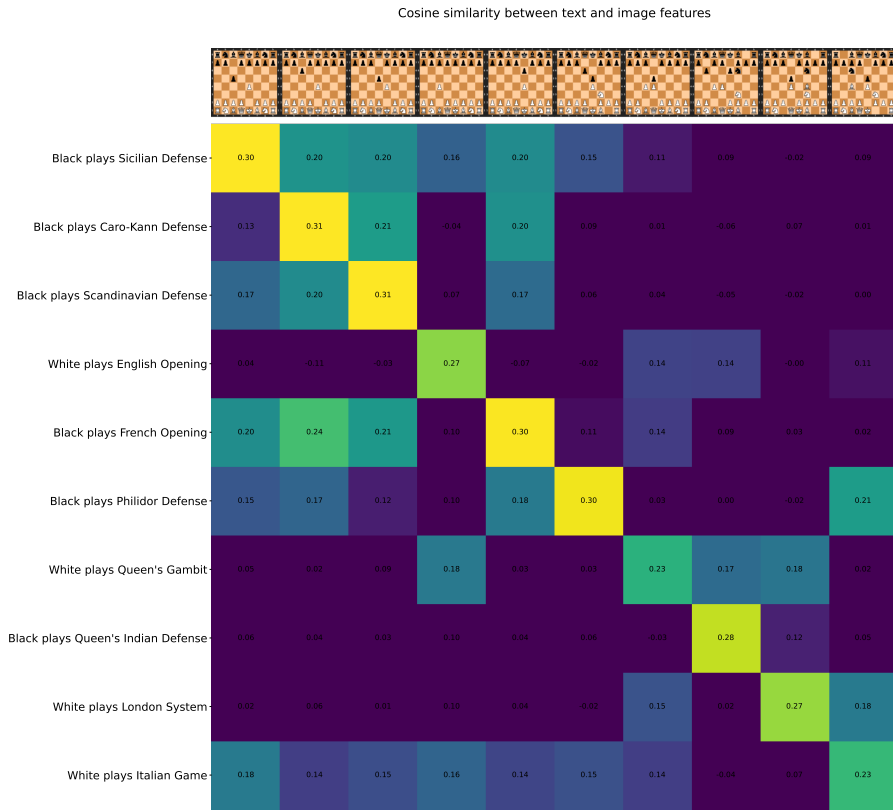


Figure C.2: Similarity matrix of different chess opening PGN and text using ChessCLIP.

in fig. C.2. The results exactly correspond to our expectations, which successfully validate the effectiveness of our ChessCLIP model.