ABSTRACT

KRISHNAN, ABHIJEET. Interpretable Strategy Synthesis for Competitive Games. (Under the direction of Dr. Chris Martens and Dr. Arnav Jhala).

Competitive games admit a wide variety of player strategies and emergent, domain-specific concepts that are not obvious from mere examination of their rules. Independently developing useful strategies requires expertise and time, which not all players have access to. Algorithmically generating strategies that players can understand could help them become more competent, improving their decision-making and leading to better performance. This thesis describes algorithmic approaches to the problem of *interpretable strategy synthesis* – modelling and learning strategies for games that are both effective and understandable to a human player.

The thesis begins by formalizing this problem in the context of reinforcement learning, as that of finding a policy for a game-based environment that achieves high reward while being *interpretable* by a human player. This problem formulation is applied to the game of chess to develop a first-order logic-based policy model inspired by the documented concept of *chess tactics*. Two novel metrics - coverage and divergence, are introduced to evaluate the performance of learned chess strategies. Existing chess tactics are translated into this policy model and evaluated, with the results showing that they are effective at capturing the behavior of beginner players, but not expert ones. I then introduce a learning algorithm based on inductive logic programming to automatically learn strategies for chess in the form of the previously introduced logic-based policy model. The learned strategies are evaluated using the coverage and divergence metrics, and the results show that the learned strategies are better at approximating human beginners than a random baseline. I build on this work by introducing two additional constraints based on precision and recall that measurably improve the performance of the learned tactics. Finally, I introduce another approach to interpretable strategy synthesis that models a strategy as a programmatic policy, and learns it using the decision transformer model. I apply this approach to tasks in the Karel programming environment and show that the learned strategies are competitive with those learned by a state-of-the-art approach, while being significantly more sample efficient.

This thesis contributes algorithmic techniques to model and learn policies for games that are both interpretable, and effective. An evaluation of the strategies learned by these approaches finds that their performance outperforms comparable baselines. I expect these techniques to be useful for esports analytics in a variety of game domains. More generally, I also expect that these techniques will be useful for generating explanations of artificial agent behavior in reinforcement learning tasks, increasing trust and verifiability of these systems.

Interpretable Strategy Synthesis for Competitive Games

by
Abhijeet Krishnan

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2024

APPROVED BY:

Dr. James C. Lester	Dr. Jonathan Stallrich Graduate School Representative
Dr. Chris Martens Co-chair of Advisory Committee	Dr. Arnav Jhala Co-chair of Advisory Committee

CONTENTS

List of Tables . . . . . **v**

List of Figures . . . . . **vi**

Chapter 1 Introduction **1**
1.1 Motivation 1
1.2 Thesis Statement 4
1.3 Thesis Overview 5

Chapter 2 Background and Related Work **8**
2.1 Player Modeling 8
2.2 Reinforcement Learning 9
2.3 Explainable Reinforcement Learning 10
2.4 Interpretable Strategy Synthesis 11

Chapter 3 A Framework for Interpretable Strategy Synthesis **14**
3.1 Problem Definition 15
3.2 Strategy Model 16
3.3 Interpretability Measure 17

Chapter 4 Strategies as First-Order Logic Rules **19**
4.1 Background 19
4.1.1 Inductive Logic Programming 19
4.2 First-Order Logic Rules as Policies 21
4.2.1 Rule-based Policy Model 21
4.2.2 Learning Rule-based Policies 22
4.3 Application to Chess 22
4.3.1 Chess 22
4.3.2 Strategy Model for Chess 23
4.3.3 Metrics to Evaluate Chess Strategy Models 24
4.3.3.1 Coverage 25
4.3.3.2 Divergence 25
4.4 Evaluation of Known Chess Tactics 26
4.4.1 PAL System 26
4.4.1.1 Pattern Formalism 27
4.4.1.2 Pattern Synthesis 27
4.4.2 Methodology 28
4.4.3 Evaluation 29
4.4.4 Results and Analysis 30
4.4.5 Conclusion and Future Work 30
4.5 Learning Chess Tactics using Inductive Logic Programming 31
4.5.1 Popper 32
4.5.2 Methodology 32

4.5.3 Evaluation . . . 33
4.5.3.1 Dataset . . . 34
4.5.3.2 Training . . . 34
4.5.3.3 Performance Metrics . . . 34
4.5.4 Results . . . 35
4.5.4.1 Qualitative Analysis . . . 39
4.5.4.1.1 Low Divergence . . . 39
4.5.4.1.2 Highest Accuracy . . . 39
4.5.4.1.3 Variable Reshuffling . . . 40
4.5.5 Discussion . . . 40
4.5.6 Conclusion . . . 41
4.6 Improving Tactic Learning using Precision and Recall . . . 42
4.6.1 Methodology . . . 42
4.6.1.1 Constraints Based on Precision and Recall . . . 42
4.6.1.2 Predicate Vocabulary . . . 43
4.6.2 Evaluation . . . 44
4.6.2.1 Dataset . . . 45
4.6.2.2 Training . . . 45
4.6.2.3 Performance Metrics . . . 45
4.6.3 Results and Analysis . . . 45
4.6.3.1 Qualitative Analysis . . . 48
4.6.3.1.1 Low Divergence . . . 48
4.6.3.1.2 Highest Accuracy . . . 48
4.6.4 Discussion and Future Work . . . 49
4.6.4.1 Extensibility to Other Domains . . . 49
4.6.5 Conclusion . . . 49
4.7 Limitations and Future Work . . . 50

Chapter 5 Strategies as Programs . . . 51
5.1 Programs as Policies . . . 51
5.1.1 Programmatic Policy Model . . . 52
5.1.2 Learning Programmatic Policies . . . 52
5.1.2.1 Offline Reinforcement Learning . . . 54
5.2 Learning Programmatic Policies using Decision Transformers . . . 55
5.2.1 Background . . . 55
5.2.1.1 Transformers . . . 55
5.2.1.2 Decision Transformers . . . 56
5.2.2 Methodology . . . 56
5.2.3 Evaluation . . . 57
5.2.3.1 Karel domain . . . 57
5.2.3.2 Dataset . . . 58
5.2.3.3 Training . . . 58
5.2.3.4 Results . . . 62
5.2.4 Discussion . . . 62
5.2.4.1 Training Requirements . . . 62

5.2.4.2 Sensitivity to Training Data Quality 63
5.2.4.3 Hardware Requirements 63
5.2.4.4 Novelty of Generated Programs 65
5.2.5 Conclusion 65

Chapter 6 Conclusion 66

Appendices 82
Appendix A Abbreviations 83
Appendix B Symbols 85
Appendix C Background Knowledge Definitions 87

LIST OF TABLES

Table 2.1 List of works in Interpretable Strategy Synthesis (ISS) according to the framework proposed in Chapter 3 13

Table 3.1 List of works in ISS demonstrating their domain-specificity and lack of interpretability evaluation. 18

Table 4.1 Patterns learned by the Patterns and Learning (PAL) system that are used to create tactics 29

Table 4.2 Coverage and Divergence for each tactic 30

Table 4.3 Summary statistics for the tactics learned by each system in the ablation study. 45

Table 5.1 Statistics for the LEAPS program dataset. 58

Table 5.2 Hyperparameters used for training the Decision Transformer for the Karel programmatic policy synthesis task. 60

Table 5.3 Mean return of the best program found and number of unique programs explored by LEAPS and the Decision Transformer model on each Karel task. The standard deviation is reported in parentheses. 62

LIST OF FIGURES

Figure 1.1 A ladder in the board game Go. 3

Figure 2.1 The agent-environment interaction interface in a Markov Decision Process. Reproduced from Figure 3.1 of Sutton and Barto (2018). 9

Figure 4.1 Rule-based Policy Model 21
Figure 4.2 Example of the fork tactic in chess 24
Figure 4.3 Chess Tactic Model 24
Figure 4.4 PAL rule for the `can_check` pattern 27
Figure 4.5 Modified PAL rule for the `pin` pattern to convert it into a tactic 28
Figure 4.6 An example of the limitations of our 1-ply `can_fork` tactic 31
Figure 4.7 An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary 33
Figure 4.8 Divergence histogram for T evaluated using Maia-1600 35
Figure 4.9 Divergence histogram for T evaluated using Stockfish 14 36
Figure 4.10 Coverage histogram for T 37
Figure 4.11 Accuracy histogram for T 38
Figure 4.12 A learned tactic with low divergence from ground as evaluated by Stockfish 14. 39
Figure 4.13 A learned tactic with the highest accuracy. 39
Figure 4.14 A learned tactic with meaningless variable reshuffling. 40
Figure 4.15 An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary. 44
Figure 4.16 Boxplot of tactic divergence (evaluated using Maia-1600) for each system 46
Figure 4.17 Boxplot of tactic divergence (evaluated using Stockfish 14) for each system 47
Figure 4.18 A learned tactic with low divergence from ground as evaluated by Maia-1600. 48
Figure 4.19 A learned tactic with the highest accuracy. 49

Figure 5.1 The domain-specific language (DSL) for constructing programs in the Karel environment. 53
Figure 5.2 A sample Karel world of size 4×6. The blue diamond represents a marker. The agent cannot travel through walls. 57
Figure 5.3 A histogram of the returns of the trajectories in the LEAPS dataset. 59
Figure 5.4 Return of the decision transformer (DT) model on each task during training. 61
Figure 5.5 Return of the DT model when trained on the top-k% of the dataset. . . . 63
Figure 5.6 Box plot of returns for programs sampled from the DT model. 64

CHAPTER

1

INTRODUCTION

1.1 Motivation

Esports (or e-sports, eSports) refers to the ecosystem around organized competition in video games. It has emerged as a rapidly growing industry in recent years, projected to reach a market volume of $12.5B by 2030 (Grand View Research 2022), and has become a mainstream form of entertainment, attracting millions of viewers and players worldwide (Lynch 2017). This has led to the development of a fledgling education ecosystem around it, consisting of community-created guides , professional coaching services (Defer 2023; Sherry 2022; Kharif 2020) and platforms (Inc. 2024), and researchers and analysts who seek to improve the performance of esports athletes, teams and organizations (Reitman et al. 2020).

Esports analytics is a relatively new field that utilizes statistical analysis and data mining techniques to gain insights into the performance of esports players, teams and games. Formally, Schubert et al. (2016) defines it as "the process of using esports related data, primarily behavioural telemetry but also other sources, to find meaningful patterns and trends in said data, and the communication of these patterns using visualization techniques to assist with decision-making processes." There also exist commercial enterprises that provide esports analytics as a service for player improvement (Shadow Esports 2023; Gamers Net, Inc. 2023).

These services tend to cater to the upper echelon of players, typically existing professionals

or those aspiring to it . Such services are either prohibitively expensive, or too sophisticated for use by the vast majority of players looking to improve . While the wealth of community-created learning resources are available to all, they require an investment of time to sift through and understand , and might not meet the needs of an individual player . Coaching platforms democratize access to coaches who can offer personalized advice, but they cannot scale to the vast population of players who seek improvement . This creates a gap in the market for an automated coach that can provide personalized advice while scaling to the needs of many players.

Solving this problem is equivalent to developing an Interactive Tutoring System (ITS) for games. One approach towards building an ITS is to model the student using cognitive theories which can explain their processes of thinking and understanding (Chrysafiadi and Virvou 2013). Researchers in varied fields agree that people create *mental models* as shorthand for understanding the human experience (Boyan and Sherry 2011). Mental models are described as "mechanisms whereby humans are able to generate descriptions of system purpose and form, as explanations of system functioning and observed system states, and as predictions (or expectations) of future system states" (Rouse et al. 1992). In the context of games, players refine their mental models through solitary practice or formal instruction, discussion of strategies among peers, or investigation of underlying game mechanics (Boyan, McGloin, et al. 2018). Evidence suggests that when provided with optimal game strategies, players are able to transfer them to their game play (Paredes-Olay et al. 2002). However, acquiring good strategies involves many hours of deliberate practice, or access to expert knowledge, which may not be easily available (Ericsson et al. 1993). Automatically learning good game-playing strategies that players can use to refine their mental model would help overcome this challenge, and would be a good way to provide personalized advice to players.

Strategies in esports (or in games more generally) are policies that tend to lead to favorable states for players. An example of a strategy in the board game chess is *acquiring center control* – having your own pieces occupy or threaten the center squares where they have maximum mobility and control over the rest of the board (Seirawan 2005a, Chapter 8). Players learn to improve in two ways: 1) by identifying good strategies themselves with sufficient practice; and 2) by learning from the ones shared by more experienced players through an established vocabulary . Strategies have been found, described, and shared by players across a wide variety of games. For example, in the board game Go, the *ladder* (see Figure 1.1) is a basic capturing technique that new players must learn to identify and play out correctly (Cho 1997). In basketball, the *Triangle offense* (Jackson and Winter 2009) is a well-known team strategy that was instrumental in leading the Chicago Bulls to victory in several NBA championships (Rose 2012). In fighting games, to *frame trap* your opponent is a basic strategy that involves setting up an

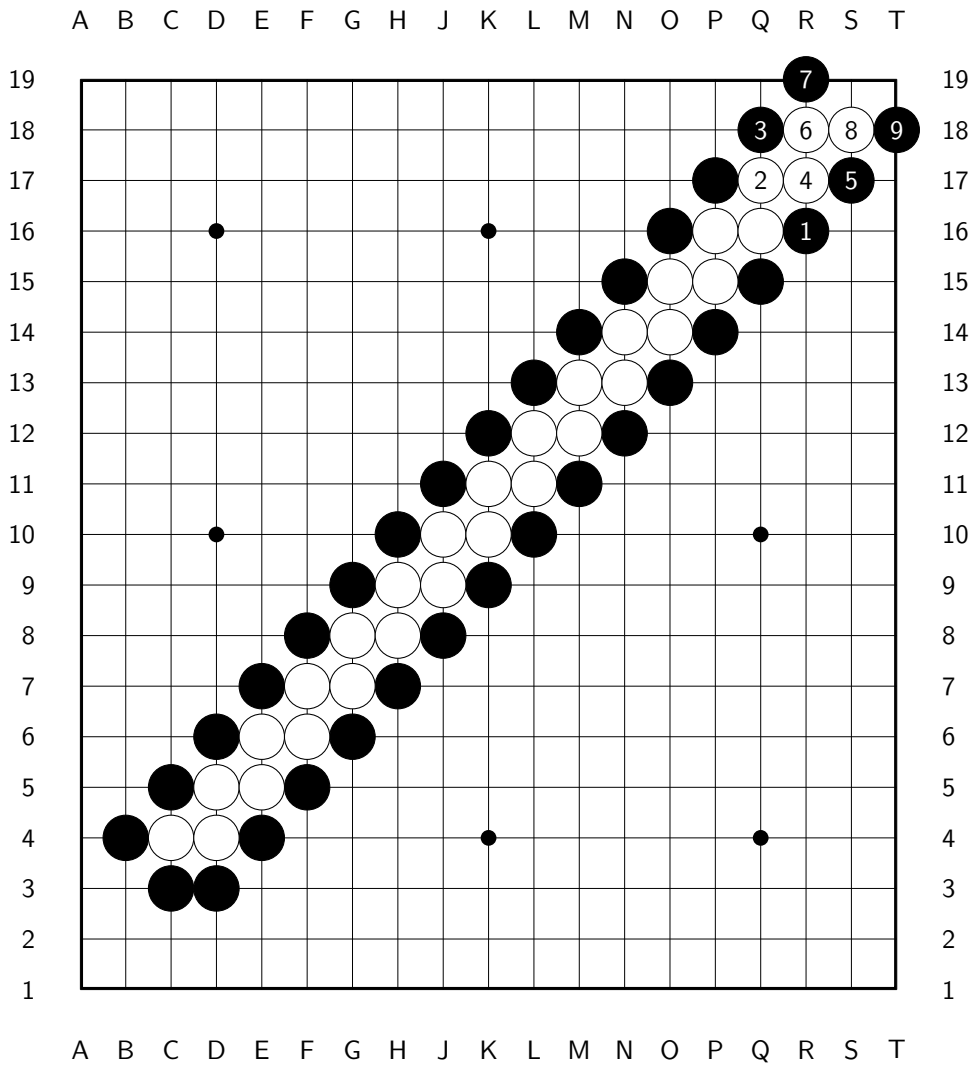

Figure 1.1: A ladder in the board game Go.

advantageous situation with which to attack your opponent, thus ensuring your attack will land without a possibility of counterattack (Silman 2021). Finding useful strategies for playing a game – and an expressive vocabulary to describe them – can help players get better at a game.

A shared vocabulary, or *jargon*, helps players with communicating and explaining strategies that they develop. An example of jargon from fighting games is *okizeme*, which describes the situation where the opponent is knocked down, and the player gets to attack them as they stand back up (Infil 2021). This concept is useful when describing a character's offensive strategy, and can be used to explain why a player chose a particular move in a given situation. An example of jargon from chess is the idea of *tempo*, describing the turns gained or lost when a player is able to make moves that forces their opponent to respond in way that deviates from their original plan (Hooper et al. 1996). Developing good ways to explain strategies can be useful in other domains where a complex system, of which games are an example, is being analyzed. The fields of Explainable Artificial Intelligence (XAI) and Explainable Reinforcement Learning (XRL) have risen in importance as AI-based systems are increasingly used to make decisions in sensitive areas of daily life, such as healthcare and criminal justice (Speith 2022). Providing explanations for their decisions can help increase trust and safety in these systems by identifying when their decisions are reasonable, and when they are not (Narayanan et al. 2018). Generating explainable strategies for games could serve as a testbed for developing explanations for such systems in other domains.

1.2 Thesis Statement

This thesis posits that the problem of ISS i.e., of automatically discovering effective and communicable models of game-playing policies, or *strategies*, would be valuable to solve for the field of esports analytics, and have implications for the fields of XAI and XRL. Strategies can be computationally modeled for various games while being interpretable to human players, and learning algorithms can be used to automatically synthesize them.

The work proposed in this dissertation will benefit the esports analytics industry by providing an additional, powerful tool to gain insight into player strategies. It will benefit XRL research by contributing an additional technique to explain the behavior of reinforcement learning (RL) agents. As such, my overall thesis statement is —

Player growth and understanding in many competitive games is achieved through the communication of concise, interpretable strategies that convey information used to aid decision-making in said games. Learning high-quality strategies that are both interpretable and, once understood, allow a player to make better decisions in-game, would be valuable to players looking to improve. A computational model of a game strategy, along with a learning method,

could meet those goals and impact the fields of competitive esports and Explainable Artificial Intelligence.

1.3 Thesis Overview

This section describes the organization of the rest of the thesis, and the research questions they tackle.

Chapter 3: A Framework for Interpretable Strategy Synthesis

In Chapter 3, I discuss how prior works have attempted to approach the problem of ISS implicitly for specific games. Subsequent works have explicitly mentioned the attempt to approach the problem of ISS, but have not situated it in the context of a formal problem definition, and thus lack meaningful comparison to existing work or the ability to be reused across games. The issue of interpretability has also not been adequately addressed. Providing a unified framework for ISS would help researchers compare and reuse their strategy models and learning algorithms across multiple game domains. I formally define the problem of ISS as that of synthesizing a game-playing policy and introduce a framework for evaluating the quality and interpretability of the learned strategies. This framework answers the research question —

> **RQ1** *How do I formally define the problem of ISS?*

Chapter 4: Strategies as First-Order Logic Rules

In Chapter 4, I describe how a strategy could be modeled as a first-order logic (FOL) rule and learned with Inductive Logic Programming (ILP). In Section §4.3, I apply this approach to chess, showing how to model a chess-playing policy using FOL and learning it from gameplay traces. I show that this approach is able to learn strategies that approximate the behavior of a human beginner, and thus answer the following research question —

> **RQ2** *How do I approach the problem of ISS for the game of chess?*

In Sections §4.3.2 and §4.4, I first show how I may computationally model a chess strategy and translate known chess tactics into the model. I show that these tactics produce better moves on average compared to a random player, thus answering the research question —

RQ2a *Could I represent known chess tactics as a learnable, computational model of a chess-playing policy and develop metrics to show that they suggest better moves than a random baseline?*

In Section §4.5, I show how I can use ILP, a symbolic machine learning (ML) method, to serve as a suitable learning method to learn chess strategies from gameplay, thus answering the research question —

RQ2b *Do the chess strategies learned using ILP outperform a random baseline in how closely their divergence scores approximate a beginner player?*

The learning system proposed in Section §4.1.1 does not produce moves comparable to a human beginner, let alone a strong chess-playing agent. In Section §4.6, I show how modifications to the learning algorithm can be made to improve the quality of the strategies it learns, thus answering the research question —

RQ2c *Do the chess strategies learned by an ILP system incorporating the changes of the new predicate vocabulary and precision/recall-based constraints produce moves better than those learned by an ILP system without these modifications?*

Chapter 5: Strategies as Programs

In Chapter 5, I describe how a strategy may be modeled as a program in a domain-specific language (DSL), and learned with techniques from program synthesis. Synthesizing programs which can be used as a policy would enable a general solution to the problem of ISS. In Section §5.2, I apply this approach to learn strategies for tasks in a toy programming environment using a DT model and show that the learned strategies outperform existing state-of-the-art while being more sample efficient, thus answering the research question —

RQ3 *How do I cast the problem of ISS as that of programmatic policy synthesis?*

In Section §5.1, I show how the problem of ISS can be cast as a programmatic policy synthesis problem. While there is extensive prior work in program synthesis and recent work in representing policies as programs for interpretability, there is no formal mapping of the program synthesis problem to that of learning a policy, nor one connecting it to the problem of ISS, and thus answering the research question —

RQ3a *How do I formally map the problem of ISS to that of programmatic policy synthesis?*

In Section §5.2, I show how a DT model can be used to synthesize strategies for a toy programming environment. I compare the performance of the DT model to an existing approach and demonstrate that it synthesizes comparable strategies while requiring less training data, thus answering the research question —

RQ3b *Could a DT model synthesize better strategies for a toy programming environment when compared to existing approaches?*

Chapter 6: Conclusion

Finally, in Chapter 6, I summarize the contributions of this dissertation, limitations of the work, and directions for future research.

CHAPTER

2

BACKGROUND AND RELATED WORK

In this chapter, I situate the problem of ISS in the broader field of RL, and provide background for the formalisms I will adopt throughout the thesis. I discuss the development of the field of XAI as a reaction to the black-box nature of modern deep learning models, and summarize current approaches to the problem. I then detail prior approaches to ISS in the context of XRL, and highlight the need for a common framework to unify the underlying strategy (policy) representations and learning methods used.

2.1 Player Modeling

Learning strategies by observing other players is akin to developing a *model* of those players. A *player model* is a loosely defined term that describes a predictive or descriptive approximation of a human player, usually computational in nature, to meet some desirable goal. Player modeling is a relatively new field, previously studied under the umbrella of Human-Computer Interaction (HCI) in the form of user modeling (Biswas and Springett 2018) and student modeling (Chrysafiadi and Virvou 2013). Recent surveys on player modeling provide useful taxonomies (Smith et al. 2011) to categorize and understand various approaches, an overview of techniques that have been used in the field (Machado et al. 2011) or for a specific genre of game, namely Massively Multiplayer Online Role-Playing Games (MMORPGs) (Har-

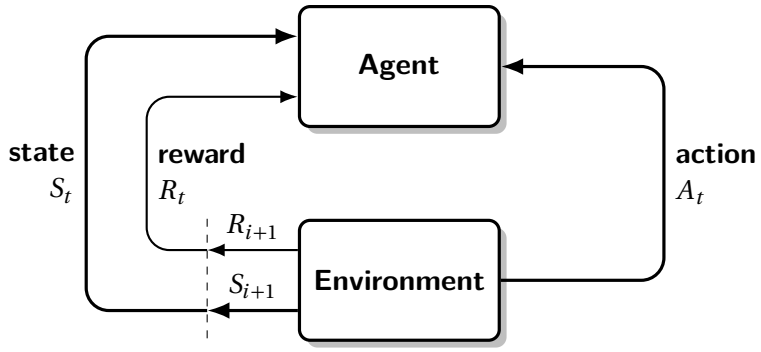

Figure 2.1: The agent-environment interaction interface in a Markov Decision Process. Reproduced from Figure 3.1 of Sutton and Barto (2018).

rison and Roberts 2011), and challenges commonly encountered in the field, including the reliance on knowledge engineering (Hooshyar et al. 2018).

Though most approaches to player modeling involve domain-specific models that involve expert knowledge of the game, there have been many attempts to build domain-agnostic player models. Snodgrass et al. (2019) describe a general player model using the *PEAS framework*, which presents a theoretical framework for providing game recommendations, but does not account for in-game player behavior. Nogueira et al. (2014) use physiological readings of players to model their emotional response to game events. While input-agnostic, this technique relies on physiological data from sensors, which is difficult to acquire. Deep neural networks (DNNs) have been used to simulate the actions of human players in interactive fiction games (Wang et al. 2018) and MMORPGs (Pfau et al. 2018) and to generate new levels in a platformer game based on learned player preferences (Summerville et al. 2016). These techniques rely only on easily obtainable input like gameplay logs or video recordings, and do not use any knowledge engineering to train the model.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a machine learning paradigm where an artificial agent learns to interact with an environment in order to maximize a numerical reward signal. It "is simultaneously a problem, a class of algorithms, and field that studies this problem and its solution methods" (Sutton and Barto 2018). The problem of RL is formally defined as a Markov Decision Process (MDP), where an *agent* interacts with an *environment* by taking *actions* in environmental *states* to receive *rewards*. The agent's goal is to learn a *policy* that maps states to actions in order to maximize the expected sum of rewards over time.

Formally, a MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where is the set of states, is the set of ac-

tions, is the transition function $\mathcal{P}(s' \mid s, a)$, is the reward function $\mathcal{R}(s, a, s')$, and is the discount factor. The agent interacts with the environment by taking actions $a_t \in \mathcal{A}$ in states $s_t \in \mathcal{S}$ at time t, and receives a reward $r_t \in \mathbb{R}$ and transitions to a new state s_{t+1} according to the dynamics of the environment, thus giving rise to a *trajectory* of the form $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$. The agent's goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected sum of rewards over time, or the *return* , defined as $G_t \doteq \sum_{k=0}^{\infty} \gamma^k r_{t+k}$.

This simple formulation has given rise to a rich set of algorithms. The *dynamic programming* class of methods attempt to approximate the optimal value function $V_*(s)$ or policy $\pi_*(s)$ by iteratively updating estimates to them using the Bellman optimality equation (Bellman 1958), while requiring full knowledge of the MDP dynamics. The *Monte Carlo* class of methods estimate the value function or policy by sampling trajectories from the environment, and are model-free. The *Temporal-Difference (TD)* class of methods combine the best of both worlds by bootstrapping from estimates of the value function, and are also model-free.

Recent developments in neural networks and deep learning have led to the development of deep reinforcement learning (DRL) methods, which use DNNs to approximate the value function or policy. These methods have led to game-playing agents capable of competing with and even outperforming the best human experts at various games like chess (Silver, Hubert, et al. 2018), Go (Silver, Huang, et al. 2016), Shogi (Li, Koyamada, et al. 2020), Mahjong (Silver, Hubert, et al. 2018), *StarCraft II* (Vinyals et al. 2019) and *Dota 2* (OpenAI et al. 2019). These agents do not simply take advantage of faster reaction and calculation abilities, but are actually employing new, better *strategies* that lead to more victories. Borrowing from Jeanette Wing's definition of computational thinking (Wing 2008), these agents appear to have better *abstractions* than human experts for the games theyre trained to play.

2.3 Explainable Reinforcement Learning

Humans usually come up with strategies by observing what works in their own play, or studying the gameplay of other experts . Since human players can be treated as a game-playing policy, the problem of ISS is equivalent to that of *explaining* the policy of human players, or more generally, any game-playing agent.

The DRL methods described in Section §2.2 have also shown promise in real-world applications like autonomous driving (Kiran et al. 2021), robotics (Newbury et al. 2023), finance (Mosavi et al. 2020) smart grids (Li, Yu, et al. 2023), and healthcare (Zhou et al. 2021). Despite their application to such high-stakes domains, the neural network (NN)-based policies produced by DRL methods are not interpretable to humans and therefore are difficult to debug when these challenges arise .

Attempts to make RL agent policies amenable to human interpretation have been pursued in the field of Explainable Reinforcement Learning (XRL). Puiutta and Veith (2020) provide a survey of recent XRL methods. Among the interpretability techniques being investigated is that of training an inherently interpretable *surrogate model* which matches the performance of the original agent. Options for this surrogate model that have been explored include decision trees (Bastani et al. 2018; Coppens et al. 2019; Sieusahai and Guzdial 2021) and programmatic policies (Verma, Murali, et al. 2018; Trivedi et al. 2021).

A common theme in early XRL (and the more general field of XAI) work is the imprecise definition of "interpretability" and a lack of metrics to quantify it. For example, Verma, Murali, et al. (2018) evaluate the performance of their methods empirically but their appeal to their programmatic policy's interpretability rests on its representation as readable rules. This theme of measuring performance, but not interpretability, is repeated in other XAI works as well (Coppens et al. 2019; Liu et al. 2019; Shu et al. 2017; Hayes and Shah 2017).

This lack of precision in defining and measuring interpretability led to the work by Doshi-Velez and Kim (2017), which provides a definition of interpretability grounded in psychology, and a taxonomy for evaluating it. Subsequent work has attempted to quantify the interpretability of their proposed XAI techniques (Madumal et al. 2020; Slack et al. 2019; Waa et al. 2018). Other works have expanded on Doshi-Velez and Kim (2017) to explore the notion of interpretability itself (Kliegr et al. 2021; Lage et al. 2019).

2.4 Interpretable Strategy Synthesis

The problem of *strategy synthesis* can be framed in RL terms as that of learning a game-playing policy. However, most existing, state-of-the-art policies are much too complex for any human to memorize and simulate. For example, the default NN model [1] provided with the chess-playing agent *Stockfish 15.1* (The Stockfish team 2022) consists of over 100,000 trainable parameters, which is impossible for a human chess player to keep reliably in memory. A non-rigorous attempt at doing so with a much simpler model was also not successful (Deutsch 2017). The goal of ISS is to learn game-playing policies that can be *interpreted* by human players. This could involve being able to simulate the model in order to play the game, or being able to read the learned policy. This goal overlaps with that of XRL, where ISS can be viewed as a subfield of XRL which attempts to learn explainable policies specifically for game-based environments, with the intended audience of the explanations being human players. The overarching goal is for the synthesized strategies to provide some value to players in terms of their in-game performance.

[1] https://tests.stockfishchess.org/api/nn/nn-52471d67216a.nnue

The earliest works in ISS primarily learned rule-based strategies. The rules are either selected from an authored rulebase, as in Spronck et al. (2004) and Canaan et al. (2018), or derived from a representation like a context-free grammar (CFG), as in Mariño, Moraes, et al. (2021) and Freitas et al. (2018). Rule-based strategies have historically been investigated based on a qualitative appeal to their interpretability. However, none of these works attempt to empirically quantify the interpretability of their synthesized strategies.

Given the prevalence of rule-based strategy representations in previous ISS work, most learning methods for these strategies involve searching over the space of rules to find a combination of them that optimize some performance criteria, usually win rate or score. These methods typically consist of derivative-free optimization techniques (Rios and Sahinidis 2013) like genetic algorithms (GAs), simulated annealing (SA) and local search.

The domains that have been tackled in prior ISS work have either been games with small action spaces, such as Blackjack (Mesentier Silva et al. 2016), Nonograms (Butler et al. 2017) and Hanabi (Canaan et al. 2018), or simplified versions of more complex games such as MicroRTS (Mariño, Moraes, et al. 2021; Medeiros et al. 2022; Mariño and Toledo 2022). Spronck et al. (2004) tackles strategy synthesis for the battle system alone in the commercial single-player game *Neverwinter Nights* (Bioware and Obsidian Entertainment 2002).

Evaluation of the synthesized strategies is primarily done by comparing their performance against hand-authored strategies. None of these works attempt to formally define what an "interpretable" strategy is, create metrics to measure the interpretability of a strategy, or attempt to evaluate the interpretability of a strategy with a user study. There are only appeals made to the interpretability of synthesized strategies due to the human-readability of the rule-based strategy models. Butler et al. (2017) compares synthesized strategies to known, existing strategies and measures how many of the known strategies were replicated by the synthesized set. Mariño, Moraes, et al. (2021) and Mariño and Toledo (2022) provide a qualitative analysis of learned strategies.

Paper	**Domain**	**Model**	**Algorithm**	$\mathcal{R}$	$\mathcal{I}$
Spronck et al. (2004)	Neverwinter Nights	ad-hoc rules	ad-hoc	win rate	–
Mesentier Silva et al. (2016)	Blackjack	if-else rules	search	score	–
Butler et al. (2017)	Nonograms	if-then rules	SMT solver	coverage	–
Canaan et al. (2018)	Hanabi	ad-hoc rules	GA	score	–
Freitas et al. (2018)	Mario AI Framework	DSL script	GA	level score	–
Mariño, Moraes, et al. (2021)	MicroRTS	DSL script	search	win rate	–
Krishnan and Martens (2022a)	Chess	FOL rules	ILP	coverage, divergence	–
Mariño and Toledo (2022)	MicroRTS	DSL script	GA	win rate	–
Medeiros et al. (2022)	MicroRTS, Can't Stop	DSL script	SA, UCT	win rate	–

Table 2.1: List of works in ISS and their classification according to framework proposed in Chapter 3.

CHAPTER

3

A FRAMEWORK FOR INTERPRETABLE STRATEGY SYNTHESIS

The problem of ISS is first mentioned in Butler et al. (2017), which identifies it as a subset of *automated game analysis*, and implicitly solves it using a specific *strategy model* for the game of Nonograms. While Butler et al. (2017) evaluate the performance of the synthesized strategies, they don't make an attempt to evaluate their interpretability. As detailed in Section §2.4, prior works have attempted to solve the problem of ISS implicitly for specific games. Subsequent works have explicitly mentioned the attempt to solve the problem of ISS, but have not situated it in the context of a formal problem definition. A shared framework that describes the problem of ISS and identifies its component elements would facilitate comparison between approaches, allowing researchers to reuse algorithms and strategy models across multiple game domains. It would also help clarify the notion of interpretability. In this chapter, I will attempt to formally define the problem of ISS by abstracting common elements and providing clear definitions, thereby answering RQ1.

3.1 Problem Definition

Definition 1 (interpretable strategy synthesis)**.** *The ISS problem* $\langle \mathcal{G}, \mathcal{M}, \mathcal{R}, \mathcal{I} \rangle$ *is that of finding a strategy* $\sigma(s;\theta^*) \in \mathcal{M}(s;\theta^*)$ *that simultaneously optimizes both* $\mathcal{R}$ *(reward function) and I (interpretability measure), as given by the equation below —*

$$\sigma^*_\theta \doteq \sigma(\cdot;\theta^*) \doteq \arg\max_\theta \mathcal{R}(\sigma_\theta) I(\sigma_\theta), \sigma \in \mathcal{M} \tag{3.1}$$

ISS also refers to finding the *optimization procedure* that finds or learns σ^*. The problem of ISS as defined is very similar to the problem of RL. It could be argued that the two are equivalent if the interpretability measure $\mathcal{I}$ were folded into the reward function $\mathcal{R}$. However, we separate the two measures to highlight the focus on interpretability in ISS.

Definition 2 (game environment)**.** *The game environment* $\mathcal{G}$ *(game environment) is a finite, episodic MDP* $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}, \mathcal{R}, \gamma \rangle$*, where —*

- $\mathcal{S}$ *is the finite set of legal game states reachable from the start state* s_0*,*
- $\mathcal{A}(s)$ *is the set of legal actions that can be taken in a state* $s \in \mathcal{S}$*,*
- $\mathcal{P}$ *is the state transition function that models the game's dynamics,*
- $\mathcal{R}$ *is the reward function describing the game's win conditions, and*
- γ *is a discount factor between* $[0,1]$

Definition 3 (strategy model)**.** *The strategy model* $\mathcal{M}$ *is a function* $\mathcal{M} : \mathcal{S} \times \Theta \to \mathcal{A}$*, where —*

- $\mathcal{S}$ *is the state space of* $\mathcal{G}$*,*
- Θ *is the associated parameter space of* $\mathcal{M}$ *(strategy model), and*
- $\mathcal{A}$ *is the action space of* $\mathcal{G}$

A strategy model $\mathcal{M}$ induces a *strategy* $\sigma(s;\theta)$ as —

$$\sigma(s;\theta) \doteq \mathcal{M}(s;\theta), \theta \in \Theta \tag{3.2}$$

I talk more about strategies and strategy models in Section §3.2.

Definition 4 (performance measure)**.** *The performance measure $\mathcal{R}$ is a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that evaluates the performance of a strategy σ in a state s by assigning a numerical score.*

As its name implies, this function measures the "performance" of a strategy in-game. It is a proxy for measuring "how good" the actions produced by a particular strategy are. It is desirable to learn strategies that maximize this measure, since that would give us strategies that lead to better game states. However, it might be useful to learn strategies that are suboptimal but are more interpretable.

Definition 5 (interpretability measure)**.** *The interpretability measure $\mathcal{I}$ is a function $\mathcal{I} : \mathcal{M} \to \mathbb{R}$ that evaluates the interpretability of a strategy by assigning a numerical score.*

Similar to $\mathcal{R}$, this function measure how "interpretable" a particular strategy is to a human player. I talk more about interpretability measures in Section §3.3.

3.2 Strategy Model

Definition 6 (strategy)**.** *A strategy σ for $\mathcal{G}$ at timestep t is a probability distribution over the available actions in a state, for a subset of states in $\mathcal{S}$ (state space). The states for which σ (strategy) is defined are termed as applicable states, and are given by $\mathrm{A}_\sigma \subseteq \mathcal{S}$.*

$$\sigma(a|s) \doteq \mathbb{P}[A_t = a | S_t = s], \forall s \in \mathrm{A}_\sigma, a \in \mathcal{A}(s) \tag{3.3}$$

In common parlance, a strategy describes a plan, or sequence of steps taken towards a particular goal (Merriam-Webster 2024). The definition of a strategy in Definition 6 borrows from the notion of a *policy* as used in RL (Sutton and Barto 2018) as well as game-theoretic notions of a strategy (Boros et al. 2012). Intuitively, this definition of a strategy describes a pattern or feature of a game state that comes up often in regular gameplay and that tends to influence the actions taken in states with that feature. A strategy is not a general policy and cannot be used to sample an action in states outside the set of applicable states.

A *strategy model* ($\mathcal{M}$) is a computational program that can be run to realize Equation 3.3. A strategy is an *instance* of a strategy model. The strategy model serves as a *template* for a strategy that a learning algorithm must output. In practice, it is tightly coupled to the learning algorithm and could incorporate domain-specific knowledge to improve its performance or interpretability. Examples of strategy models used in literature are if-else rules or programmatic scripts derived from a DSL.

3.3 Interpretability Measure

Strategy synthesis attempts to solve the same problem as RL, with the strategy model being the parametrization θ of a policy, and the synthesis algorithm being the optimization procedure. Unlike RL, strategy synthesis has the additional constraint that the learned strategy model must be *aligned* with the mental model of the player. Model matching theory posits that the "alignment of game models and external situations can facilitate the player's transfer of mental models between games and external situations" (Boyan, McGloin, et al. 2018). Alignment is framed as a matter of creating accurate facsimiles of the game, and providing the necessary tools and scaffolds to help players "make sense of" the strategy (Martinez-Garza and Clark 2017). However, it is unclear how alignment may be meaningfully evaluated.

The field of XAI uses computational metrics as a substitute for alignment between a learned strategy and a player's mental model. Interpretability in XRL is defined as "the ability to not only extract or generate explanations for the decisions of the model, but also to present this information in a way that is understandable by human (non-expert) users to, ultimately, enable them to predict a model's behavior" (Puiutta and Veith 2020). Some metrics that have been found to correlate with interpretability are the number of cognitive chunks used and the model size (Lage et al. 2019).

While computational proxies are useful for constructing optimization procedures, they do not sufficiently capture the notion of interpretability. Obtaining human-oriented measures of interpretability might be prohibitively expensive if used for training, and ought to be used only for evaluating the learned strategy models. Most current research in ISS does not define an interpretability measure, nor do they conduct an empirical evaluation of the interpretability of the learned strategy models (see Table 3.1).

Paper	Number Used			Interpretability
	Domains	Models	Algorithms	
Spronck et al. (2004)	2	1	1	✗
Mesentier Silva et al. (2016)	1	1	4	✓
Butler et al. (2017)	1	1	1	✗
Canaan et al. (2018)	1	1	1	✗
Freitas et al. (2018)	1	1	1	✗
Mariño, Moraes, et al. (2021)	1	1	1	✗
Krishnan and Martens (2022a)	1	1	1	✗
Mariño and Toledo (2022)	1	1	1	✗
Medeiros et al. (2022)	2	1	2	✗

Table 3.1: List of works in ISS demonstrating their domain-specificity and lack of interpretability evaluation.

CHAPTER

4

STRATEGIES AS FIRST-ORDER LOGIC RULES

In this chapter, I describe how strategies can be modeled as first-order logic (FOL) rules. I describe how FOL rules can be used to model strategies, and how they can be learned from data using a symbolic ML technique called Inductive Logic Programming (ILP). I then apply this technique to learn strategies for the game of chess, describing first how chess strategies may be modeled as FOL rules (Section §4.4), and then how these rules may be learned using ILP (Section §4.5). Finally, I describe an improvement to the proposed learning method that uses precision and recall to constrain the learning process (Section §4.6). In doing so, I answer RQ2 and its associated RQs 2a, 2b and 2c.

4.1 Background

4.1.1 Inductive Logic Programming

Inductive Logic Programming (ILP) is a form of symbolic ML that aims to induce a *hypothesis* (a set of logical rules) that generalizes given training examples. It can learn human-readable hypotheses from smaller amounts of data than NN-based models (Cropper and Dumančić

2022).

An ILP problem is specified by three sets of Horn clauses —

- B, the background knowledge,
- E^+, the set of positive examples of the concept, and
- E^-, the set of negative examples of the concept.

The ILP problem is to induce a hypothesis $H \in \mathcal{H}$ (an appropriately chosen hypothesis space) that, in combination with B (background knowledge), *entails* all the positive examples in E^+ (positive examples) and none of the negative examples in E^- (negative examples). Formally, this can be written as —

$$\forall e \in E^+, H \cup B \models e \text{ (i.e., } H \text{ is } \textit{complete)}$$
$$\forall e \in E^-, H \cup B \not\models e \text{ (i.e., } H \text{ is } \textit{consistent)}$$

To make the ILP problem more concrete, I reproduce an example due to Cropper and Dumančić (2022).

Example 1. *E^+ and E^- contain positive and negative examples of the target* `last` *relation respectively. B contains background knowledge, i.e., clauses that might be useful in inducing a hypothesis for* `last`.

$$E^+ = \left\{ \begin{array}{l} \texttt{last([m,a,c,h,i,n,e], e).} \\ \texttt{last([l,e,a,r,n,i,n,g], g).} \\ \texttt{last([a,l,g,o,r,i,t,h,m], m).} \end{array} \right\}$$
$$E^- = \left\{ \begin{array}{l} \texttt{last([m,a,c,h,i,n,e], m).} \\ \texttt{last([m,a,c,h,i,n,e], c).} \\ \texttt{last([l,e,a,r,n,i,n,g], x).} \\ \texttt{last([l,e,a,r,n,i,n,g], i).} \end{array} \right\}$$
$$B = \left\{ \begin{array}{l} \texttt{empty(A) :- ...} \\ \texttt{head(A,B) :- ...} \\ \texttt{tail(A,B) :- ...} \end{array} \right\}$$

From this information, I could induce a hypothesis for `last` *as —*

$$H = \left\{ \begin{array}{l} \texttt{last(A,B) :- head(A,B),tail(A,C),empty(C).} \\ \texttt{last(A,B) :- tail(A,C),last(C,B).} \end{array} \right\}$$

4.2 First-Order Logic Rules as Policies

In this section, I describe how a policy π (policy) for a MDP $\mathcal{G} \doteq \langle \mathcal{S}, \mathcal{A}(s), \mathcal{P}, \mathcal{R}, \gamma \rangle$ may be modeled as a set of FOL rules. I first describe the policy model, and how π may be translated into a set of FOL rules. I then describe how these rules may be learned from data.

4.2.1 Rule-based Policy Model

Given a MDP $\mathcal{G}$ as $\langle \mathcal{S}, \mathcal{A}(s), \mathcal{R}, \mathcal{P}, \gamma \rangle$, we can model a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}(s)$ as a set of FOL *policy rules* expressed in a particular *predicate vocabulary* $\mathcal{V}$, as shown in Figure 4.1.

```
policy(State, Action) ←
    feature_1(···),
    feature_2(···),
    ⋮
    feature_n(···)
```

Figure 4.1: Our FOL-based policy model expressed in Prolog pseudocode. Every `feature_i` clause is a rule defined in a predicate vocabulary $\mathcal{V}$.

Here, a `State` is a Prolog list of state predicates `state_i` $\in \mathcal{V}$ that describe the current state of the MDP $\mathcal{G}$. The feature predicates `feature_i` $\in \mathcal{V}$ are rules defined in the predicate vocabulary that describe higher-order features of the state that are relevant to the policy. The `Action` is the action to be taken in the state described by the `State`, and may itself be a Prolog list of action predicates `action_i` $\in \mathcal{V}$. The predicate vocabulary must be hand-authored for each domain, and the features defined influence the expressiveness and performance of the policies learned (see Section §4.6).

We can obtain an action from a policy rule by providing it as a query to the Prolog interpreter. When this is done with a grounded instance of `State` and non-ground `Action` variables, Prolog will attempt to *unify* the latter with ground action (Sterling and Shapiro 1994). The variable binding(s) (i.e., actions) it finds are treated as the action distribution defined on the state, with every found action being equi-probable, and every other action accorded a probability of 0.

A single policy rule, or even a set of policy rules, does not represent a complete policy for $\mathcal{G}$. This is because we might encounter a state for which no policy rule matches. In this case, our model cannot take an action. There might also be positions to which multiple policy rules apply, in which case an arbitration process for selecting a single action among the various suggestions is not obvious.

4.2.2 Learning Rule-based Policies

Given a MDP $\mathcal{G}$ and a training set of state-action pairs $E = \{(s_1, a_1), (s_2, a_2), \ldots, (s_n, a_n)\}$ drawn from a target policy π_T and expressed using $\mathcal{V}$, we can learn a policy π by learning a set of FOL policy rules that approximate π_T by modeling the learning problem as a ILP task $\langle E^+, E^-, B\rangle$. E^+ is the set of positive examples, and is drawn from E. E^- is the set of negative examples, and must be generated. One approach to this is by augmenting E as follows: for every state-action pair $(s, a) \in E$, we generate a negative example by randomly selecting an action $a' \in \mathcal{A}(s)$ such that $a' \neq a$. B is the background knowledge, and is exactly $\mathcal{V}$. With the problem of learning a FOL policy modeled as an ILP problem, we can apply any ILP algorithm in order to learn the policy rules.

4.3 Application to Chess

4.3.1 Chess

The earliest known precursor to the game of chess was a game called *chaturanga*, first played in India in the 7th century CE. Its modern form took shape in Spain in the 15th century (Averbakh 2012). This long history has led to a wealth of study and analysis of the game over time, which has only been accelerated with the advent of chess-playing computers (chess *engines*). This is useful because it provides a rich set of existing player strategies to compare against.

Chess' extensive history and appeal as a test of intelligence has led it to be called the *drosophila*[1] of artificial intelligence (McCarthy 1990). It has been a mainstay of Artificial Intelligence (AI) research from the invention of the digital computer (Claude 1950) to the neural network (NN) revolution (Silver, Hubert, et al. 2018). There has been extensive research investigating the idea of using patterns to guide a computer to play chess. Section §**??** examines some of this research.

For the game of chess, I present a model for a human-readable chess strategy, inspired by documented chess tactics, for playing chess (Krishnan and Martens 2022b). I use this to

[1] fruit fly; easily bred and thus extensively used in genetics research

model known chess tactics from literature. Using the novel metrics of coverage and divergence, I show that these tactics produce better moves than a random baseline. This answers RQ2a. I next present a system that uses a modified ILP system to optimize divergence for learning chess strategies from gameplay data (Krishnan and Martens 2022a). This system can learn chess strategies from positions drawn from play traces of human vs. human games. It requires no more domain knowledge than the base rules of chess. I show that the learned strategies generalize well to real-world chess positions and produce moves that outperform a random player. This answers RQ2b. Finally, I improve upon this system by introducing new constraints to reduce the search space, and a new predicate vocabulary. Using an ablation study, I show that these changes significantly ($p < 0.01$) reduce the difference in strength between the moves suggested by the chess strategies, and the ground truth (Krishnan, Martens, and Jhala 2023). This answers RQ2c.

4.3.2 Strategy Model for Chess

Tactics in chess are maneuvers that take advantage of short-term opportunities (Seirawan 2005b, Chapter 1). They are a move or series of moves that bring about an immediate advantage for the player. They are identified by *motifs* – positional patterns that indicate whether a tactic might exist in the given position. An example of a tactic is the *fork*, where a piece simultaneously attacks two opponent pieces at once (see Figure 4.2). Tactics are very important in chess. The German chess master Richard Teichmann is attributed with the quote "Chess is 99% tactics". While not exactly true, players nevertheless devote a significant amount of attention to the study and recognition of tactics in games.

Our strategy model for chess uses the concept of tactics, and I will refer to an instance of our strategy model for chess as a *chess tactic* throughout the rest of this chapter. Ambiguity with the notion of a tactic from chess literature will be resolved from context. Formally, I define our strategy model for chess as a first-order logic (FOL) rule expressed in Prolog (Wielemaker 2003) using a domain-specific predicate vocabulary . As seen in Figure 4.3, the rule head of the tactic consists of the variables `Position`, `From` and `To`. The input state is described by `Position`, which is also expressed in FOL using our predicate vocabulary. `From` and `To` describe the output action, namely, the move which begins from the square `From` and ends on the square `To`. This replicates the long algebraic notation for moves used in the Universal Chess Interface (UCI) protocol, an open communication protocol allowing chess engines to interact with user interfaces (Kahlen 2004).

Our usage of FOL to model chess tactics is motivated by the following reasons —

[2] https://lichess.org/training/4pEpj

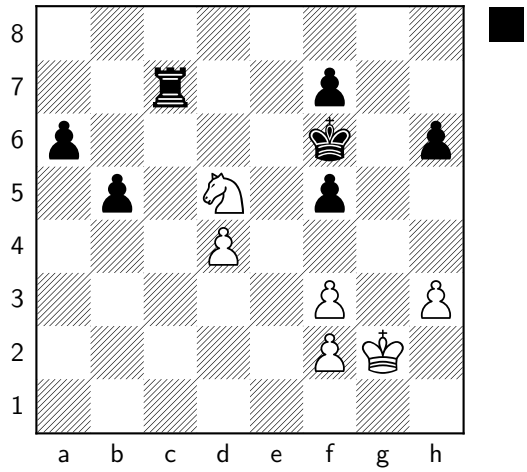

Figure 4.2: An example of a fork from a Lichess puzzle[2]. The white knight on d5 simultaneously checks the opponent king on f6 and attacks the rook on c7.

```
tactic(Position, From, To) ←
        feature_1(···),
        feature_2(···),
        ⋮
        feature_n(···)
```

Figure 4.3: Our tactic model expressed in Prolog pseudocode. Every `feature_i` clause is a rule defined in the predicate vocabulary.

1. Chess tactics are an important concept that human players use to think about chess (Szabo 1984) and are useful in chess education (Gobet and Jansen 2006).

2. FOL has been extensively used to model chess patterns (see Section §**??**).

4.3.3 Metrics to Evaluate Chess Strategy Models

Having a computational representation of a chess strategy (tactic) is not enough; we need a computational evaluation metric to guide any potential learning system towards "good" tactics and away from "bad" tactics. Characteristics of good tactics are – 1) they produce *strong* moves, i.e., they produce moves that tend to lead to winning game states, and 2) they are encountered relatively frequently in the states reached during typical gameplay. To realize these goals, I introduce two metrics – *coverage* and *divergence*.

4.3.3.1 Coverage

A tactic σ's coverage for a set of positions P is calculated as —

$$P_{\mathrm{A}} \doteq P \cap \mathrm{A}_{\sigma} \tag{4.1}$$

$$\text{Coverage}(\sigma, P) \doteq \frac{|P_{\mathrm{A}}|}{|P|} \tag{4.2}$$

Coverage is the fraction of positions in a given set to which the tactic is *applicable*. If P is representative of positions encountered in games, coverage is a useful measure of how likely it is that the tactic can be used to make moves in games. A low coverage value indicates that the tactic is *situational*, whereas a high coverage value indicates that the tactic is general.

4.3.3.2 Divergence

In order to find good moves, chess engines (chess-playing agents) must have some way of evaluating a board position to produce a numerical score. A good move is one which results in the maximum increase of this evaluation. In RL terms, a chess engine must have a good state-value function V_{π}.

Classical chess engines used a collection of hand-crafted heuristics to evaluate a position (Bijl and Tiet 2021). Modern NN-based engines use a combination of heuristics and NN models trained on millions of positions to produce evaluations. These evaluations are standardized to be reported in Cp (centipawn), or roughly $1/100^{\text{ths}}$ of the value of a pawn (Guid and Bratko 2017). Positive values indicate white is favored, and negative values indicate black is favored. A position evaluated to be +1 roughly corresponds to an advantage of a single pawn for white. If a sequence of moves leading to a forced checkmate is found (the condition for winning in chess), then that is reported instead.

To measure the quality of moves suggested by a tactic, I extend a metric previously used to analyze world chess champions (Guid and Bratko 2006; Guid and Bratko 2011; Romero 2019). A chess engine E is used that can provide a position evaluation function $v_E(s)$. From this, I can obtain a move evaluation function $q_E(s,a)$ as follows —

$$q_E(s,a) = \sum_{s',r} \mathcal{P}(s',r|s,a)[r + v_E(s')] \tag{4.3}$$

$$= v_E(s'), s' \text{ is non-terminal} \tag{4.4}$$

Equation 4.4 follows from Equation 4.3 since rewards in chess are 0 for non-terminal states,

$\gamma = 1$, and chess rules are deterministic. A chess engine could evaluate a position to be a 'Mate in X' rather than a numerical score. In this case, I assign an arbitrary large value to the evaluation.

Given two moves a_1, a_2 made in a position s, I can calculate their difference $d_E(s, a_1, a_2)$ as —

$$d_E(s, a_1, a_2) \doteq \mid q_E(s, a_1) - q_E(s, a_2) \mid \tag{4.5}$$

I can now define the *divergence* of a tactic from a set of examples P as the average difference in evaluation between the moves suggested by the tactic and the ground truth move —

$$\begin{aligned} &\text{Divergence}_E(\sigma(\cdot), P) \doteq \\ &\frac{1}{|P_\text{A}|} \sum_{(s,a_1) \in P_\text{A}} \sum_{a_2 \in \mathcal{A}(s)} \sigma(a_2|s) d_E(s, a_1, a_2) \end{aligned} \tag{4.6}$$

Divergence is also reported in units of Cp. Divergence of a tactic from the ground truth is low and close to 0 when its suggestions are similar in engine evaluation to the ground truth moves, and takes on large values when it differs significantly. Divergence is a useful metric to measure how closely a tactic approximates a reference policy.

4.4 Evaluation of Known Chess Tactics

In a proof-of-concept study of the ISS framework proposed in Chapter 3, I show how the symbolic strategy models (tactics) for chess as described in Section §4.3.2 can be learned. I use patterns learned by an existing ILP system called PAL (Morales 1992) to derive these tactics. I then present an evaluation of a set of tactics obtained from PAL against a random baseline using the metrics defined in Section §4.3.3. In showing that the learned tactics resemble a weak chess-playing program, but are unable to outperform a strong one, I answer RQ2a.

4.4.1 PAL System

The Patterns and Learning (PAL) system was introduced in Morales (1992). It attempts to use ILP to synthesize patterns for chess play, which are expressed using a subset of Horn clause logic. It contributes a predicate vocabulary for expressing these patterns and chess positions as Horn clauses. The pattern-learning problem is framed as an ILP problem, for which a heuristically-constrained version of the relative least general generalization (rlgg) algorithm is used to induce plausible hypotheses. Patterns learned can be *static* and not involve any piece movement, or be

```
can_check(S1,P1,(X1,Y1),S2,king,(X2,Y2),(X3,Y3),Pos1) ←
        contents(S1,P1,(X1,Y1),Pos1),
        contents(S2,king,(X2,Y2),Pos1),
        other_side(S1,S2),
        ¬ in_check(S2,(X2,Y2),P1,(X1,Y1),Pos1),
        make_move(S1,P1,(X1,Y1),(X3,Y3),Pos1,Pos2),
        in_check(S2,(X2,Y2),P1,(X3,Y3),Pos2).
```

Figure 4.4: PAL rule for the `can_check` pattern. A piece (P1) can check the opponent's King after moving to (X3,Y3).

dynamic and describe multi-move tactics. I expand upon how the PAL system formally defines and synthesizes these chess patterns.

4.4.1.1 Pattern Formalism

$$\text{Head} \rightarrow D_1, D_2, \cdots, D_n, F_1, F_2, \cdots, F_m$$

A pattern in PAL is formally defined as a non-recursive Horn clause as shown in Equation 4.4.1.1, where —

- Head is the head of the pattern definition
- The D_i are "input" predicates used to describe the position and represent pieces involved in the pattern
- The F_j are instances of definitions which are either provided as background knowledge or learned by PAL, and represent the conditions (relations between pieces and places) to be satisfied by the pattern.

An example of a *checking move* pattern, where a move that puts the opponent king in check is suggested, is reproduced from the paper in Figure 4.4. A key predicate is `make_move`, which determines whether a pattern is static or dynamic. The `contents` predicates are used to describe the position on the board. The remaining predicate definitions are provided as background knowledge.

4.4.1.2 Pattern Synthesis

The input to the PAL generalization algorithm is a set of pattern definitions (both predefined and learned) along with a description of a chess position (as ground unit clauses). The algorithm extends the method due to Buntine (1988) for constructing the rlgg of two clauses to multiple

clauses. It uses the following constraints and heuristics to limit hypothesis size and increase the algorithm's generalization steps —

- Disallowing variables in the head or body of a rule which are not *connected* to a literal i.e., not equal to a variable of that literal
- Labeling constants occurring in the ground literals of a rule body to make patterns piece-invariant
- Restricting the legal moves from a position to only be those which introduce a new predicate name or remove an existing predicate name

PAL uses an automatic example generator to manually guide the generalization algorithm towards learning desired concepts. Given an example of the target concept, the generator *perturbs* the example to create a new example for which a classification label must be provided. To restrict the example space searched, the automatic example generator attempts to generate examples which specialize the current hypothesis in case of a prior positive example, or generalize it in case of a prior negative example. I refer interested readers to Morales (1992) for further details.

4.4.2 Methodology

I use the PAL system to synthesize tactics. I select seven patterns that PAL was shown to learn, and modify them to output a move suggestion. These patterns and their verbal definitions are listed in Table 4.1. All patterns learned other than `pin` are 1-ply *dynamic* patterns, which means they include a single `make_move` predicate in the rule body looking ahead one move. I modify these patterns to introduce a `suggestion` predicate with the same variables as `make_move`. For `pin`, which is a static pattern as learned by PAL, I convert it into a dynamic pattern as shown in Figure 4.5 and introduce the `suggestion` predicate in the same way.

```
pin(S1,P1,(X1,Y1),S2,king,(X2,Y2),S2,P3,(X3,Y3),(X4,Y4),Pos1) ←
    sliding piece(P1,(X1,Y1),Pos1),
    make_move(S1,P1,(X1,Y1),(X4,Y4),Pos1,Pos2)
    sliding piece(P1,(X4,Y4),Pos2),
    stale(S2,P3,(X3,Y3),Pos2),
    threat(S1,P1,(X4,Y4),S2,P3,(X3,Y3),Pos2),
    in_line(S2,king,(X2,Y2),S2,P3,(X3,Y3),S1,P1,(X4,Y4),Pos2).
```

Figure 4.5: Modified PAL rule for the `pin` pattern to convert it into a tactic

Pattern	Definition
`can_threat`	A piece (P1) can threaten another piece (P2) after making a move to (X3,Y3)
`can_fork`	A piece (P1) can produce a fork to the opponent's King and piece (P3) after making a move to (X4,Y4)
`can_check`	A piece (P1) can check the opponent's King after a moving to (X3,Y3)
`discovered_check`	A check by piece (P2) can be "discovered" after moving another piece (P1) to (X4,Y4)
`discovered_threat`	A piece (P1) can threaten an opponent's piece (P3) after moving another piece (P2) to (X4,Y4)
`skewer`	A King in check by a piece (P1) "exposes" another piece (P3) when it is moved out of check to (X4,Y4)
`pin`	A piece (P3) cannot move because it will produce a check on its own side by piece (P1)

Table 4.1: Patterns learned by the PAL system that are used to create tactics

4.4.3 Evaluation

To evaluate the synthesized tactics, I investigate whether they suggest good moves to play. I do this by measuring coverage and divergence for each of our tactics over a set of positions using both a strong and a weak reference engine. For our strong reference engine, I use *Stockfish 14* (The Stockfish team 2021), the winner of the Top Chess Engine Championship (TCEC) 2020 Championship (Haworth and Hernandez 2021). For our weak reference engine, I use Maia Chess (McIlroy-Young et al. 2020), a chess engine trained to produce human-like moves. I use the `maia1` model (hereafter Maia-1100), which is targeted toward an Elo rating of 1100 (a measure of relative playing strength, and roughly equal to a beginner). I limit the search depth to 1-ply (i.e., no look-ahead) for both Stockfish 14 and Maia-1100. This choice follows from the capabilities of our tactic model and the architecture of the Maia family of chess engines in that they do not conduct tree search. As a baseline, I use a random tactic which is applicable to all positions and produces a random legal move in the position. I limit the number of suggestions from a tactic to 3. For ease of implementation, I manually translated the tactics from Prolog definitions to Python functions. I use 5000 games from the January 2013 archive of standard rated games played on lichess.com (lichess.org 2021). For each game, I generate positions by iterating through the move list, making the move, and adding the resulting position to the evaluation set. In total, I generate 325,830 positions.

4.4.4 Results and Analysis

I summarize the results of our evaluation in Table 4.2.

From the high coverage values obtained, I conclude that tactics like `can_threat` and `discovered_threat` are too general, whereas tactics like `discovered_check` are too specific. Tactics like `can_check`, `can_fork` and `skewer` strike a balance between these extremes.

From the divergence metrics calculated using Maia-1100 (our weak engine), I see that most of our tactics have lower divergence scores than our random baseline, indicating that they tend to produce moves which are evaluated somewhat similarly to a weak engine's best moves. For Stockfish 14, however, all our tactics have higher divergence scores than random, indicating that they do not tend to produce moves similar to a strong engine. Thus, I qualitatively conclude that our tactics resemble that of a beginner chess player.

Tactic	**Coverage**	**Divergence**	
		SF14	Maia
`can_threat`	0.96	378.94	9.22
`can_check`	0.45	549.19	4.02
`can_fork`	0.32	676.45	4.67
`discovered_check`	≈0	338.55	18.64
`discovered_threat`	0.96	375.97	1.19
`skewer`	0.22	748.4	5.41
`pin`	0.79	526.45	4.9
`random`	**1**	**328.09**	**8.28**

Table 4.2: Coverage and Divergence for each tactic

4.4.5 Conclusion and Future Work

I have described a symbolic sub-policy model for chess inspired by the pattern-action model of chess tactics. I have used patterns learned by an ILP system to construct these tactics. I have contributed a metric for measuring the divergence of these tactics to a reference chess-playing agent. I evaluated a set of tactics learned by a chess pattern learning system using our metric to find that they resembled a weak engine, but were not similar to a strong one.

I use patterns learned by PAL to obtain our tactics. However, PAL uses manual labeling of generated examples to learn specific concepts, and requires additional effort to convert the learned patterns into tactics for our model.

Our tactic model is loosely inspired by how chess tactics are learned and practiced. However, our tactics are limited to looking 1-*ply* in the future i.e., they can only recognize the presence of a matching pattern in the immediately next position. Many chess tactics suggest *combinations* of moves, a series of moves where the matching pattern shows up only in a particular sequence (see Figure 4.6). Extending the tactic model to express and recognize such combinations will be a useful avenue for future work. I also wish to investigate the expression of longer-term plans from chess literature like center control and pawn structure using tactics.

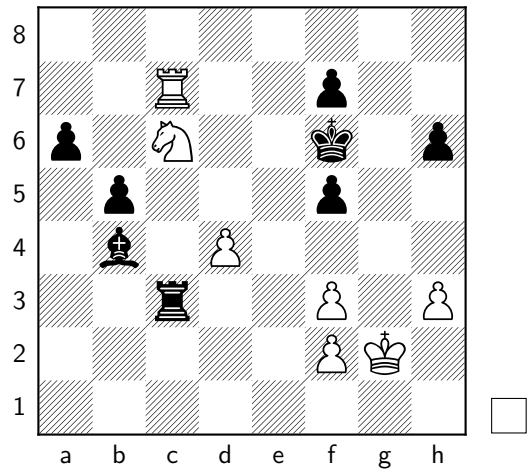

Figure 4.6: An example of the limitations of our 1-ply `can_fork` tactic. White has no immediate forking move here, leading to the tactic not matching. However, if they play **1. Nxb4**, then Black's best response is **1. ... Rxc7** which allows a fork with **2. Nd5+** leading to the capture of the rook.

Our appeal to the interpretability of these tactics rests on similar claims made regarding the interpretability of rule-based strategies. Future work will involve rigorously testing these assumptions with user studies using evidence-based measures of interpretability (Lage et al. 2019; Kliegr et al. 2021). Specifically, I wish to investigate the ease of learning and applying these tactics in real games played by human players.

4.5 Learning Chess Tactics using Inductive Logic Programming

In this work, I present a system that integrates a modified ILP system with coverage and divergence metrics for learning tactics. My system can extrapolate chess tactics from positions

drawn from play traces of human vs. human games. It requires no more domain knowledge than the base rules of chess. I show that the learned tactics generalize well to real-world chess positions and produce moves that outperform a random player. I discuss some limitations of our approach and conclude with directions for future work. In so doing, I answer RQ2b.

4.5.1 Popper

Popper is an ILP system that implements an approach called *learning from failures* (Cropper and Morel 2021). It operates in three stages: generate, test and constrain. Given an ILP problem, it generates a candidate solution, tests it against examples in the training set, and formulates constraints based on the outcome of failed examples to cut down the solution space. These three stages are repeated until a solution is found. Popper also allows for specifying hypothesis biases to further constrain the solution space based on domain knowledge of the problem being solved.

Popper uses two general types of constraints – *generalization* and *specialization*. Assume an ILP problem $\langle E^+, E^-, B\rangle$ and a generated hypothesis H. If H entails a negative example, then H is too general and so I can prune generalizations of it. Similarly, if H does not entail a positive example, it is too specific, and I can prune specializations of it. The notions of the generalization and specialization of a hypothesis are defined in terms of *clausal subsumption*, and I refer readers to the original paper for formal definitions of these terms. Popper supports providing a *language bias* to influence the hypothesis space.

4.5.2 Methodology

I formalize the problem of learning a chess tactic model as an ILP problem $\langle E^+, E^-, B\rangle$. E^+ is a set of $\langle$position, move$\rangle$ tuples where the move made in the position is drawn from a target policy. E^- is a set of $\langle$position, move$\rangle$ tuples where the move made in the position is *not* drawn from the target policy. B is the predicate vocabulary used to express positions, moves and chess tactics. I want to learn a hypothesis H that maximizes the number of examples entailed in E^+ and minimizes the number of examples entailed in E^-. The hypothesis H is our chess strategy σ.

Since I might not be able to find a single hypothesis that meets our objectives, I instead learn a collection of tactics and evaluate their usefulness using the metrics of *coverage* and *divergence*.

I use training examples drawn from real games played by humans online. Each training example is a $\langle$position, move$\rangle$ tuple, where position is the board state converted to FOL using a hand-engineered predicate vocabulary, and move is the move made in that position, also

represented in FOL. I define this predicate vocabulary in the background knowledge, along with predicates representing the board state and relationships between squares. Our choice of predicates is motivated by the ability to use them to express tactics from chess literature, like the pin or the fork. I also introduce a foreign predicate implemented externally to represent a legal move. See Figure 4.7 for an expression of the *fork* tactic from chess literature in this model. Our background knowledge design borrows from that of the PAL system (Morales 1992). I refer readers to the Appendix C for the complete list of predicates in our background knowledge.

I select Popper as the ILP system to learn tactics with. Popper searches for the hypothesis that maximizes the F1 score when evaluated against the examples. However, I wish to learn *multiple* tactics that might not cover the entire example set. Our proposed system supports learning multiple tactics. I do so by modifying the *generate* and *constrain* stages of the Popper ILP system in the following ways —

- *generate*: modified to only generate tactics that produce legal moves
- *constrain*: prevent further *specializations* of a tactic that does not match any position in the training set from being generated

```
fork(Position,From,To) ←
    make_move(From, To, Position, NewPosition),
    attacks(To, Square1, NewPosition),
    attacks(To, Square2, NewPosition),
    different_pos(Square1, Square2),
```

Figure 4.7: An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary. The first `attacks` clause states that the piece at `To` attacks the opposing piece at `Square1` in the current position.

4.5.3 Evaluation

In this section, I describe the procedure used to evaluate the tactics produced by our learning system. I describe our testing and training datasets, how I generate the tactics used for evaluation, and the metrics I use to measure the performance of the learned tactics. Since I do not have a reference set of existing tactics expressed in our predicate vocabulary to compare against, I choose to compare against the following baseline tactics —

- **random move tactic**: makes a random legal move in a given position, and is applicable to all positions
- **ground move tactic**: replicates the move made in the ground truth example, and is applicable to all positions
- **engine move tactic**: makes the best move suggested by an engine, and is applicable to all positions. I use two engines – Stockfish 14 and Maia-1600.

Hypothesis 1. *The average divergence of the tactics learned by the system is lower than that of a tactic that makes random moves.*

4.5.3.1 Dataset

To source our ⟨position, move⟩ training examples, I use a collection of games played by human players on the Internet chess server Lichess[3]. Specifically, I use the January 2013 archive of standard (played with regular chess rules) rated (users stand to gain or lose rating points based on the outcome) games played on the website (lichess.org 2021). The archive consists of 121,332 games, with players having an Elo rating of 1601 ± 289 (new players start at 1500 (Lichess 2021)). To generate N examples, I randomly sample N games and select a single random position from that game, along with the move made in that position. I select positions beginning from move 12 (Guid and Bratko 2006; Romero 2019), and exclude games which did not end normally or by time forfeit. Using this procedure, I generated a dataset of 100 examples. I found that using more training examples did not produce new tactics since the hypothesis space was exhausted. For testing, I use the February 2013 archive of 123,961 games and with an Elo rating of 1595 ± 298 to generate 10 testing examples.

4.5.3.2 Training

Using the bias settings provided by Popper, I limit the size of the learnable hypotheses to a maximum of one clause, five variables and five body literals. Empirically, I find that this strikes a good balance between learning time and quality of learned tactics. I run our proposed method until no more solutions are found. I obtain a list T of 837 tactics.

4.5.3.3 Performance Metrics

To measure the performance of our tactics on the test data, I replicate the evaluation procedure as in Section §4.4.3, but use Maia 1600 instead of Maia 1100 to better match the distribution of Elo rating in our dataset.

[3] https://lichess.org/

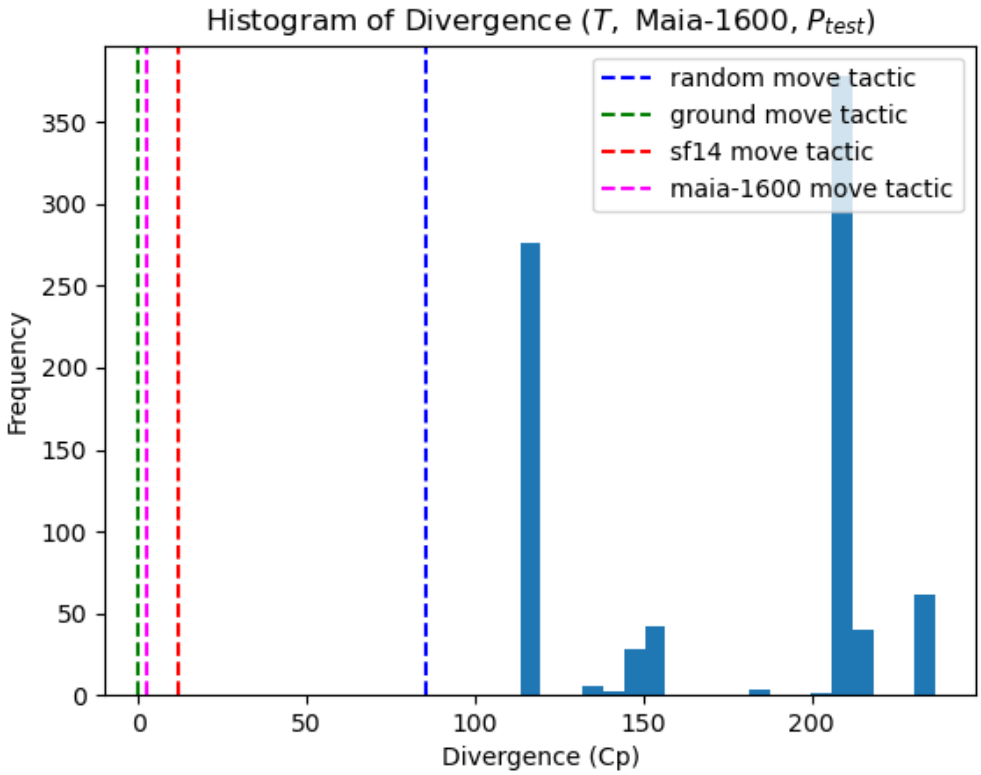

Figure 4.8: Divergence histogram for T evaluated using Maia-1600

4.5.4 Results

Based on the data obtained, I report the results for each of accuracy, coverage and divergence as a histogram with 20 buckets.

From the coverage values in Figure 4.10, I see that all the tactics learned by our system cover 30% - 60% of the test set. I conclude that the tactics learned by our system are moderately likely to be applicable to positions that arise in real games.

From the accuracy histogram in Figure 4.11, I see the tactics learned by our system are only marginally more accurate at predicting moves in the test set than the random baseline. Overall, the highest accuracy of a learned tactic in T is 42% compared to the 10% accuracy of the random baseline.

From the divergence histogram for T calculated using Maia-1600 (Figure 4.8), I see that Maia-1600 evaluates the learned tactics as having *greater* divergence from ground that the random baseline. However, from Figure 4.9, I see that the stronger Stockfish 14 evaluates the same tactics as having *lower* divergence from ground. I can conclude that the learned tactics are better approximating Stockfish 14's policy as compared to Maia-1600's policy.

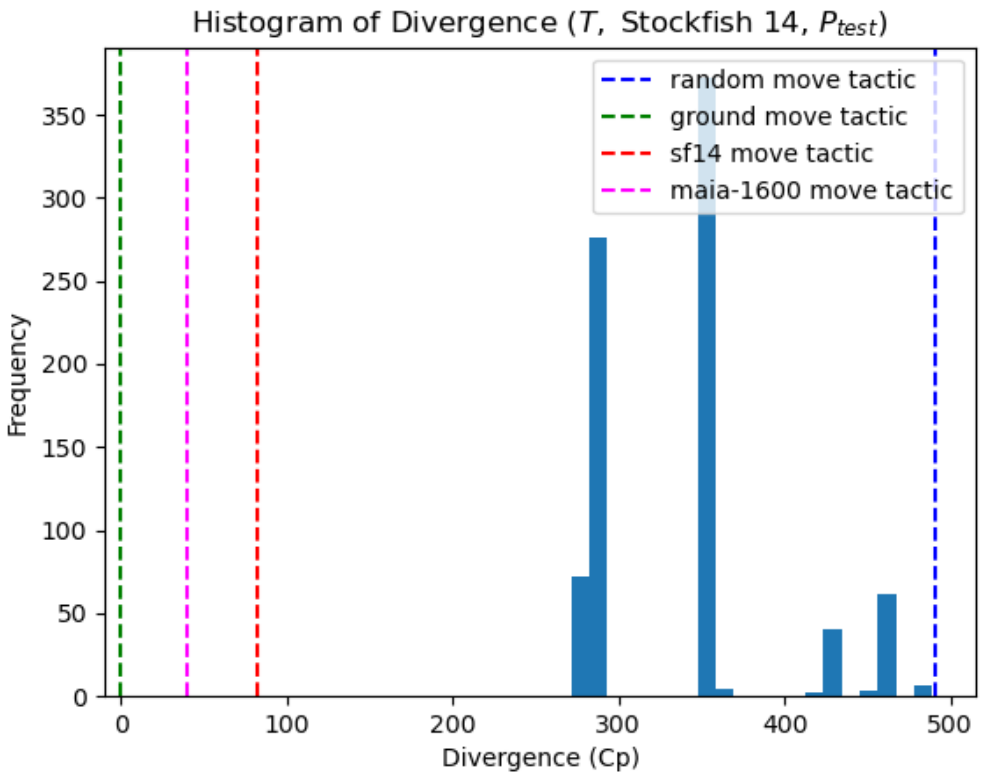

Figure 4.9: Divergence histogram for T evaluated using Stockfish 14

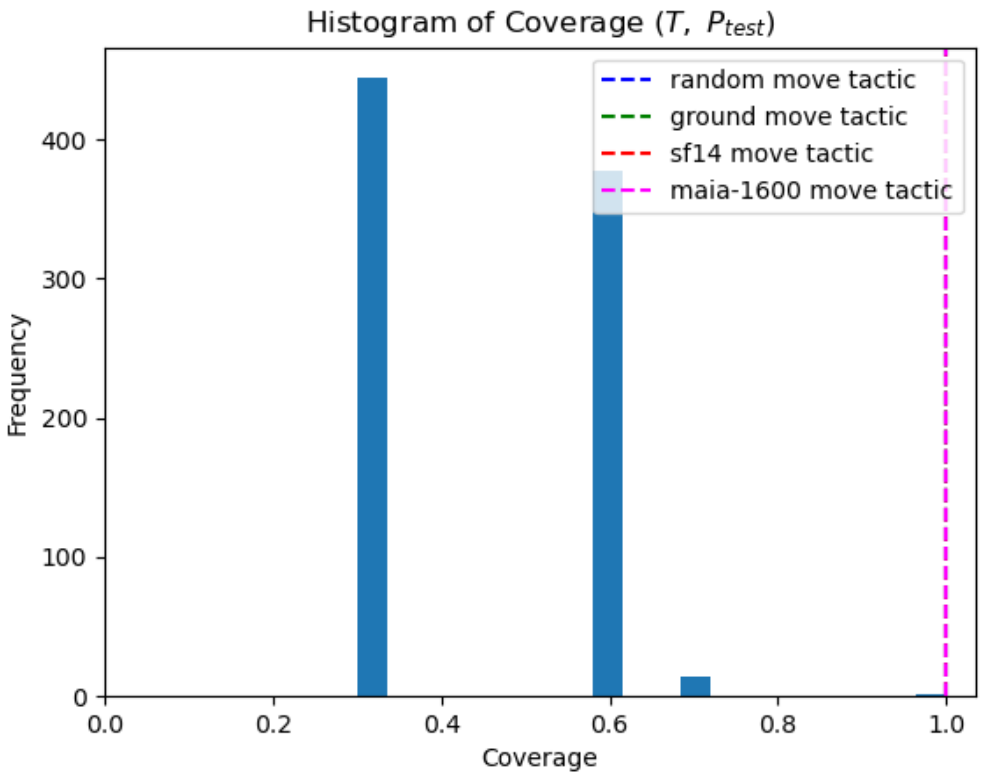

Figure 4.10: Coverage histogram for T

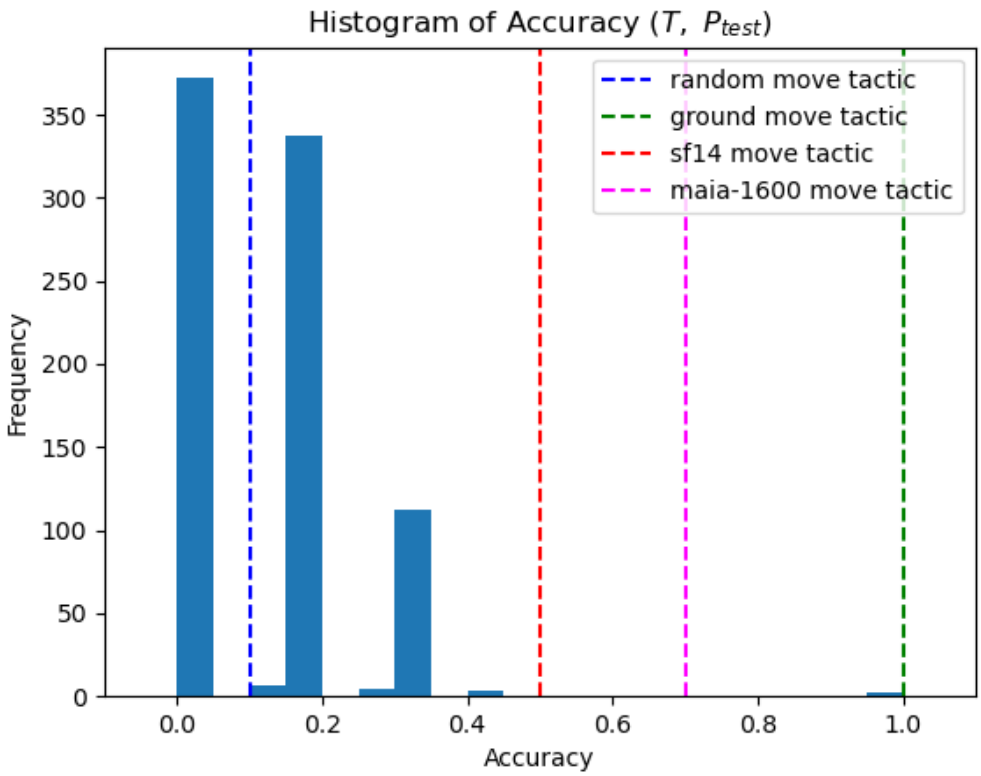

Figure 4.11: Accuracy histogram for T

4.5.4.1 Qualitative Analysis

I present three tactics learned by our system to further analyze qualitatively.

4.5.4.1.1 Low Divergence The tactic in Figure 4.12 was chosen from those having the least divergence from ground as measured by Stockfish 14. It can be interpreted as follows — if one of your pieces is attacking the opponent's piece (`attacks`(From, Square1, Position)), move it instead to a square which puts it in line with two of your opponent's pieces (as seen in the predicate `behind`(To, Square1, Square2, Position)). To the author's knowledge, this does not represent an existing tactical idea in chess.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  attacks(From,Square1,Position),
  behind(To,Square1,Square2,Position),
  different_pos(Square1,Square2).
```

Figure 4.12: A learned tactic with low divergence from ground as evaluated by Stockfish 14.

4.5.4.1.2 Highest Accuracy The tactic in Figure 4.13 was chosen from that having the highest accuracy. It can be interpreted as follows — if one of your pieces is attacking your opponent's piece (`attacks`(From, Square1, Position)), move it instead to a different square. This somewhat resembles the idea of a tactical retreat in chess. It can be thought of as a more general tactic than that in Figure 4.18.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  attacks(From,Square1,Position),
  different_pos(From,To),
  different_pos(From,Square1).
```

Figure 4.13: A learned tactic with the highest accuracy.

4.5.4.1.3 Variable Reshuffling The tactic in Figure 4.14 represents a possibility in the tactic hypothesis space that is merely a permutation of the variables in the rule. It is difficult to interpret this in terms of chess, and is very likely meaningless. It points to opportunities to restrict the hypothesis space using language biases or constraints to prevent such tactics from being generated.

```
f(Position,From,To) ←
  legal_move(From,To,Position),
  behind(From,To,Square1,Position),
  behind(From,Square2,Square3,Position),
  behind(From,Square3,Square2,Position),
  behind(From,Square2,To,Position).
```

Figure 4.14: A learned tactic with meaningless variable reshuffling.

4.5.5 Discussion

Since our system uses ILP as its learning method, it inherits many benefits and challenges associated with the paradigm. Our learning system requires carefully selected language biases in the form of background knowledge in order to learn efficiently (Cropper and Dumančić 2022). Designing predicates suitable to game-like domains will make searching for better tactics more efficient. The learned tactics are also interpretable (Muggleton et al. 2018), and can be added back into the background knowledge to allow lifelong learning (Cropper and Tourret 2020). However, work needs to be done in this area in the context of games.

The learned tactics are able to beat a random baseline in producing moves similar to a beginner player. However, they still have high divergence from player moves, have low accuracy and do not suggest the same moves, indicating that they do not yet adequately represent a human beginner player. Improving the learning algorithm to penalize tactics which do not match the ground truth moves could lead to better results.

Given that our tactics serve as an interpretable model of human players, they could also be used to interpret chess engines, and serve as a general technique for XAI. However, further work is required to learn tactics of playing strength comparable to top engines.

In order to apply our work to another game domain, one would need to engineer a set of Prolog-based features rich enough to express a single-move policy. Due to its discrete, grid-like

and turn-based nature, chess lends itself to representation using Prolog more easily than other games. Thus, it was easy to write features to express chess tactics since there existed a wealth of tactics from extant literature. For other game domains, it would be helpful to review existing strategies developed by the player community and design features to express them in Prolog. The features involved in expressing known tactics will hopefully prove general enough to learn effective unknown tactics as well. For new games which do not have an extensive knowledge base of known tactics, it would be necessary to design features based on the rules of the game. The combination of rules would become the features used to learn new tactics.

The interpretability of FOL rules is not a given, and particularly long rules involving predicates with many variables cannot be considered "interpretable" by any measure. The use of an interpretability score which models how interpretable a human player would find a tactic can be used to optimize tactics for both divergence and interpretability. Further work is needed to identify tactic features that contribute to interpretability and design a computational model for calculating an interpretability score. It might also be the case that such a compact, generalizing hypothesis does not exist for a given set of examples.

I use a reference engine to provide the evaluation scores for a given move. If the reference engine and the target policy are not aligned, it would mean that moves are considered to be similar as per the divergence metric are perhaps not actually similar. In Section §4.5.4, I saw that Maia-1600 gave poorer divergence scores for the tactic moves as compared to Stockfish 14. It is possible that Stockfish 14, with its greater playing strength, sees less of a difference between the ground and tactic moves since they are both equally bad, or can be equally good with superhuman play. It is necessary to investigate further the effects of reference engine mis-alignment with the ground policy that I am trying to approximate with tactics.

4.5.6 Conclusion

I have presented the problem of learning chess tactics in the form of first-order logic rules from examples. I have presented a learning method that uses ILP to learn these tactics. I have evaluated this system and shown that the tactics learned cover a large portion of the test set. Using the metrics of divergence, I showed that they can approximate a human beginner better than a random baseline. I discussed limitations of our method and concluded that while it is promising, further work is required to learn tactics of acceptable playing strength.

4.6 Improving Tactic Learning using Precision and Recall

In an extension to the work in Section §4.5, I present an improvement to the strategy learning approach for chess using ILP. I present a new vocabulary for chess tactics that implements the full range of basic chess rules as FOL predicates. I formulate new constraints for the learning system based on precision and recall to help reduce the hypothesis space of chess tactics. Using an ablation study, I show that this approach is able to learn rules that are correct, provide sufficient coverage, and minimize divergence from expert recommendations. I discuss some limitations of our approach and conclude with directions for future work [4]. I thus answer RQ2c.

4.6.1 Methodology

In this section, I describe the additional constraints provided to the learning system based on precision and recall. I detail a proof of why the recall constraint will always improve recall of the learned hypotheses. I provide a description of our new predicate vocabulary for describing chess tactics.

4.6.1.1 Constraints Based on Precision and Recall

I propose new constraints to limit the size of the hypothesis space during generation based on precision and recall. I formally prove that a recall constraint will always improve recall of the learned hypotheses. The semantics of the $\preceq$ operator in our proofs is equivalent to *theory subsumption* as defined in Cropper and Morel (2021). $H_1 \preceq H_2$ indicates that all the examples entailed by hypothesis H_1 are also entailed by H_2.

Definition 7 (Precision constraint)**.** *A precision constraint prunes the specializations of a hypothesis if its precision on a set of examples is less than some pre-defined lower limit.*

A precision constraint cannot be shown to prune hypotheses with less precision than the current one. I add it to investigate its effect on the learned tactics.

Definition 8 (Recall constraint)**.** *A recall constraint prunes specializations of a hypothesis if its recall on a set of examples is less than some pre-defined lower limit.*

Theorem 1. *Given hypotheses $H_1, H_2 \in \mathbb{H}$ with $H_1 \preceq H_2$ and having recall values of r_1 and r_2 on a training set respectively, then $r_1 \leq r_2$.*

[4] The code release for this work is available at https://github.com/AbhijeetKrishnan/interpretable-chess-tactics

Proof.

$$\text{Recall} = \frac{tp}{tp + fn} \tag{4.7}$$

$$H_1 \preceq H_2 \tag{4.8}$$

$$r_1 = \frac{tp_1}{tp_1 + fn_1} \tag{4.9}$$

$$r_2 = \frac{tp_2}{tp_2 + fn_2} \tag{4.10}$$

By the definition of theory subsumption, I have

$$tp_1 \leq tp_2, \text{ and} \tag{4.11}$$

$$fp_1 \leq fp_2 \tag{4.12}$$

Since the number of positive examples is constant for the same population,

$$\therefore r_1 \leq r_2 \tag{4.13}$$

□

4.6.1.2 Predicate Vocabulary

I contribute a new predicate vocabulary for expressing chess tactics written in Prolog. Our new predicate vocabulary allows us to express tactics involving more situational rules of chess, such as en passant and pawn promotion. It improves the efficiency of tactic unification by removing the need for relying on a foreign predicate for checking legal moves. Figure 4.15 shows an interpretation of the *fork* expressed using our predicate vocabulary. Our predicate vocabulary consists of 25 predicates in total. These do not include the supporting rules required to construct these predicates since those are not allowed to be used in our tactic model. A listing of all predicates used in our vocabulary along with their descriptions can be found in Appendix C.

```
fork(Position,Move) ←
    legal_move(Position,Move),
    move(Move,_,To,_),
    make_move(Position,Move,NewPosition),
    can_capture(NewPosition,To,ForkSquare1),
    can_capture(NewPosition,To,ForkSquare2),
    different(ForkSquare1,ForkSquare2).
```

Figure 4.15: An interpretation of the *fork* tactic from the chess literature using our predicate vocabulary.

4.6.2 Evaluation

In this section, I describe the procedure used to evaluate the improvements made to our tactic learning system.

Hypothesis 2. *The average divergence of the tactics learned by the system including the precision and recall-based constraints and the new predicate vocabulary is lower than the average divergence of the tactics learned by the old system.*

I use the same experimental setup for learning tactics as in Section §4.5. I perform an ablation study to test the effect of the new predicate vocabulary, precision constraint and recall constraint on the tactics learned. I learn tactics for the following systems —

- T_{old}: the system from Krishnan and Martens (2022a)
- T_{none}: the new system described in this section, with the new predicate vocabulary but without the precision and recall constraints
- T_{prec}: the new system with the new predicate vocabulary and only the precision constraint (with a lower limit of 0.1)
- T_{rec}: the new system with the new predicate vocabulary and only the recall constraint (with a lower limit of 0.1)
- T_{new}: the new system (with a lower limit of precision and recall of 0.1)

To refute the hypothesis, I perform a one-sided Welch's t-test using the divergence values of the learned tactics.

4.6.2.1 Dataset

I use the same procedure as in Section §4.5.3.1 to source our train and test data. For positive examples, I select a single random position from each game and the move made in that position. For negative examples, I augment the dataset with the same position and the top-3 choices of Maia-1600 that do not include the ground truth move. Using this procedure, I generated a dataset of 1297 training examples, of which 488 were positive and the rest negative, and 100 testing examples.

4.6.2.2 Training

I use the same training hyperparameters as in Section §4.5.3.2.

4.6.2.3 Performance Metrics

I use the same performance metrics as in Section §4.5.3.3. I also compare them against the following baseline tactics —

- **random move tactic**: makes a random legal move in a given position, and is applicable to all positions
- **ground move tactic**: replicates the move made in the ground truth example, and is applicable to all positions
- **engine move tactic**: makes the best move suggested by an engine, and is applicable to all positions. I use two engines – Stockfish 14 and Maia-1600.

4.6.3 Results and Analysis

System	Size	Accuracy	Coverage	Maia-1600		Stockfish 14	
				Divergence	p	Divergence	p
T_{old}	837	0.12 ± 0.13	0.45 ± 0.16	174.48 ± 48.69	-	337.56 ± 57.98	-
T_{none}	141	0.34 ± 0.24	0.26 ± 0.35	103.28 ± 63.67	< 0.01	268.45 ± 131.67	< 0.01
T_{prec}	511	0.32 ± 0.23	0.21 ± 0.29	276.59 ± 147.13	1.0	276.59 ± 147.13	< 0.01
T_{rec}	387	0.10 ± 0.12	0.52 ± 0.41	143.27 ± 161.61	< 0.01	467.78 ± 138.69	1.0
T_{new}	284	0.11 ± 0.13	0.54 ± 0.41	143.57 ± 167.53	< 0.01	473.31 ± 151.90	1.0

Table 4.3: Summary statistics for the tactics learned by each system in the ablation study.

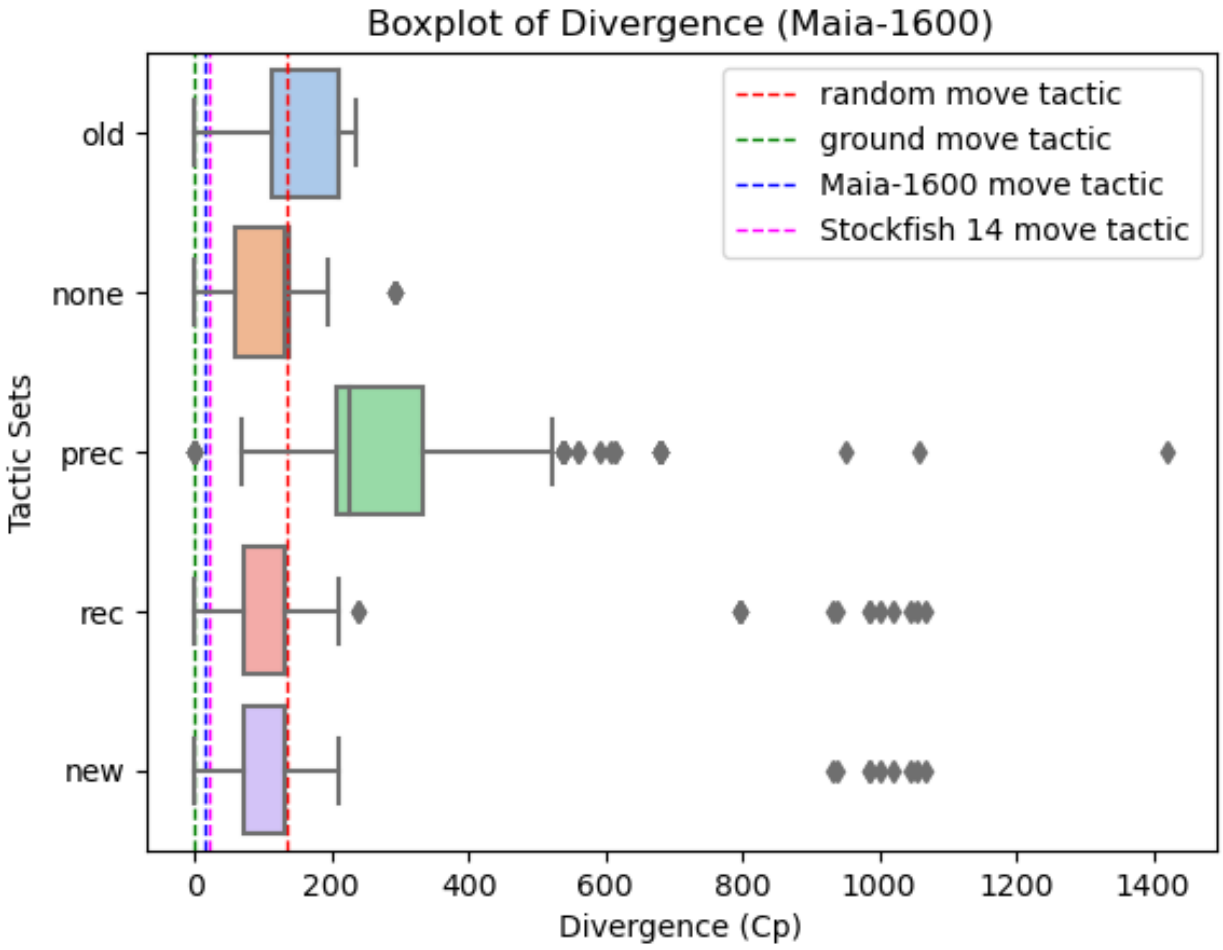

Figure 4.16: Boxplot of tactic divergence (evaluated using Maia-1600) for each system

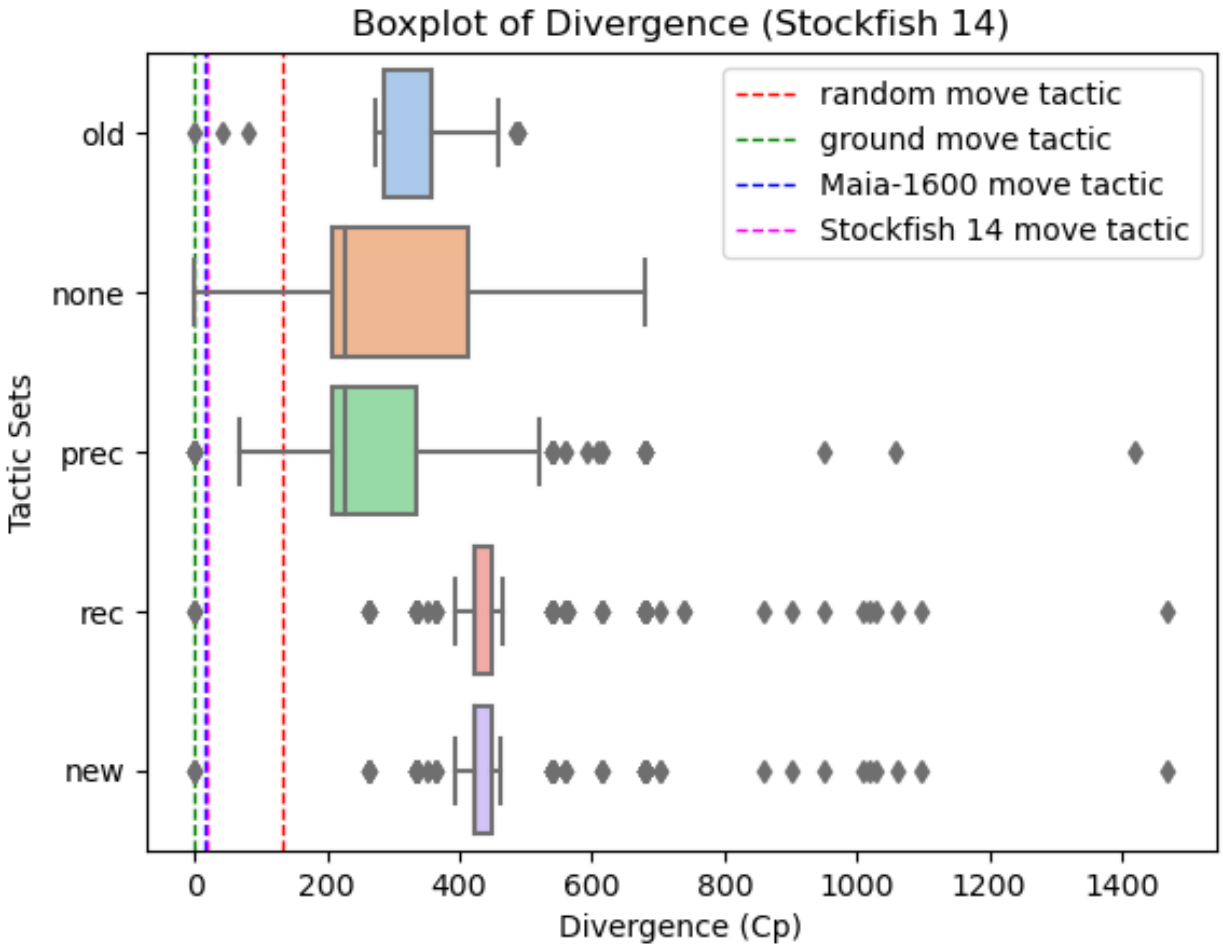

Figure 4.17: Boxplot of tactic divergence (evaluated using Stockfish 14) for each system

Based on the data obtained, I report boxplots for the tactics obtained from each of our systems. The summary statistics of the tactics learned by each system is shown in table 4.3.

Based on the results of the one-sided Welch's t-test shown in Table 4.3, I see that every system other than T_{prec} demonstrates a significant ($p < 0.01$) improvement in divergence over the old system when calculated using Maia-1600. When divergence is calculated using Stockfish 14, only T_{none} and T_{prec} demonstrate a significant improvement. I conclude that the new predicate vocabulary is effective at improving divergence universally, whereas the recall constraint is effective at improving divergence when calculated using a weaker engine.

4.6.3.1 Qualitative Analysis

I present two tactics learned by our system to further analyze qualitatively. These were learned by T_{new}.

4.6.3.1.1 Low Divergence The tactic in Figure 4.18 was chosen from those having the least divergence from ground as measured by Maia-1600. It can be interpreted as follows – make the move that allows you to castle in the subsequent position. Castling early is a common piece of advice given to beginner players.

```
f(Board,Move) ←
  legal_move(Board,Move),
  make_move(Board,Move,NewBoard),
  castling_move(NewBoard,NewMove).
```

Figure 4.18: A learned tactic with low divergence from ground as evaluated by Maia-1600.

4.6.3.1.2 Highest Accuracy The tactic in Figure Figure 4.19 was chosen from those having the highest accuracy. It can be interpreted as making a move that puts your opponent in check. This tactic does not have high accuracy when evaluated using the stronger Stockfish 14 engine, which reveals a tendency for lower-rated players to tend to make checking moves when available.

```
f(Position,From,To) ←
  legal_move(Board,Move),
  make_move(Board,Move,NewBoard),
  color(Color),
  in_check(NewBoard,Color,CheckingPiece).
```

Figure 4.19: A learned tactic with the highest accuracy.

4.6.4 Discussion and Future Work

The contribution of the new predicate vocabulary and precision/recall-based constraints helps our learning system generate better tactics that more closely approximate the ground moves made. Adding constraints based directly on divergence could help the system search for better tactics.

Our system is able to learn tactics that are evaluated favorably by a weak chess engine. Thus, the tactics seem to be effective at low levels of play. However, further work needs to be done to learn tactics that are evaluated favorably by a stronger chess engine. This would help these tactics be effective even at high levels of play.

4.6.4.1 Extensibility to Other Domains

The contribution of a new predicate vocabulary resulted in measurable improvement to the learned tactics for chess, due to the expressiveness of the new predicates in describing more rules of chess when specifying tactics. This suggests that an automated translation of a game's rules to a predicate vocabulary could be beneficial for learning FOL strategies in other domains.

4.6.5 Conclusion

I have presented the problem of learning chess tactics in the form of first-order logic rules from examples. I have introduced two significant improvements to a previous tactic-learning system in the form of a) a new predicate vocabulary, and b) constraints based on precision and recall. I have presented a novel evaluation method for these improvements using an ablation study to measure their effects on the divergence of the learned tactics from a target policy of beginner human players. I showed that the new predicate vocabulary and recall-based constraint contributed significantly to reducing the divergence, whereas the precision-based constraint did not. I discussed limitations of the improvements made and suggested future

improvements in the form of new divergence-based constraints.

4.7 Limitations and Future Work

The FOL rules-based strategy model presented in this chapter has several limitations. First, there is the issue of the high cost of *knowledge engineering*. Learning FOL-based strategies requires the background knowledge to be manually authored for each domain. This is a significant bottleneck in the development of the strategy model. It is conceivable that closely related domains may share some common knowledge. However, the relying on the same knowledge base for multiple domains may not be feasible due to the sensitivity of the performance of the learned strategies on the background knowledge (see Section §4.6.3). This suggests the possibility of an automated process to translate the description of a MDP into a set of FOL rules. This would be a significant step towards making the use of FOL rules as a strategy model practical. It would also allow learning algorithms to be benchmarked across a wider range of domains.

Second, the FOL rules-based strategy model did not have promising results in terms of performance. From Sections §4.5.4 and §4.6.3, we see that they were only able to outperform a random baseline, and were not competitive with modern chess engines, even a NN-based engine tuned to resemble a human beginner in playing strength. This suggests that the FOL rules-based strategy model may not be suitable for modeling strategies in complex domains like chess. Another possibility is the need for an even more expressive background knowledge to model strategies in chess. However, the improvements to the ILP-based learning algorithm shown in Section §4.6.3 show that there is room for improvement in the learning process itself, which may lead to better strategies.

Finally, there is no quantitative evidence that the FOL-rules based strategy model is interpretable. While small rules might be readable, and can be understood with some familiarity with FOL, larger rules with many predicates and variables may be difficult to interpret. Moreover, there has not been a user study of sufficient power to determine that the learned strategies have a statistically significant effect on improving human chess play. This is a useful area for future work that has implications for the use of rule-based models in XAI more broadly.

CHAPTER

5

STRATEGIES AS PROGRAMS

In this chapter, I describe how strategies can be modeled as programmatic policies. I describe how programmatic policies can be used to model strategies, and how they can be learned from data using techniques from program synthesis. I also show how the problem of learning programmatic policies may be modeled as a RL problem, allowing additional techniques to be brought to bear, and answering '3a. I then apply the DT model to the problem of learning strategies, thus answering '3b.

5.1 Programs as Policies

Programmatic policy synthesis is the problem of approximating a target policy using a program, as specified by some subsection. I cast this as a RL problem by modeling the program synthesis environment as a MDP. Formally, given a DSL $G = (V, \Sigma, R, S)$ (Sipser 1996) which defines the space of possible programmatic policies, and a MDP $\mathcal{M}_{\text{exec}}$ in which a programmatic policy σ can execute and obtain reward, I define the programmatic policy synthesis task as that of finding an optimal policy in the MDP $\mathcal{M}_{\text{syn}}$, where:

- $\mathcal{S}_{\text{syn}} \doteq \{w \in (V \cup \Sigma)^* \mid S \overset{*}{\Rightarrow} w\}$ and is the current sequence of terminals and non-terminals in the partially expanded program $(w_0, w_1, \cdots, w_i, \cdots)$.

- $\mathcal{A}_{\text{syn}} \doteq R \times V \times \mathbb{N}$ and is the space of all actions consisting of applying a rule r to a non-terminal symbol v located at index i in the current state.
- $\mathcal{P}_{\text{syn}}$ is the transition function which represents the deterministic transition to a state where the production rule has been applied to the selected non-terminal symbol if such an action was valid. If not, the state remains unchanged.
- $d_{0,\text{syn}}(S) = 1$
- $r_{\text{syn}} \doteq \begin{cases} r_{\text{exec}}, & \text{if current state } s \in \mathcal{S}_{\text{syn}} \text{ is terminal} \\ 0, & \text{otherwise} \end{cases}$
- $\gamma_{\text{syn}} = 1$

The goal of the programmatic policy synthesis task is to synthesize a program ρ that maximizes the objective function $J(\rho)$ which is defined similarly as in Equation 5.1.

5.1.1 Programmatic Policy Model

The strategy model for a programmatic policy is the DSL specified as a CFG $G = (V, \Sigma, R, S)$, where V is the set of non-terminal symbols, Σ is the set of terminal symbols, R is the set of production rules, and S is the start symbol. The *language* of G is the set of all possible strings that can be generated by G starting from S. Any string in the language of G is a valid program in the DSL, and represents a strategy σ that accepts as input a state s and outputs an action a. This requires an associated interpreter and simulator in which the script can be executed.

The DSL for tasks in the Karel programming environment (described in Section §**??**) is shown in Figure 5.1. Its associated interpreter and simulator are contributed by Trivedi et al. (2021). Another example of a DSL is the one used for the Synthesis of Programmatic Strategies (SynProS) competition (Moraes 2021), which describes scripts for the MicroRTS game (Ontañón et al. 2018). Both these DSLs are domain-specific, and are designed to capture the structure of the problem domain. Verma, Murali, et al. (2018) describes a general-purpose DSL for programmatic policies, but which can only represent if-else conditions, without loops, function calls or variables. Similarly, Qiu and Zhu (2022) also describes a general-purpose DSL for programmatic policies, but which also cannot represent higher-order constructs.

5.1.2 Learning Programmatic Policies

Programmatic scripts have received considerable attention as a surrogate model for explaining black-box policies. They have proven attractive due to their readability, and the ability to

⟨*program*⟩ ::= 'DEF' 'run' 'm(' ⟨*stmt*⟩ 'm)'

⟨*stmt*⟩ ::= 'REPEAT' ⟨*cste*⟩ 'r(' ⟨*stmt*⟩ 'r)'
| ⟨*stmt*⟩ ⟨*stmt*⟩
| ⟨*action*⟩
| 'IF' 'c(' ⟨*cond*⟩ 'c)' 'i('⟨*stmt*⟩ 'i)'
| 'IFELSE' 'c(' ⟨*cond*⟩ 'c)' 'i(' ⟨*stmt*⟩ 'i)' 'e(' ⟨*stmt*⟩ 'e)'
| 'WHILE' 'c(' ⟨*cond*⟩ 'c)' 'w(' ⟨*stmt*⟩ 'w)'

⟨*cond*⟩ ::= 'not' 'c(' ⟨*cond_without_not*⟩ 'c)'
| ⟨*cond_without_not*⟩

⟨*cond_without_not*⟩ ::= 'frontIsClear'
| 'leftIsClear'
| 'rightIsClear'
| 'markersPresent'
| 'noMarkersPresent'

⟨*action*⟩ ::= 'move'
| 'turnLeft'
| 'turnRight'
| 'putMarker'
| 'pickMarker'

⟨*cste*⟩ ::= 'R='⟨*int*⟩

Figure 5.1: The domain-specific language (DSL) for constructing programs in the Karel environment.

draw on existing research in *program synthesis* to learn them. Program synthesis is the task of automatically finding a program in the underlying programming language that satisfies the user intent expressed in some form of specification (Gulwani et al. 2017). Examples of specifications include formal rules (Manna and Waldinger 1980), input/output pairs (Neelakantan et al. 2016; Devlin et al. 2017; Gaunt et al. 2017; Shin et al. 2018; Bunel et al. 2018; Lázaro-Gredilla et al. 2019; X. Chen et al. 2019; Yang et al. 2021), demonstrations (Sun et al. 2018; Xu et al. 2018; Burke et al. 2019), natural language instructions/prompts (Liang et al. 2023) and MDP rewards (Li, Gimeno, et al. 2020; Trivedi et al. 2021; Qiu and Zhu 2022). Neural program synthesis focuses specifically on applying statistical learning methods based on NNs to the problem of program synthesis.

Many work in XRL attempt to learn programmatic policies. Bastani et al. (2018) learns a decision tree as a programmatic policy for continuous environments by imitating a target neural policy. However, decision trees are incapable of representing repeating behaviors on their own. Inala et al. (2020) learns programmatic policies as finite state machines by imitating a teacher policy, although finite state machine complexity can scale quadratically with the number of states, making them difficult to scale to more complex behaviors. Another line of work instead synthesizes programs structured in DSLs, allowing humans to design *tokens* (e.g., conditions and operations) and *control flows* (e.g.,while loops, if statements, reusable functions) to induce desired behaviors and can produce human interpretable programs. Verma, Murali, et al. (2018) and Verma, Le, et al. (2019) distill NN policies into programmatic policies. Yet, the initial programs are constrained to a set of predefined program templates. This significantly limits the scope of synthesizable programs and requires designing such templates for each task. A number of works synthesize programs in a more general-purpose DSL for a game-based MDP (Mariño, Moraes, et al. 2021; Mariño and Toledo 2022; Medeiros et al. 2022) using evolutionary algorithms.

5.1.2.1 Offline Reinforcement Learning

I borrow the definition of a MDP from Levine et al. (2020) as a tuple $\mathcal{M} \doteq (\mathcal{S}, \mathcal{A}, \mathcal{P}, d_0, r, \gamma)$, consisting of states $s_t \in \mathcal{S}$ that may be discrete or continuous, actions $a_t \in \mathcal{A}(s)$ which can also be discrete or continuous, transition dynamics $\mathcal{P}(s'|s,a) \doteq \Pr\{S_t = s'|S_{t-1} = s, A_{t-1} = a\}$ that describe the dynamics of the system, initial state distribution $d_0(s_0)$, reward function $r \in \mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ at timestep t, and a scalar discount factor $\gamma \in [0,1)$. A policy π describes a process for selecting actions given states, thus inducing a trajectory $\tau \doteq (s_0, a_0, \cdots, s_H, a_H)$, with the trajectory distribution $p_\pi(\tau) = \prod_{t=0}^{H} \pi(a_t|s_t)\mathcal{P}(s_{t+1}|s_t, a_t)$. The reinforcement learning objective, $J(\pi)$, can then be written as an expectation under this trajectory distribution:

$$J(\pi) \doteq \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t) \right] \tag{5.1}$$

In offline RL, instead of the same policy being used to explore the environment while also being optimized, the environment is explored using a separate, fixed *behavior* policy b, and its experience is used to optimize the target policy π (Sutton and Barto 2018).

5.2 Learning Programmatic Policies using Decision Transformers

In this section, I present a novel method for learning programmatic strategies using transformers that is competitive with state-of-the-art methods while being more sample efficient. I first describe how problem of synthesizing programmatic strategies can be modeled as a RL problem. I then present our method, which applies the DT model to discrete environments. I evaluate our method using tasks in a grid-based programming environment and show that the strategies learned by our method are competitive with a state-of-the-art programmatic policy synthesis algorithm, while being much more sample efficient (Krishnan, Potts, et al. 2024).

5.2.1 Background

5.2.1.1 Transformers

Transformers were proposed by Vaswani et al. (2017) as an architecture to efficiently model sequences. They consist of stacked self-attention layers with residual connections. Each self-attention layer receives n embeddings $\{x_i\}_{i=1}^n$ corresponding to unique input tokens, and outputs n embeddings $\{z_i\}_{i=1}^n$, preserving the input dimensions. The i-th token is mapped via linear transformations to a key k_i, query q_i, and value v_i. The i-th output of the self-attention layer is given by weighting the values v_j by the normalized dot product between the query q_i and other keys k_j:

$$z_i = \sum_{j=1}^{n} \texttt{softmax}\left(\{ \langle q_i, k_{j'} \rangle \}_{j'=1}^{n} \right)_j \cdot v_j \tag{5.2}$$

This allows the layer to assign "credit" by implicitly forming state-return associations via similarity of the query and key vectors (maximizing the dot product). Decision transformers use the Generative Pre-trained Transformer (GPT) architecture (Radford et al. 2018), which modifies the transformer model with a causal self-attention mask to enable autoregressive

generation, replacing the summation/softmax over the n tokens with only the previous tokens in the sequence ($j \in [1, i]$).

The transformer architecture (Vaswani et al. 2017) has demonstrated tremendous success in modelling large-scale distributions of semantic concepts, including zero-shot generalization in language (Brown et al. 2020) and out-of-distribution image generation (Ramesh et al. 2021). The work by L. Chen et al. (2021) examines their application to sequential decision-making problems, where they show that their proposed DT model is competitive or better than other task-specific algorithms with minimal modification from standard language modeling architectures.

5.2.1.2 Decision Transformers

Decision transformer (DT) were proposed by L. Chen et al. (2021) as an architecture to apply the sequence-modeling capabilities of transformers (Vaswani et al. 2017) to sequential decision problems. Trajectories τ are represented as:

$$\tau = \left(\widehat{R}_1, s_1, a_1, \widehat{R}_2, s_2, a_2, \cdots, \widehat{R}_T, s_T, a_T\right) \tag{5.3}$$

where rewards r_t are modeled as the returns-to-go $\widehat{R}_t = \sum_{t'=t}^{T} r_{t'}$. States, actions and returns are embedded using separate, fully-connected layers. Additionally, an embedding is learned for each timestep and added to each token. This is different from the standard positional embedding used by transformers, as one timestep corresponds to three tokens. The tokens are then processed by a GPT (Radford et al. 2018) model, which predicts future actions via autoregressive modeling.

5.2.2 Methodology

In this section, I describe the modifications made to the DT architecture to solve the programmatic policy synthesis task.

The original DT architecture is designed for environments with continuous action spaces. The programmatic policy synthesis MDP (described in Section §5.1) has discrete actions, and hence I make the following modifications to the architecture:

- **Action Masking**: To ensure that the model does not sample invalid actions, I mask out the invalid actions by following the approach described in Huang and Ontañón (2022). Informally, given the unnormalized logits representing action predictions, I mask out invalid actions by setting their logits to $-\infty$ and apply a softmax to obtain a probability distribution.

- **Cross-entropy loss**: The original architecture used a simple L2 loss since actions were continuous. For discrete actions, I use a cross-entropy loss as is usual with classification tasks (Bishop 2006).
- **Sampling from action distribution**: The original architecture uses a regression layer to predict the continuous-valued action. Since I am dealing with discrete actions, I sample from the predicted action distribution to obtain the action.

Additionally, to better represent the sequential nature of states in the MDP, I embed the state using a sequential model, specifically a Gated Recurrent Unit (GRU) (Cho et al. 2014). This follows from existing approaches to sentence embedding for information retrieval (Palangi et al. 2016; Kiros et al. 2015; Le and Mikolov 2014). I empirically find that this improves the model's performance when compared to embedding the state directly [1].

5.2.3 Evaluation

In this section, I investigate if the DT architecture can perform well on the programmatic policy synthesis task when compared to existing approaches. I use the same environment and tasks as in Trivedi et al. (2021), which I briefly describe in Section §5.2.3.1.

5.2.3.1 Karel domain

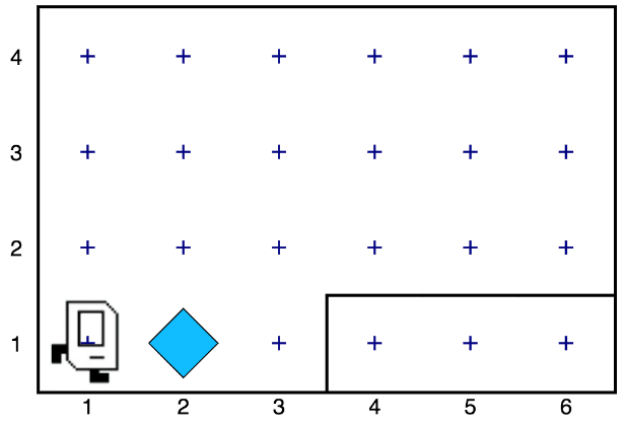

Figure 5.2: A sample Karel world of size 4×6. The blue diamond represents a marker. The agent cannot travel through walls.

[1] The code release for this paper can be found at https://github.com/AbhijeetKrishnan/decision-transformer .

Karel is a simple programming language designed for teaching programming to beginners (Pattis 1994). Programs written in Karel are used to control an agent that can move around a grid world and interact with markers in the world. The agent has actions for moving and interacting with markers, and perceptions for detecting obstacles and markers. Trivedi et al. (2021) introduced six specific tasks for the Karel domain, which I use in our experiments. The initial configuration for these tasks is randomly sampled for each episode reset.

I implement the MDP $\mathcal{M}_{\text{syn}}$ for the Karel task environment using the Gymnasium library (Towers et al. 2023) to allow interoperability with other RL algorithms. Its source code has been published online under an open-source license[2].

5.2.3.2 Dataset

For each task, I obtained trajectories using the randomly generated training data contributed by Trivedi et al. (2021). For each program in the dataset, I parsed it to obtain a sequence of state-action pairs, which I then executed in the Karel environment to obtain a trajectory. I plot a histogram of their returns using 100 bins in Figure 5.3, with some additional statistics about the dataset in Table 5.1.

Transitions	870,782
Programs	50,000
Program Length	19.30 ± 5.96 ($min = 7, max = 50$)
Episode Length	17.42 ± 4.33 ($min = 5, max = 30$)

Table 5.1: Statistics for the LEAPS program dataset.

5.2.3.3 Training

I train the DT similarly to L. Chen et al. (2021). I tune the hyperparameters for each task using the Optuna framework (Akiba et al. 2019). The tuned list of hyperparameters for each task are detailed in Table 5.2. I sample trajectories from a batch weighted by their length. During training, I calculate the reward of a program in the task environment as its average reward over 10 random initial states given a fixed seed. To obtain a program from the model for a task T, I first sample 64 programs from the model trained on T and use the program with the best return. I parameterize the top-k% of the programs to train over as a tunable hyperparameter.

[2]https://github.com/AbhijeetKrishnan/grammar-synthesis

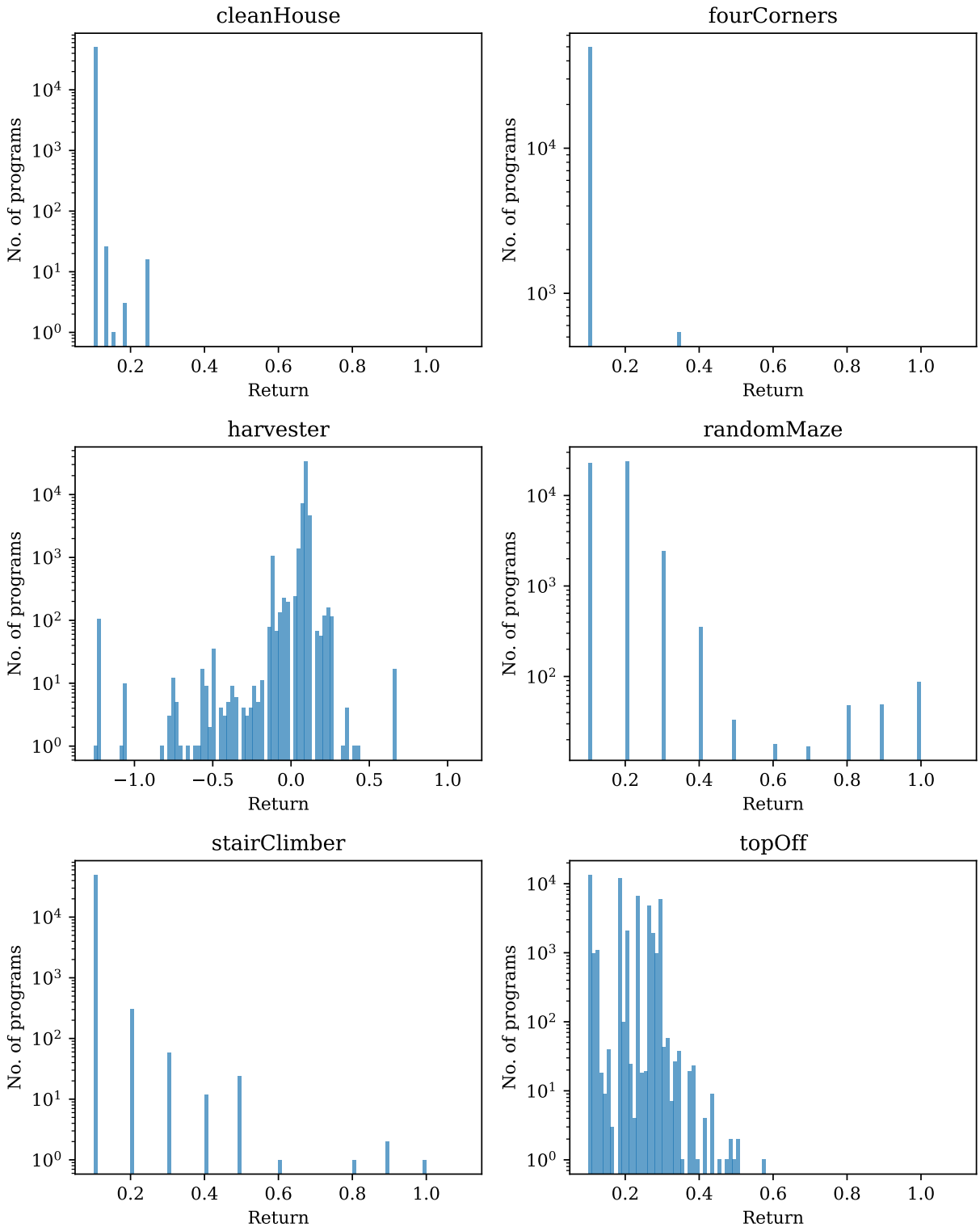

Figure 5.3: A histogram of the returns of the trajectories in the LEAPS dataset.

Hyperparameter	**Value**
Embedding dimension	128
Context length (K)	50
Activation function	ReLU
Batch size	64
No. of attention layers	3
No. of attention heads	2
Dropout	0.05
Learning rate	1.0×10^{-3}
Weight decay	5.0×10^{-4}
Warm-up steps	200
No. of training steps	200
GRU hidden size	512
GRU layers	1
GRU dropout	0.05
Return-to-go conditioning	2.2
Grad norm clip	0.25
No. of eval. episodes	64
Top-k%	0.01
Max iterations	10

Table 5.2: Hyperparameters used for training the Decision Transformer for the Karel programmatic policy synthesis task.

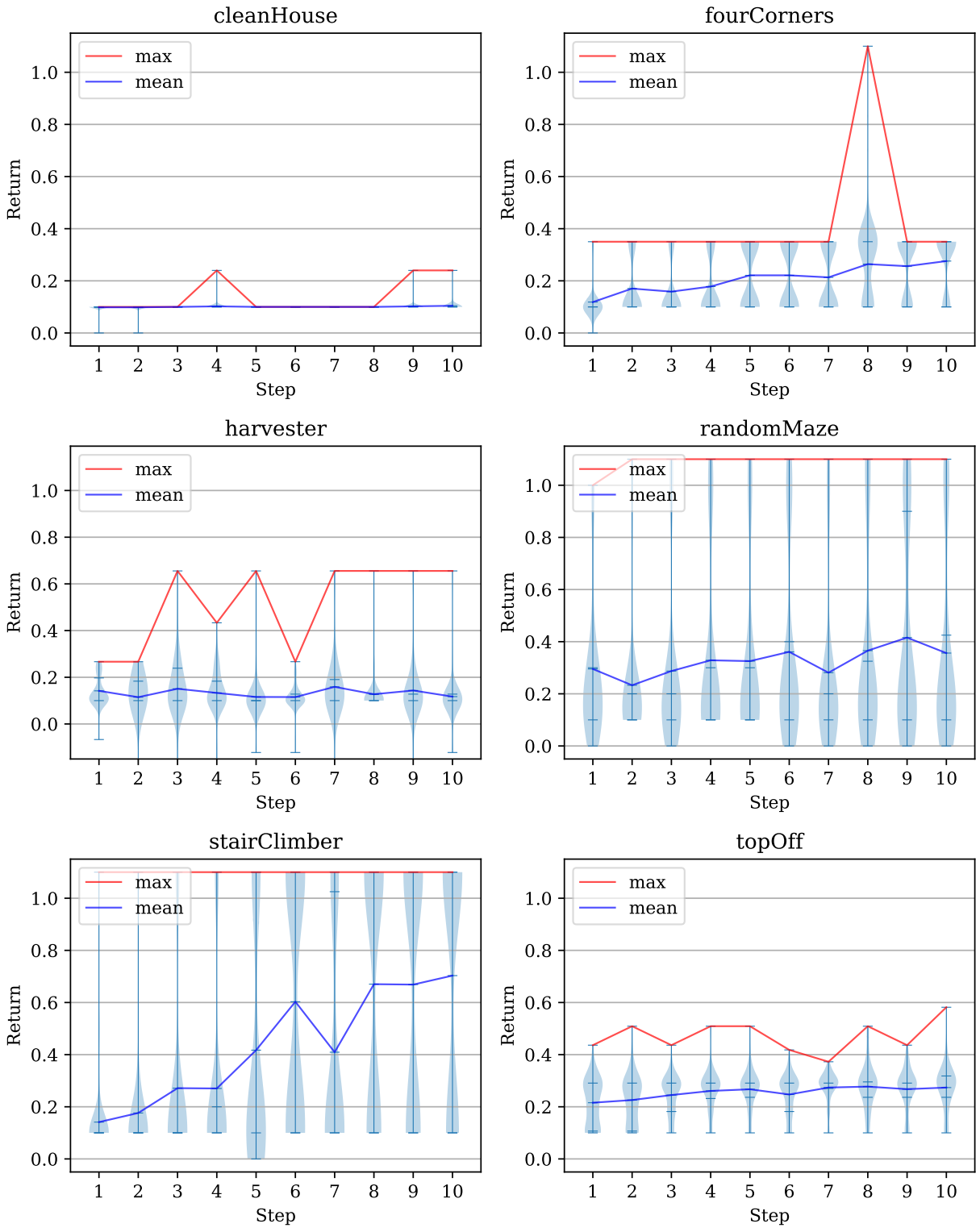

Figure 5.4: Return of the DT model on each task during training.

5.2.3.4 Results

To reproduce the performance of LEAPS on each Karel task, I run the Cross-Entropy Method (CEM) search phase using five different randomly chosen seeds to obtain five different programs. I also record the unique programs explored for each task during the search phase. I then evaluate each program in the task environment and report the mean and standard deviation of the returns for 100 different initial states. For the DT model, I use the single best program found during the final training iteration and evaluate it similarly over the same 100 different initial states.

Task	**Mean Return**		**Unique Programs**	
	LEAPS	DT	LEAPS	DT
cleanHouse	0.16 (0.13)	0.23	3627	59
fourCorners	0.35 (0.00)	0.35	9872	55
harvester	0.61 (0.21)	0.66	11708	28
randomMaze	0.97 (0.04)	1.0	295	63
stairClimber	0.74 (0.49)	1.1	298	49
topOff	0.80 (0.11)	0.66	30278	63

Table 5.3: Mean return of the best program found and number of unique programs explored by LEAPS and the Decision Transformer model on each Karel task. The standard deviation is reported in parentheses.

5.2.4 Discussion

I modify the DT architecture to handle environments with discrete action spaces and apply it to the task of programmatic policy synthesis in the Karel domain. I train our model using trajectories sampled from a random policy. Our experiments show that the model is competitive with LEAPS while requiring significantly less training data and search over program space.

5.2.4.1 Training Requirements

The DT model requires less training data and searches over a smaller program space than LEAPS. LEAPS trains on 35,000 programs while the DT model requires just 500 programs to achieve the same performance. Table 5.3 shows that LEAPS searches over a much larger number of programs for each task, compared to the 64 programs that the DT searches over.

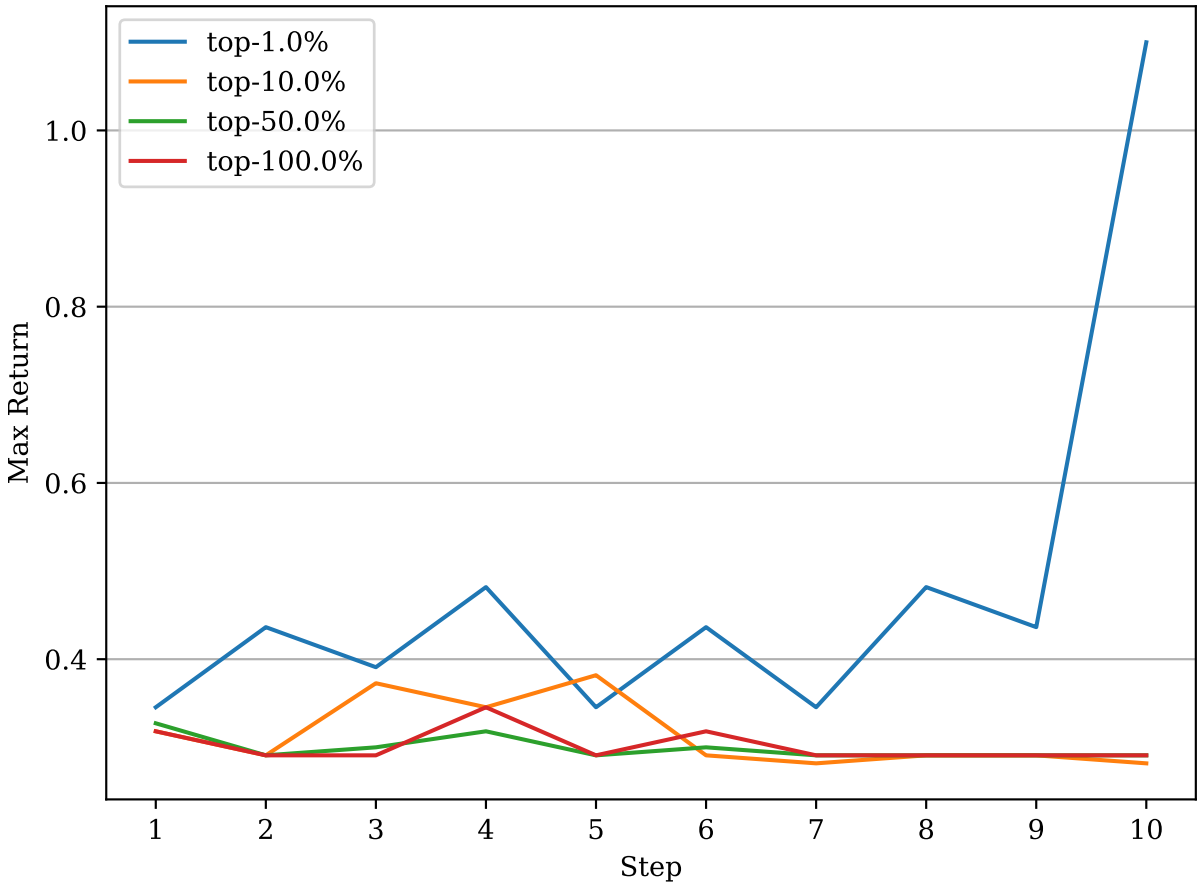

Figure 5.5: Return of the DT model when trained on the top-k% of the dataset.

5.2.4.2 Sensitivity to Training Data Quality

To evaluate the sensitivity of the DT model to the quality of the training data, I train the model on the top-k% of trajectories by return on the `topOff` task. I sample 64 programs from the model at each time step and plot the maximum return achieved by a program. Figure 5.5 shows that the model is highly sensitive to the quality of the training dataset.

5.2.4.3 Hardware Requirements

The DT model for each Karel task was trained on a compute cluster node with an AMD Epyc Rome series CPU, 128GB RAM, and a single NVIDIA GeForce RTX 2060 Super GPU. Each model for a task took approximately 30 minutes to train.

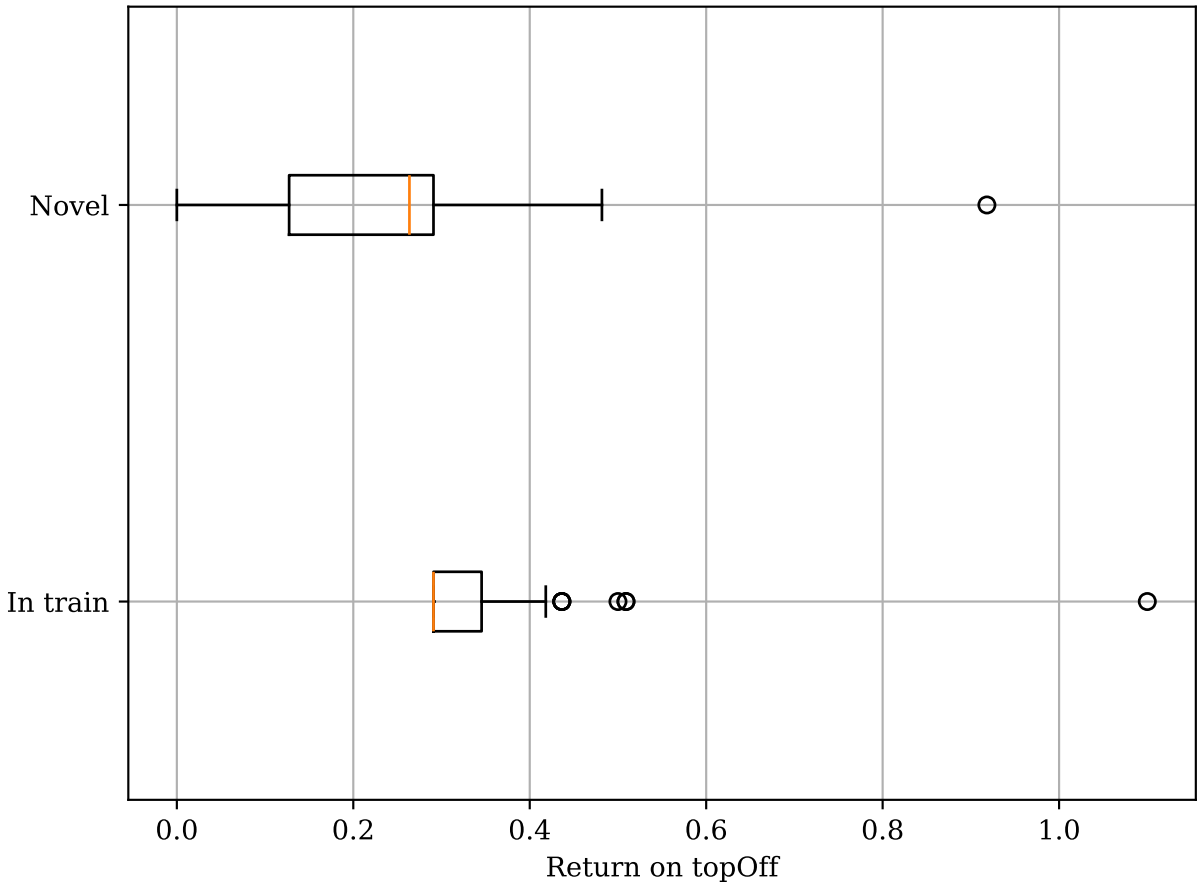

Figure 5.6: Box plot of returns for programs sampled from the DT model.

5.2.4.4 Novelty of Generated Programs

To evaluate the similarity of programs generated by the DT model to those in the training dataset, I sample 998 programs from a model trained on the `topOff` task. After de-duplication, I identify the set of novel synthesized programs (P_{novel}) that are not present in the training dataset and those that are (P_{dup}), and plot the distribution of returns for both as a boxplot in Figure 5.6. To test the hypothesis that the mean returns of P_{nov} is not less than that of P_{dup}, I use a one-sided Welch's t-test due to them having unequal sample sizes ($|P_{\text{nov}}| = 643$, $|P_{\text{dup}}| = 115$) and variances ($\sigma^2_{nov} = 8.99 \times 10^{-3}$, $\sigma^2_{dup} = 7.73 \times 10^{-3}$). I find that the mean return of the programs in P_{dup} is significantly higher ($p < 0.01$) than the mean return of the novel programs in P_{nov}. This suggests that while the DT is capable of producing novel programs, they tend to have lower return than the dataset the model was trained on.

5.2.5 Conclusion

I applied the decision transformer (DT) model to the task of programmatic policy synthesis solely from MDP reward signals. I introduced modifications to make the model compatible with discrete action environments. Using the Karel domain as a testbed, I determined that the model is competitive with a state-of-the-art programmatic policy learning method while being more sample-efficient.

In future work, it would be useful to benchmark the DT, and other programmatic policy learning methods, on a wider range of game-based environments. It would also help to investigate the interpretability of the learned strategies, and to measure the extent to which they are able to measurably improve a player's performance.

CHAPTER

6

CONCLUSION

In this dissertation, I have investigated approaches to the problem of Interpretable Strategy Synthesis (ISS) for multiple game domains in order to better understand how to design effective strategy models for games, and which algorithms are most successful to learn them. Recent efforts in synthesizing programmatic strategies for games (Mariño and Toledo 2022), as well as learning more general programmatic policies (Qiu and Zhu 2022) are useful steps in this direction, but do not present a general framework with which other strategy models and learning algorithms can be applied to other games. I present such a framework in Chapter 3, with a formal definition of the problem of ISS and a description of its components.

I then present two possibilities of modeling a game strategy using 1) first-order logic rules and 2) programmatic policies. In Chapter 4, I describe how strategies can be modeled using these representations, and how they can be learned. I present a case study for learning chess strategies as first-order logic rules in Section §4.3, describing how a chess strategy can be modeled as a first-order logic rule (Section §4.3.2) and how it can be learned using Inductive Logic Programming (ILP) (Section §4.5). I introduce two metrics - coverage and divergence, to evaluate the learned strategies (Section §4.3.3), and show that the learned strategies are better at approximating a human beginner than a random baseline. I improve the learning method using novel constraints and a richer background knowledge of chess concepts (Section §4.6).

In Chapter 5, I describe how strategies may be modeled as programmatic policies, and how

they can be learned. I present a case study for learning programmatic strategies via offline RL using the DT model, for a toy programming environment (Section §5.2). I demonstrate that the learned strategies are competitive with existing state-of-the-art while being much more sample-efficient.

For both case studies, I contribute open-source software to enable reproduction and reuse of the work. I also describe limitations of these strategy models and suggest avenues of future work to address them.

The knowledge generated from this work will inform existing research and practices in esports analytics that try to generate insights from player data, or provide personalized coaching to players. It will also help the player base uncover new strategies which can be easily communicated and shared. It contributes to the field of XRL by providing new methods for interpreting policies learned by RL agents.

BIBLIOGRAPHY

Akiba, Takuya et al. (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework." In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Averbakh, Yuri (2012). *A history of chess: from Chaturanga to the present day*. SCB Distributors.

Bastani, Osbert, Yewen Pu, and Armando Solar-Lezama (2018). "Verifiable Reinforcement Learning via Policy Extraction." In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/hash/e6d8545daa42d5ced125a4bf747b3688-Abstract.html.

Bellman, Richard (1958). "Dynamic programming." In: *Princeton University Press, Princeton, NJ, USA* 1, pp. 3–25.

Bijl, Pieter and AP Tiet (July 2021). "Exploring modern chess engine architectures." PhD thesis. Victoria University, Melbourne.

Bioware and Obsidian Entertainment (June 2002). *Neverwinter Nights*. [CD-ROM].

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387310738.

Biswas, Pradipta and Mark Springett (2018). "User Modeling." In: *The Wiley Handbook of Human Computer Interaction Volume*, p. 143.

Boros, Endre et al. (2012). "On Nash equilibria and improvement cycles in pure positional strategies for Chess-like and Backgammon-like n-person games." In: *Discrete Mathematics* 312.4, pp. 772–788.

Boyan, Andy, Rory McGloin, and Joe A. Wasserman (2018). "Model matching theory: a framework for examining the alignment between game mechanics and mental models." In: *Media and Communication* 6.2, pp. 126–136. ISSN: 2183-2439. DOI: https://doi.org/10.17645/mac.v6i2.1326.

Boyan, Andy and John L. Sherry (2011). "The Challenge in Creating Games for Education: Aligning Mental Models With Game Models." en. In: *Child Development Perspectives* 5.2, pp. 82–87. ISSN: 1750-8606. DOI: 10.1111/j.1750-8606.2011.00160.x.

Brown, Tom et al. (2020). "Language Models are Few-Shot Learners." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc.,

pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Bunel, Rudy et al. (May 2018). *Leveraging Grammar and Reinforcement Learning for Neural Program Synthesis*. arXiv:1805.04276 [cs, stat]. DOI: 10.48550/arXiv.1805.04276. URL: http://arxiv.org/abs/1805.04276 (visited on 06/07/2023).

Buntine, Wray (1988). "Generalized subsumption and its applications to induction and redundancy." In: *Artificial Intelligence* 36.2, pp. 149–176. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(88)90001-X. URL: https://www.sciencedirect.com/science/article/pii/000437028890001X.

Burke, Michael, Svetlin Valentinov Penkov, and Subramanian Ramamoorthy (June 2019). "From Explanation to Synthesis: Compositional Program Induction for Learning from Demonstration." In: *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation. DOI: 10.15607/rss.2019.xv.015. URL: https://doi.org/10.15607%2Frss.2019.xv.015.

Butler, Eric, Emina Torlak, and Zoran Popović (2017). "Synthesizing Interpretable Strategies for Solving Puzzle Games." In: *Proceedings of the 12th International Conference on the Foundations of Digital Games*. FDG '17. Hyannis, Massachusetts: Association for Computing Machinery. ISBN: 9781450353199. DOI: 10.1145/3102071.3102084. URL: https://doi.org/10.1145/3102071.3102084.

Canaan, Rodrigo et al. (2018). "Evolving Agents for the Hanabi 2018 CIG Competition." In: *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8. DOI: 10.1109/CIG.2018.8490449.

Chen, Lili et al. (2021). "Decision Transformer: Reinforcement Learning via Sequence Modeling." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 15084–15097. URL: https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f063-Abstract.html (visited on 06/08/2023).

Chen, Xinyun, Chang Liu, and Dawn Song (2019). "Execution-Guided Neural Program Synthesis." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=H1gfOiAqYm.

Cho, Ch'i-hun (1997). *GO: A complete introduction to the game*. Kiseido.

Cho, Kyunghyun et al. (Oct. 2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation." In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Ed. by Alessandro Moschitti, Bo Pang,

and Walter Daelemans. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: https://aclanthology.org/D14-1179.

Chrysafiadi, Konstantina and Maria Virvou (2013). "Student modeling approaches: A literature review for the last decade." In: *Expert Systems with Applications* 40.11, pp. 4715–4729. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2013.02.007. URL: http://www.sciencedirect.com/science/article/pii/S095741741300122X.

Claude, E. Shannon (1950). "Programming a Computer for Playing Chess." In: *Philosophical Magazine, Ser* 7.41, p. 314.

Coppens, Youri et al. (2019). "Distilling deep reinforcement learning policies in soft decision trees." In: *Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence*, pp. 1–6.

Cropper, Andrew and Sebastijan Dumančić (June 2022). "Inductive Logic Programming At 30: A New Introduction." en. In: *Journal of Artificial Intelligence Research* 74, pp. 765–850. ISSN: 1076-9757. DOI: 10.1613/jair.1.13507.

Cropper, Andrew and Rolf Morel (2021). "Learning programs by learning from failures." In: *Machine Learning* 110.4, pp. 801–856.

Cropper, Andrew and Sophie Tourret (2020). "Logical reduction of metarules." In: *Machine Learning* 109.7, pp. 1323–1369.

Defer, Aurélien (Feb. 2023). "Video game coaching, a booming business." en. In: *Le Monde.fr*. URL: https://www.lemonde.fr/en/international/article/2023/02/05/video-game-coaching-a-booming-business_6014469_4.html.

Deutsch, Max (Dec. 2017). *My month-long quest to become a chess master from scratch*. URL: https://medium.com/@maxdeutsch/my-month-long-quest-to-become-a-chess-master-from-scratch-51ff8003d3f2 (visited on 03/21/2023).

Devlin, Jacob et al. (July 2017). "RobustFill: Neural Program Learning under Noisy I/O." In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 990–998. URL: https://proceedings.mlr.press/v70/devlin17a.html.

Doshi-Velez, Finale and Been Kim (2017). *Towards A Rigorous Science of Interpretable Machine Learning*. arXiv: 1702.08608 [stat.ML].

Ericsson, K. Anders, Ralf T. Krampe, and Clemens Tesch-Römer (1993). “The role of deliberate practice in the acquisition of expert performance.” In: *Psychological Review* 100.3, pp. 363–406. ISSN: 1939-1471(Electronic),0033-295X(Print). DOI: 10.1037/0033-295X.100.3.363.

Freitas, João Marcos de, Felipe Rafael de Souza, and Heder S. Bernardino (2018). “Evolving Controllers for Mario AI Using Grammar-based Genetic Programming.” In: *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. DOI: 10.1109/CEC.2018.8477698.

Gamers Net, Inc. (2023). *Mobalytics - The All-in-One Companion for Every Gamer*. en-US. URL: https://mobalytics.gg/ (visited on 03/21/2023).

Gaunt, Alexander L. et al. (July 2017). “Differentiable Programs with Neural Libraries.” In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 1213–1222. URL: https://proceedings.mlr.press/v70/gaunt17a.html.

Gobet, Fernand and Peter J Jansen (2006). “Training in chess: A scientific approach.” In: *Education and chess*.

Grand View Research (2022). *Esports Market Size, Share & Trends Analysis Report By Revenue Source (Sponsorship, Advertising, Merchandise & Tickets, Media Rights), By Region, And Segment Forecasts, 2022 - 2030*. URL: https://www.grandviewresearch.com/industry-analysis/esports-market (visited on 03/21/2023).

Guid, Matej and Ivan Bratko (2006). “Computer analysis of world chess champions.” In: *ICGA journal* 29.2, pp. 65–73.

— (2011). “Using heuristic-search based engines for estimating human skill at chess.” In: *ICGA journal* 34.2, pp. 71–81.

— (2017). “Influence of search depth on position evaluation.” In: *Advances in computer games*. Springer, pp. 115–126.

Gulwani, Sumit, Oleksandr Polozov, and Rishabh Singh (2017). “Program Synthesis.” In: *Foundations and Trendső in Programming Languages* 4.12, pp. 1–119. ISSN: 2325-1107. DOI: 10.1561/2500000010.

Harrison, Brent and David L Roberts (2011). “Using sequential observations to model and predict player behavior.” In: *Proceedings of the 6th International Conference on Foundations of Digital Games*, pp. 91–98.

Haworth, Guy and Nelson Hernandez (2021). “The 20th Top Chess Engine Championship, TCEC20.” In: *J. Int. Comput. Games Assoc.* 43.1, pp. 62–73.

Hayes, Bradley and Julie A Shah (2017). “Improving robot controller transparency through autonomous policy explanation.” In: *Proceedings of the 2017 ACM/IEEE international conference on human-robot interaction*, pp. 303–312.

Hooper, D., K. Whyld, and K. Whyld (1996). “The Oxford Companion to Chess.” In: Oxford Companions Series. Oxford University Press, pp. 348–349. ISBN: 978-0-19-280049-7. URL: https://books.google.com/books?id=edEZAQAAIAAJ.

Hooshyar, Danial, Moslem Yousefi, and Heuiseok Lim (Jan. 2018). “Data-Driven Approaches to Game Player Modeling: A Systematic Literature Review.” In: *ACM Comput. Surv.* 50.6. ISSN: 0360-0300. DOI: 10.1145/3145814. URL: https://doi.org/10.1145/3145814.

Huang, Shengyi and Santiago Ontañón (May 2022). “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms.” en. In: *The International FLAIRS Conference Proceedings* 35. arXiv:2006.14171 [cs, stat]. ISSN: 2334-0762. DOI: 10.32473/flairs.v35i.130584. URL: http://arxiv.org/abs/2006.14171 (visited on 07/04/2023).

Inala, Jeevana Priya et al. (2020). “Synthesizing Programmatic Policies that Inductively Generalize.” In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=S1l8oANFDH.

Inc., Metafy (June 2024). *Play and Learn from the Best Pro Gamers in the World - Metafy*. URL: https://metafy.gg/ (visited on 06/09/2024).

Infil (May 2021). *The Fighting Game Glossary by Infil*. Accessed: 2024-06-10. URL: https://glossary.infil.net/?t=Okizeme.

Jackson, Phil and Tex Winter (2009). “NBA coaches playbook.” In: ed. by Giorgio Gandolfi. Human Kinetics Publishers. Chap. 8, pp. 89–109.

Kahlen, Stefan-Meyer (Apr. 2004). *UCI protocol*. URL: http://wbec-ridderkerk.nl/html/UCIProtocol.html (visited on 09/21/2022).

Kharif, Olga (Apr. 2020). “$1 Billion Video-Game Coaching Business Ramps Up During Lockdown.” en. In: *Bloomberg.com*. URL: https://www.bloomberg.com/news/articles/2020-04-23/how-big-is-market-for-video-game-coaches-more-than-1-billion.

Kiran, B Ravi et al. (2021). “Deep reinforcement learning for autonomous driving: A survey.” In: *IEEE Transactions on Intelligent Transportation Systems* 23.6, pp. 4909–4926.

Kiros, Ryan et al. (2015). "Skip-Thought Vectors." In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2015/hash/f442d33fa06832082290ad8544a8da27-Abstract.html.

Kliegr, Tomáš, Štěpán Bahník, and Johannes Fürnkranz (2021). "A review of possible effects of cognitive biases on interpretation of rule-based machine learning models." In: *Artificial Intelligence* 295, p. 103458. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2021.103458. URL: https://www.sciencedirect.com/science/article/pii/S0004370221000096.

Krishnan, Abhijeet and Chris Martens (Oct. 2022a). "Synthesizing interpretable chess tactics from player games." In: *Proceedings of the Workshop on Artificial Intelligence for Strategy Games (SG) and Esports Analytics (EA), 18th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. American Association for Artificial Intelligence.

— (Mar. 2022b). "Towards the automatic synthesis of interpretable chess tactics." In: *Proceedings of the Explainable Agency in Artificial Intelligence Workshop, 36th AAAI Conference on Artificial Intelligence*. American Association of Artificial Intelligence, pp. 91–97.

Krishnan, Abhijeet, Chris Martens, and Arnav Jhala (Mar. 2023). "Improving strategy synthesis for chess using precision and recall." In: [Manuscript submitted for publication].

Krishnan, Abhijeet, Colin M. Potts, et al. (June 2024). "Learning explainable representations of complex game-playing strategies." In: *Proceedings of the Eleventh Annual Conference on Advances in Cognitive Systems*. (to appear).

Lage, Isaac et al. (Oct. 2019). "Human Evaluation of Models Built for Interpretability." In: *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing* 7.1, pp. 59–67. DOI: 10.1609/hcomp.v7i1.5280. URL: https://ojs.aaai.org/index.php/HCOMP/article/view/5280.

Lázaro-Gredilla, Miguel et al. (Jan. 2019). "Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs." In: *Science Robotics* 4.26, eaav3150. DOI: 10.1126/scirobotics.aav3150.

Le, Quoc and Tomas Mikolov (June 2014). "Distributed Representations of Sentences and Documents." en. In: *Proceedings of the 31st International Conference on Machine Learning*. PMLR, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

Levine, Sergey et al. (Nov. 2020). *Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems*. arXiv:2005.01643 [cs, stat]. DOI: 10.48550/arXiv.2005.01643. URL: http://arxiv.org/abs/2005.01643 (visited on 06/07/2023).

Li, Junjie, Sotetsu Koyamada, et al. (2020). *Suphx: Mastering Mahjong with Deep Reinforcement Learning*. arXiv: 2003.13590 [cs.AI].

Li, Yuanzheng, Chaofan Yu, et al. (2023). "Deep Reinforcement Learning for Smart Grid Operations: Algorithms, Applications, and Prospects." In: *Proceedings of the IEEE* 111.9, pp. 1055–1096. DOI: 10.1109/JPROC.2023.3303358.

Li, Yujia, Felix Gimeno, et al. (2020). *Strong Generalization and Efficiency in Neural Programs*. arXiv: 2007.03629 [cs.LG].

Liang, Jacky et al. (May 2023). *Code as Policies: Language Model Programs for Embodied Control*. en. arXiv:2209.07753 [cs]. URL: http://arxiv.org/abs/2209.07753 (visited on 06/28/2023).

Lichess (Jan. 2021). *Chess rating systems*. URL: https://lichess.org/page/rating-systems (visited on 09/23/2022).

lichess.org (2021). *lichess.org open database*. URL: https://database.lichess.org/ (visited on 10/27/2021).

Liu, Guiliang et al. (2019). "Toward interpretable deep reinforcement learning with linear model u-trees." In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018, Proceedings, Part II 18*. Springer, pp. 414–429.

Lynch, John (Sept. 2017). *As NFL ratings drop, a new internet study says young men like watching eSports more than traditional sports*. URL: https://www.businessinsider.com/nfl-ratings-drop-study-young-men-watch-esports-more-than-traditional-sports-2017-9 (visited on 03/21/2023).

Machado, Marlos C, Eduardo PC Fantini, and Luiz Chaimowicz (2011). "Player modeling: Towards a common taxonomy." In: *2011 16th international conference on computer games (CGAMES)*. IEEE, pp. 50–57.

Madumal, Prashan et al. (Apr. 2020). "Explainable Reinforcement Learning through a Causal Lens." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.03, pp. 2493–2500. DOI: 10.1609/aaai.v34i03.5631. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5631.

Manna, Zohar and Richard Waldinger (Jan. 1980). "A Deductive Approach to Program Synthesis." In: *ACM Trans. Program. Lang. Syst.* 2.1, pp. 90–121. ISSN: 0164-0925. DOI: 10.1145/357084.357090.

Mariño, Julian R. H., Rubens O. Moraes, et al. (May 2021). "Programmatic Strategies for Real-Time Strategy Games." In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.1, pp. 381–389. DOI: 10.1609/aaai.v35i1.16114. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16114.

Mariño, Julian RH and Claudio FM Toledo (2022). "Evolving interpretable strategies for zero-sum games." In: *Applied Soft Computing* 122, p. 108860.

Martinez-Garza, Mario M. and Douglas B. Clark (2017). "Two Systems, Two Stances: A Novel Theoretical Framework for Model-Based Learning in Digital Games." en. In: *Instructional Techniques to Facilitate Learning and Motivation of Serious Games*. Ed. by Pieter Wouters and Herre van Oostendorp. Cham: Springer International Publishing, pp. 37–58. ISBN: 978-3-319-39298-1. DOI: 10.1007/978-3-319-39298-1_3. URL: https://doi.org/10.1007/978-3-319-39298-1_3.

McCarthy, John (1990). "Chess as the Drosophila of AI." In: *Computers, chess, and cognition*. Springer, pp. 227–237.

McIlroy-Young, Reid et al. (2020). "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*. KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, pp. 1677–1687. ISBN: 9781450379984. DOI: 10.1145/3394486.3403219. URL: https://doi.org/10.1145/3394486.3403219.

Medeiros, Leandro C., David S. Aleixo, and Levi H. S. Lelis (Mar. 2022). "What can we Learn Even From the Weakest? Learning Sketches for Programmatic Strategies." en. In: arXiv:2203.11912. arXiv:2203.11912 [cs]. URL: http://arxiv.org/abs/2203.11912.

Merriam-Webster (2024). *Strategy*. URL: https://www.merriam-webster.com/dictionary/strategy (visited on 06/16/2024).

Mesentier Silva, Fernando de et al. (2016). "Generating heuristics for novice players." In: *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, pp. 1–8.

Moraes, Rubens (July 2021). *SynProS - Synthesis of Programmatic Strategies*. URL: https://rubensolv.github.io/synpros-microrts/ (visited on 03/26/2023).

Morales, Eduardo (1992). "First order induction of patterns in Chess." PhD thesis. PhD thesis, The Turing Institute-University of Strathclyde.

Mosavi, Amirhosein et al. (2020). “Comprehensive Review of Deep Reinforcement Learning Methods and Applications in Economics.” In: *Mathematics* 8.10. ISSN: 2227-7390. DOI: 10.3390/math8101640. URL: https://www.mdpi.com/2227-7390/8/10/1640.

Muggleton, Stephen H et al. (2018). “Ultra-strong machine learning: comprehensibility of programs learned with ILP.” In: *Machine Learning* 107.7, pp. 1119–1140.

Narayanan, Menaka et al. (Feb. 2018). “How do Humans Understand Explanations from Machine Learning Systems? An Evaluation of the Human-Interpretability of Explanation.” In: arXiv:1802.00682. number: arXiv:1802.00682 arXiv:1802.00682 [cs]. DOI: 10.48550/arXiv.1802.00682. URL: http://arxiv.org/abs/1802.00682.

Neelakantan, Arvind, Quoc V. Le, and Ilya Sutskever (2016). “Neural Programmer: Inducing Latent Programs with Gradient Descent.” In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1511.04834.

Newbury, Rhys et al. (2023). “Deep Learning Approaches to Grasp Synthesis: A Review.” In: *IEEE Transactions on Robotics* 39.5, pp. 3994–4015. DOI: 10.1109/TRO.2023.3280597.

Nogueira, Pedro Alves et al. (2014). “Fuzzy affective player models: A physiology-based hierarchical clustering method.” In: *Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Ontañón, Santiago et al. (Mar. 2018). “The First microRTS Artificial Intelligence Competition.” en. In: *AI Magazine* 39.11, pp. 75–83. ISSN: 2371-9621. DOI: 10.1609/aimag.v39i1.2777.

OpenAI et al. (2019). *Dota 2 with Large Scale Deep Reinforcement Learning*. arXiv: 1912.06680 [cs.LG].

Palangi, Hamid et al. (Apr. 2016). “Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval.” In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 24.4, pp. 694–707. ISSN: 2329-9304. DOI: 10.1109/TASLP.2016.2520371.

Paredes-Olay, Concepción et al. (Aug. 2002). “Transfer of control between causal predictive judgments and instrumental responding.” en. In: *Animal Learning & Behavior* 30.3, pp. 239–248. ISSN: 1532-5830. DOI: 10.3758/BF03192833.

Pattis, Richard E (1994). *Karel the robot: a gentle introduction to the art of programming*. John Wiley & Sons.

Pfau, Johannes, Jan David Smeddinck, and Rainer Malaka (2018). "Towards Deep Player Behavior Models in MMORPGs." In: *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*. CHI PLAY 18. Melbourne, VIC, Australia: Association for Computing Machinery, pp. 381–392. ISBN: 9781450356244. DOI: 10.1145/3242671.3242706. URL: https://doi.org/10.1145/3242671.3242706.

Puiutta, Erika and Eric MSP Veith (2020). "Explainable reinforcement learning: A survey." In: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, pp. 77–95.

Qiu, Wenjie and He Zhu (2022). "Programmatic Reinforcement Learning without Oracles." In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=6Tk2noBdvxt.

Radford, Alec et al. (June 2018). "Improving language understanding by generative pre-training." In: URL: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Ramesh, Aditya et al. (July 2021). "Zero-Shot Text-to-Image Generation." In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 8821–8831. URL: https://proceedings.mlr.press/v139/ramesh21a.html.

Reitman, Jason G et al. (2020). "Esports research: A literature review." In: *Games and Culture* 15.1, pp. 32–50.

Rios, Luis Miguel and Nikolaos V Sahinidis (2013). "Derivative-free optimization: a review of algorithms and comparison of software implementations." In: *Journal of Global Optimization* 56, pp. 1247–1293.

Romero, Oscar (2019). "Computer analysis of world chess championship players." In: *ICSEA 2019*, p. 212.

Rose, Lee (2012). *Winning basketball fundamentals*. Human Kinetics.

Rouse, W.B., J.A. Cannon-Bowers, and E. Salas (1992). "The role of mental models in team performance in complex systems." In: *IEEE Transactions on Systems, Man, and Cybernetics* 22.6, pp. 1296–1308. DOI: 10.1109/21.199457.

Schubert, Matthias, Anders Drachen, and Tobias Mahlmann (2016). "Esports analytics through encounter detection." In: *Proceedings of the MIT Sloan Sports Analytics Conference*. Vol. 1. MIT Sloan Boston, MA, p. 2016.

Seirawan, Yasser (2005a). *Winning chess strategies*. Everyman Chess.

— (2005b). *Winning chess tactics*. Everyman Chess.

Shadow Esports (2023). *Shadow | Game-changing esports analytics - Shadow*. en. URL: https://shadow.gg (visited on 03/21/2023).

Sherry, Ben (Aug. 2022). *How Catering to Top Talent Changed the Game for This Coaching Platform*. en. URL: https://www.inc.com/ben-sherry/metafy-gaming-coach.html (visited on 06/14/2024).

Shin, Eui Chul, Illia Polosukhin, and Dawn Song (2018). "Improving Neural Program Synthesis with Inferred Execution Traces." In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/7776e88b0c189539098176589250bcba-Paper.pdf.

Shu, Tianmin, Caiming Xiong, and Richard Socher (2017). "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning." In: *arXiv preprint arXiv:1712.07294*.

Sieusahai, Alexander and Matthew Guzdial (Oct. 2021). "Explaining Deep Reinforcement Learning Agents in the Atari Domain through a Surrogate Model." In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 17.1, pp. 82–90. DOI: 10.1609/aiide.v17i1.18894. URL: https://ojs.aaai.org/index.php/AIIDE/article/view/18894.

Silman, Jon (Aug. 2021). *Fighting Games Explained: What is Frame Data and How to Use it to Your Advantage?* URL: https://compete.playstation.com/en-us/all/articles/fighting-games-explained-what-is-frame-data-and-how-to-use-it-to-your-advantage (visited on 03/24/2023).

Silver, David, Aja Huang, et al. (Jan. 2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* 529.7587, pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://doi.org/10.1038/nature16961.

Silver, David, Thomas Hubert, et al. (Dec. 2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419. Publisher: American Association for the Advancement of Science, pp. 1140–1144. DOI: 10.1126/science.aar6404. URL: https://doi.org/10.1126/science.aar6404 (visited on 08/14/2023).

Sipser, Michael (1996). *Introduction to the Theory of Computation*. 1st. International Thomson Publishing. ISBN: 053494728X.

Slack, Dylan et al. (2019). "Assessing the local interpretability of machine learning models." In: *arXiv preprint arXiv:1902.03501*.

Smith, Adam M et al. (2011). “An inclusive view of player modeling.” In: *Proceedings of the 6th International Conference on Foundations of Digital Games*, pp. 301–303.

Snodgrass, Sam, Omid Mohaddesi, and Casper Harteveld (2019). “Towards a Generalized Player Model through the PEAS Framework.” In: *Proceedings of the 14th International Conference on the Foundations of Digital Games*. FDG 19. San Luis Obispo, California: Association for Computing Machinery. ISBN: 9781450372176. DOI: 10.1145/3337722.3341856. URL: https://doi.org/10.1145/3337722.3341856.

Speith, Timo (June 2022). “A Review of Taxonomies of Explainable Artificial Intelligence (XAI) Methods.” In: *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. FAccT 22. New York, NY, USA: Association for Computing Machinery, pp. 2239–2250. ISBN: 978-1-4503-9352-2. DOI: 10.1145/3531146.3534639. URL: https://dl.acm.org/doi/10.1145/3531146.3534639.

Spronck, Pieter, Ida Sprinkhuizen-Kuyper, and Eric Postma (2004). “Online adaptation of game opponent AI with dynamic scripting.” In: *International Journal of Intelligent Games and Simulation* 3.1, pp. 45–53.

Sterling, Leon and Ehud Shapiro (1994). “The Art of Prolog: Advanced Programming Techniques.” In: 2nd. MIT Press, pp. 87–90.

Summerville, Adam et al. (2016). “Learning player tailored content from observation: Platformer level generation from video traces using lstms.” In: *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Sun, Shao-Hua et al. (July 2018). “Neural Program Synthesis from Diverse Demonstration Videos.” In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 4790–4799. URL: https://proceedings.mlr.press/v80/sun18a.html.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Szabo, Alexander (1984). “Computer chess tactics and strategy.” PhD thesis. University of British Columbia. DOI: http://dx.doi.org/10.14288/1.0051870. URL: https://open.library.ubc.ca/collections/ubctheses/831/items/1.0051870.

The Stockfish team (July 2021). *Stockfish 14*. URL: https://stockfishchess.org/blog/2021/stockfish-14/ (visited on 06/14/2024).

The Stockfish team (Dec. 2022). *Stockfish 15.1*. URL: https://stockfishchess.org/blog/2022/stockfish-15-1/ (visited on 01/14/2023).

Towers, Mark et al. (Mar. 2023). *Gymnasium*. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (visited on 07/08/2023).

Trivedi, Dweep et al. (2021). "Learning to Synthesize Programs as Interpretable and Generalizable Policies." In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., pp. 25146–25163. URL: https://proceedings.neurips.cc/paper/2021/hash/d37124c4c79f357cb02c655671a432fa-Abstract.html (visited on 06/07/2023).

Vaswani, Ashish et al. (2017). "Attention is All you Need." In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Verma, Abhinav, Hoang Le, et al. (2019). "Imitation-Projected Programmatic Reinforcement Learning." In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/5a44a53b7d26bb1e54c05222f186dcfb-Paper.pdf.

Verma, Abhinav, Vijayaraghavan Murali, et al. (July 2018). "Programmatically Interpretable Reinforcement Learning." In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 5045–5054. URL: https://proceedings.mlr.press/v80/verma18a.html.

Vinyals, Oriol et al. (Nov. 2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature* 575.7782, pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. URL: https://doi.org/10.1038/s41586-019-1724-z.

Waa, Jasper van der et al. (2018). "Contrastive explanations for reinforcement learning in terms of expected consequences." In: *arXiv preprint arXiv:1807.08706*.

Wang, Pengcheng et al. (2018). "High-Fidelity Simulated Players for Interactive Narrative Planning." In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. IJCAI 18. Stockholm, Sweden: AAAI Press, pp. 3884–3890. ISBN: 9780999241127.

Wielemaker, Jan (Jan. 2003). "An Overview of the SWI-Prolog Programming Environment." In: *Proceedings of the 13th International Workshop on Logic Programming Environments*, pp. 1–16.

Wing, Jeannette M (2008). “Computational thinking and thinking about computing.” In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366.1881, pp. 3717–3725.

Xu, Danfei et al. (2018). “Neural Task Programming: Learning to Generalize Across Hierarchical Tasks.” In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3795–3802. DOI: 10.1109/ICRA.2018.8460689.

Yang, Yichen David et al. (Nov. 2021). *Program Synthesis Guided Reinforcement Learning for Partially Observed Environments*. en. arXiv:2102.11137 [cs]. URL: http://arxiv.org/abs/2102.11137 (visited on 06/28/2023).

Zhou, S. Kevin et al. (2021). “Deep reinforcement learning in medical imaging: A literature review.” In: *Medical Image Analysis* 73, p. 102193. ISSN: 1361-8415. DOI: https://doi.org/10.1016/j.media.2021.102193. URL: https://www.sciencedirect.com/science/article/pii/S1361841521002395.

APPENDICES

APPENDIX

A

ABBREVIATIONS

A summary of abbreviations used in alphabetical order.

AI	Artificial Intelligence
CEM	Cross-Entropy Method
CFG	context-free grammar
DNN	deep neural network
DRL	deep reinforcement learning
DSL	domain-specific language
DT	decision transformer
FOL	first-order logic
GA	genetic algorithm
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Unit
HCI	Human-Computer Interaction

ILP	Inductive Logic Programming
ISS	Interpretable Strategy Synthesis
ITS	Interactive Tutoring System
MDP	Markov Decision Process
ML	machine learning
MMORPG	Massively Multiplayer Online Role-Playing Game
NN	neural network
PAL	Patterns and Learning
RL	reinforcement learning
SA	simulated annealing
SMT	Satisfiability Modulo Theories
SynProS	Synthesis of Programmatic Strategies
TCEC	Top Chess Engine Championship
TD	Temporal-Difference
UCI	Universal Chess Interface
UCT	Upper Confidence Bound for Trees
XAI	Explainable Artificial Intelligence
XRL	Explainable Reinforcement Learning
rlgg	relative least general generalization

APPENDIX

B

SYMBOLS

A summary of symbols used in order of use.

Symbol	Description
$\mathcal{S}$	state space
$\mathcal{A}$	action space
$\mathcal{P}$	state transition function
$\mathcal{R}$	reward function
γ	discount factor
G_t	return
I	interpretability measure
$\mathcal{G}$	game environment
$\mathcal{M}$	strategy model
σ	strategy
B	background knowledge

Symbol	Description
E^+	positive examples
E^-	negative examples
π	policy
Cp	centipawn

APPENDIX

C

BACKGROUND KNOWLEDGE DEFINITIONS

This appendix has all the background knowledge definitions used by the chess strategy learning system described in Section §4.5. These do not include any helper rules used to define the predicates in the background knowledge since those are not used to define tactics. The complete set of definitions, including helper rules, has been written in Prolog and can be found at https://github.com/AbhijeetKrishnan/prolog-chess.

- `legal_move(Board,Move)`: Defines legal moves of chess pieces.
- `in_check(Board,Side,Square)`: Defines whether a particular side is in check by a piece.
- `pawn_capture(Board,Move)`: Defines a pawn capture move.
- `castling_move(Board,Move)`: Defines valid castling move.
- `make_move(Board,Move,NewBoard)`: Performs the actual movement of a piece by changing the state appropriately.
- `can_capture(Board,Sq1,Sq2)`: Defines whether a piece on square `Sq1` can capture another on square `Sq2`.
- `is_capture(Board,Move)`: Defines valid capture move.

- `en_passant(Board,Square)`: Defines the en passant square on the current board, if any.
- `queenside_castle(Board,Side)`: Defines the queenside castling rights of the side.
- `kingside_castle(Board,Side)`: Defines the kingside castling rights of the side.
- `turn(Board,Side)`: Defines who's turn it is to make a move on the current board.
- `is_attacked(Board,Square,SquareSet)`: Defines the squares that are being attacked by a piece.
- `can_attack(Board,Sq1,Sq2)`: Defines an attack being carried out by a piece on one square, on another square.
- `is_empty(Board,SquareSet)`: Defines a set of empty squares on the board.
- `xrays(Board,Behind,Middle,Front)`: Defines a piece that x-rays another piece.
- `valid_piece_at(Board,PieceType,Col,Sq)`: Defines the existence of a piece and its attributes at a given square.
- `remove_piece_at(Board,Square,NewBoard)`: Removes a piece from the given square.
- `move(Move,From,To,PromoList)`: Unifies the list representation of a move with its component parts, for e.g., `move([a7,a8,q],a7,a8,[q])`.
- `other_color(Color,OtherColor)`: Defines a relationship between the two opposing colors on the board.
- `color(Color)`: Defines a valid color.
- `sliding(PieceType)`: Defines a piece that moves in a straight line.
- `piece_type(PieceType)`: Defines a valid piece type.
- `piece(Piece)`: Defines a valid piece.
- `sq_between_non_incl(Sq1,Sq2,Sq2)`: Square `Sq3` is in the same straight line defined by squares `Sq1` and `Sq2` (non-inclusive).
- `different(Sq1,Sq2)`: Squares `Sq1` and `Sq2` are different.