UniMaia: Steering Chess Policies with Language for Human-like Play

by

Sherman Siu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Sherman Siu 2026

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Recent advances in large language models have enabled natural language to serve as a flexible interface for controlling complex systems, but often require large-scale multimodal training or sacrifice domain-specific inductive biases. In structured decision-making domains such as chess, specialized models achieve strong performance but lack high-level semantic controllability, while prompt-conditioned approaches are more flexible but typically exhibit weaker domain grounding.

In this thesis, we study prompt-conditioned policy modulation for chess by adapting a pretrained neural policy network using natural language prompts. We propose **UniMaia**, a framework that combines a frozen Lc0-based chess policy network with a LoRA-adapted text encoder and a ControlNet-style conditioning mechanism. This design enables semantic conditioning over gameplay, providing a more expressive alternative to discrete metadata for modeling human play while preserving the underlying representations of the base model. We further introduce **UniMaia-Aux**, an extension that incorporates auxiliary temporal conditioning and behavioral prediction objectives.

To support this work, we construct a large-scale, metadata-augmented version of the Lichess dataset, introduce a semi-automated pipeline for generating natural language prompt templates, and propose evaluation benchmarks spanning both prompt-conditioned and metadata-conditioned settings.

Empirically, UniMaia achieves competitive or superior performance relative to prior work across multiple benchmarks. It attains the highest top-move accuracy on prompt-conditioned benchmarks while remaining competitive with metadata-conditioned models on human move prediction tasks. Prompt-conditioned models perform strongly in frequency-dominated regimes, such as common openings and highly active player behavior, whereas metadata-conditioned models generally achieve stronger expected accuracy. UniMaia bridges these approaches by combining strong domain-specific inductive biases with flexible prompt-based control.

UniMaia-Aux further demonstrates that auxiliary temporal conditioning can improve expected accuracy and behavioral modeling across several evaluation settings, although this introduces trade-offs between top-move accuracy and dependence on temporally structured information.

Overall, this work demonstrates that prompt-conditioned control of domain-specific policy networks is feasible without end-to-end multimodal training. At the same time, the results highlight ongoing challenges related to prompt sensitivity, policy calibration, robustness, and the trade-offs between controllability and predictive performance in prompt-conditioned decision-making systems.

Acknowledgements

I would like to thank my co-supervisors, Lesley Istead and Jeff Orchard, for their guidance and support. I am also grateful to my parents for their encouragement throughout the process of completing this thesis.

I would like to thank my former bosses, Mila Grandes, Austin Wald, and Daniel Yakubov, for accommodating my work schedule and enabling me to complete this work.

Thank you all for making this thesis possible.

Table of Contents

Author's Declaration ii

Abstract iii

Acknowledgements iv

List of Figures x

List of Tables xiii

List of Abbreviations xvi

1 Introduction 1

2 Background 4

2.1 Chess 4

2.1.1 Brief Overview of Chess 4

2.1.2 Chess Notation 5

2.1.3 Rating System 5

2.2 LeelaChess Zero 6

2.2.1 Input Encoding 7

2.2.2 Output Encoding 7

3 Related Work **9**

3.1 Chess Engines . . . 9

3.2 Metadata-Conditioned Human Move Prediction . . . 10

3.3 Prompt-Conditioned Policies . . . 12

3.4 Multimodal Conditioning Methods . . . 12

4 Problem Formulation **14**

5 Dataset Construction **15**

5.1 Data Sources . . . 15

5.2 Creating LichessGames . . . 17

5.3 Creating the Prompt Templates . . . 20

5.3.1 Creating LichessTemplates-Pretrain . . . 21

5.3.2 Creating LichessTemplates-Instruct . . . 21

5.4 Metadata Preprocessing for Prompt Templates . . . 22

5.4.1 Enriching Player Metadata with Real Names . . . 23

5.4.2 Obtaining Opening Move Sequences . . . 23

6 Model Architecture and Training **26**

6.1 Experimental Setup . . . 26

6.1.1 Training Data and Procedure . . . 26

6.1.2 Optimization . . . 27

6.1.3 Evaluation Protocol . . . 27

6.1.4 Baseline Configuration . . . 28

6.2 Design Evolution and Key Improvements . . . 29

6.3 Ablation Studies . . . 31

6.3.1 Architecture Design . . . 31

6.3.2 Optimization Strategy . . . 35

6.3.3 Parameter-Efficient Fine-Tuning . . . 37
6.3.4 Lc0 Scaling . . . 38
6.3.5 Data Sampling . . . 39
6.3.6 LoRA Rank Ablations . . . 41
6.3.7 Learning Rate Schedule Ablations . . . 42
6.3.8 Negative Results . . . 44
6.4 Reproducibility and Sources of Non-Determinism . . . 51
6.5 Full Training Run . . . 52
6.5.1 Final Model Architecture . . . 52
6.5.2 Final Training Configuration . . . 54
6.5.3 Training Dynamics . . . 55
6.6 Fine-tuning on Auxiliary Targets . . . 58
6.6.1 Motivation and Dataset . . . 58
6.6.2 Prompt Augmentation . . . 58
6.6.3 Auxiliary Prediction Heads . . . 59
6.6.4 Training Dynamics . . . 59
6.7 Training Cost and Accessibility . . . 63

7 Evaluation **64**
7.1 Benchmarks . . . 64
7.1.1 Opening Tasks . . . 64
7.1.2 Prompt-Conditioned Tasks . . . 65
7.1.3 Metadata-Conditioned Tasks . . . 67
7.1.4 Evaluation Protocol Details . . . 68
7.1.5 Evaluation Metrics . . . 68
7.2 Models . . . 68
7.2.1 Engine-Based Models . . . 68
7.2.2 Metadata-Conditioned Models . . . 69

7.2.3 Prompt-Conditioned Models . . . 71
7.2.4 Model Comparison . . . 73
7.3 Results . . . 76
7.3.1 Prompt-Conditioned . . . 78
7.3.2 Metadata-Conditioned . . . 83
7.3.3 Performance by Elo Range . . . 85

8 Analysis of Trained Model **87**
8.1 Policy Continuity Across Elo . . . 87
8.2 Mechanistic Interpretability . . . 90
8.2.1 Layer-wise Contribution . . . 90
8.2.2 Cross-attention Sinks . . . 93
8.3 Failure Modes and Stress Cases . . . 94

9 Discussion and Future Work **96**
9.1 Key Findings . . . 97
9.2 Limitations . . . 99
9.2.1 Prompt Design and Data Quality . . . 99
9.2.2 Data Representation and Training Objective . . . 99
9.2.3 Architectural Trade-offs . . . 100
9.2.4 Evaluation and Analysis . . . 101
9.2.5 Mechanistic Interpretability . . . 101
9.3 Future Work . . . 102
9.3.1 Prompt Modeling . . . 102
9.3.2 Prompt and Structured Conditioning . . . 102
9.3.3 Architectural Improvements . . . 103
9.3.4 Training Objectives . . . 103
9.3.5 Expanded Data Sources . . . 104

9.3.6 Evaluation and Deployment . . . 104
9.3.7 Mechanistic Interpretability . . . 104
9.4 Conclusion . . . 104

References **106**

APPENDICES **120**

A Termination Statistics **121**

B Policy Continuity Evaluation Details **123**
B.1 Policy Continuity Evaluation Prompts . . . 123
B.1.1 Policy Continuity Across Positions . . . 123
B.1.2 Policy Continuity Under Opening Conditioning . . . 125
B.2 Quantitative Diagnostics of Policy Continuity . . . 126

C Prompt Templates **131**
C.1 Example LichessTemplates Prompt Templates . . . 131
C.2 Metadata-Conditioned Benchmark Prompts . . . 132
C.3 GM PGN Header Prompt . . . 132
C.4 Elo and Time Control PGN Header . . . 133
C.5 Auxiliary Targets Prompt Suffix . . . 133

D Detailed Evaluation Results **135**
D.1 Top-Move Accuracy . . . 135
D.2 Expected Accuracy . . . 135
D.2.1 Prompt-Conditioned . . . 136
D.2.2 Metadata-Conditioned . . . 136
D.2.3 Inference Strategy Ablations . . . 137
D.3 Performance by Elo Range . . . 137

Glossary **155**

List of Figures

5.1 Data processing diagram. 16

6.1 Comparison of training losses between the initial and final models. 29
6.2 The simple moving averages of the training losses for the architecture ablations. 33
6.3 Simple moving averages of the losses for each optimizer. 36
6.4 Simple moving averages of the training losses for the PEFT ablations. . . . 38
6.5 The three learning rate schedules. 43
6.6 Simple moving averages of the training losses for the learning rate schedule ablations. 44
6.7 Training loss (left) and entropy (right) over the course of training. 47
6.8 The training losses for two local and two server runs, all with the same training configuration. 52
6.9 Model architecture diagram. 53
6.10 Training losses for the full training run. Left: raw training loss. Right: simple moving average of the loss (window size 1,024). 55
6.11 Top move accuracy and expected accuracy for five benchmarks across monthly checkpoints during training. 56
6.12 Training dynamics for LOB-P templates, grouped by final performance. . . 57
6.13 The simple moving averages of the policy entropy, policy loss, and moves-left loss throughout auxiliary-target fine-tuning. 60
6.14 Top move accuracy and expected accuracy for seven benchmarks across monthly checkpoints during auxiliary-target fine-tuning. 61

6.15 Top move accuracy and expected accuracy for seven benchmarks across monthly checkpoints during auxiliary-target fine-tuning, where the first 10 plies are omitted from evaluation. 61

6.16 Top move accuracy and expected accuracy for LIF-Aux and LGB-Aux across monthly checkpoints during auxiliary-target fine-tuning. 62

6.17 Top move accuracy and expected accuracy for LIF-Aux and LGB-Aux across monthly checkpoints during auxiliary-target fine-tuning, where the first 10 plies are omitted from evaluation. 62

7.1 LIF accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy. 86

7.2 ABB accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy. 86

8.1 Top-5 move probabilities predicted by UniMaia as a function of Elo. 88

8.2 Top-5 move probabilities predicted by UniMaia as a function of Elo for opening-conditioned prompts. 89

8.3 Distribution of the mean and maximum absolute residual updates for each encoder layer and the controllable policy head. 90

8.4 Mean and maximum absolute residual updates as a function of ply index for layer 14 and the controllable policy head. 91

8.5 Residual concentration ratio for each encoder layer and the controllable policy head. 92

8.6 Left: Distribution of cosine similarity between flattened encoder layer 13 outputs of UniMaia and Lc0. Middle: Cosine similarity as a function of mean player . Right: Cosine similarity as a function of ply index. 93

8.7 Mean absolute cross-attention activations, averaged over attention heads. Left: encoder layer 14. Right: controllable policy head. 94

B.1 Quantitative policy continuity diagnostics across Elo values (early opening). 127

B.2 Quantitative policy continuity diagnostics across Elo values (late opening). 128

B.3 Quantitative policy continuity diagnostics across Elo values (middlegame). 129

B.4 Quantitative policy continuity diagnostics across Elo values (endgame). . . 130

D.1 Number of LIF entries in each rounded White and Black Elo bucket. . . . 143
D.2 Number of ABB entries in each rounded White and Black Elo bucket. . . . 144
D.3 LIF-D accuracy as a function of White and Black Elo. Left: top-move accuracy. Right: expected accuracy. 145
D.4 Number of LIF-D entries in each rounded White and Black Elo bucket. . . 146
D.5 LIF-T10 accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy 147
D.6 Number of LIF-T10 entries in each rounded White and Black Elo bucket. . 148
D.7 LGB accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy 149
D.8 Number of LGB entries in each rounded White and Black Elo bucket. . . . 150
D.9 M1-S accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy 151
D.10 Number of M1-S entries in each rounded White and Black Elo bucket. . . . 152
D.11 M2R accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy 153
D.12 Number of M2R entries in each rounded White and Black Elo bucket. . . . 154

List of Tables

5.1 Summary statistics of the LichessGames dataset. 17
5.2 The fields extracted from the Lichess PGN games in Stage 1. 19
5.3 The fields extracted from the Lichess PGN games in Stage 2. 20
5.4 Auxiliary metadata sources used during training-example and benchmark construction. 23

6.1 Summary of sequential improvements over the course of the ablation study. 30
6.2 Parameter changes associated with each design decision. 30
6.3 Architecture ablation results. 34
6.4 Full fine-tuning results for selected fusion architectures. 35
6.5 Optimizer ablation results. 36
6.6 PEFT ablation results. 37
6.7 PEFT parameter counts. 38
6.8 Leela Chess Zero (Lc0) ablation results. 39
6.9 Downsampling ablation results. 41
6.10 LoRA rank ablation results. 42
6.11 Learning rate schedule ablation results. 43
6.12 Data ablation results. 45
6.13 LLM ablation results. 46
6.14 Fine-tuned base model ablation results. 48

6.15 Fine-tuned base model ablations evaluated on opening instruction-following benchmarks. . . . 48
6.16 LoRA target modules ablation results. . . . 49
6.17 LoRA rank with scaled α ablation results. . . . 50
6.18 Key padding mask ablation results. . . . 51
6.19 Results for repeated runs across local and server execution environments. . . . 51
6.20 Top move accuracy for five benchmarks at December checkpoints for each year of training, along with the cumulative number of positions processed up to each checkpoint. . . . 58

7.1 Comparison of model inputs. . . . 74
7.2 Comparison of model outputs. . . . 74
7.3 Comparison of training configurations for structured-input models. . . . 75
7.4 Comparison of training configurations for language-based models. . . . 75
7.5 UniMaia training configuration. . . . 77
7.6 UniMaia-Aux training configuration. . . . 77
7.7 Prompt-conditioned benchmark results, measured by Acc@1. . . . 79
7.8 Prompt-conditioned benchmark results with the first 10 plies omitted, measured by Acc@1. . . . 80
7.9 Prompt-conditioned results, measured by $\mathbb{E}$[Acc]. . . . 80
7.10 Prompt-conditioned results with the first 10 plies omitted, measured by $\mathbb{E}$[Acc]. 81
7.11 Auxiliary targets results. . . . 82
7.12 Auxiliary targets results with the first 10 plies omitted. . . . 83
7.13 Metadata-conditioned benchmark prediction results with the first 10 plies omitted, measured by Acc@1. . . . 84
7.14 Metadata-conditioned results with the first 10 plies omitted, measured by $\mathbb{E}$[Acc]. . . . 84

A.1 Termination reason frequencies for the only shard of March 2014 games in LichessGames. . . . 121

A.2 Termination reason frequencies for the first shard of October 2022 games in LichessGames. 122
A.3 Termination reason frequencies for the first shard of January 2023 games in LichessGames. 122

B.1 PGNs and prompt templates used for policy continuity evaluation across representative positions (early opening, late opening, middlegame, and endgame). 124

D.1 Metadata-conditioned results, measured by Acc@1. 136
D.2 Prompt-conditioned results, measured by $\mathbb{E}$[Acc]. 137
D.3 Prompt-conditioned results with the first 10 plies omitted, measured by $\mathbb{E}$[Acc]. 138
D.4 Metadata-conditioned results, measured by $\mathbb{E}$[Acc]. 139
D.5 Metadata-conditioned results with the first 10 plies omitted, measured by $\mathbb{E}$[Acc]. 140
D.6 ChessGPT prompt-conditioned results, measured by Acc@1. 140
D.7 ChessGPT prompt-conditioned results with the first 10 plies omitted, measured by Acc@1. 141
D.8 ChessGPT metadata-conditioned results, measured by Acc@1. 141
D.9 ChessGPT metadata-conditioned results with the first 10 plies omitted, measured by Acc@1. 141
D.10 GPT-3.5 prompt-conditioned results, measured by Acc@1. 142
D.11 GPT-3.5 prompt-conditioned results with the first 10 plies omitted, measured by Acc@1. 142
D.12 GPT-3.5 metadata-conditioned results, measured by Acc@1. 142
D.13 GPT-3.5 metadata-conditioned results with the first 10 plies omitted, measured by Acc@1. 142

List of Abbreviations

CF ChessFormer 4, 6, 28, 38, 39, 52, 69, 70, 91

CNN convolutional neural network 4, 6, 69

ECO Encyclopedia of Chess Openings 19, 23–25, 64

FEN Forsyth–Edwards Notation 5, 10, 71

Lc0 Leela Chess Zero iii, xi, xiii, 2, 4, 6–8, 10, 11, 14, 26, 28, 30–33, 38, 39, 47, 52, 54, 59, 68–71, 73, 74, 76, 90–93, 100, 101, 104

LLM large language model 1, 4, 11, 12, 20, 30, 35, 46, 47, 99, 102

MCTS Monte Carlo Tree Search 6, 8, 10, 11

NNUE Efficiently Updatable Neural Network 10, 14, 155

OTB over-the-board 66

PGN Portable Game Notation xiii, xv, 5, 6, 11, 15–20, 22, 23, 25, 48, 49, 56, 57, 65, 66, 71–73, 79, 80, 84, 87, 123–125, 132, 133, 136, 140–142

SAN Standard Algebraic Notation 5, 72

TCEC Top Chess Engine Competition 10

UCI Universal Chess Interface 5, 7, 14, 70

Chapter 1

Introduction

Recent advances in large language models (LLMs) have significantly expanded the use of natural language as a flexible interface for automation and decision-making. In particular, through tool use [75, 82, 14, 40], many of these models are now capable of carrying out economically viable real-world tasks, as demonstrated by systems such as OpenAI Codex [72] and Claude Cowork [5]. For specialized domains, open-source LLMs are often fine-tuned to achieve additional performance gains. However, in many structured domains, fine-tuning language models alone is insufficient, and domain-specific models with strong inductive biases remain essential for achieving high performance.

There are inherent trade-offs between **prompt-conditioned** and **metadata-conditioned** policies. Metadata-conditioned models capture structured variables such as player rating and time control, but are limited to predefined conditioning signals. Human decision-making, however, depends on richer semantic concepts such as playing style, strategic intent, and opening preferences, which are more naturally expressed through language. Prompt-conditioned models provide this flexibility, but often sacrifice the domain-specific inductive biases needed for accurate policy prediction. This capability is particularly valuable in chess, where players may wish to practice against opponents with specific openings, skill levels, or playing styles. While end-to-end multimodal models provide a unified framework for reasoning and action, they typically require substantial compute, large-scale curated datasets, and significant engineering effort. Instead, we seek to augment pretrained domain-specific policies with natural language, retaining their strong inductive biases while enabling semantic control.

In this thesis, we introduce **UniMaia**, a framework for prompt-conditioned modulation of pretrained policy networks. Specifically, we investigate whether natural language can serve as a flexible conditioning signal for pretrained chess policies, enabling both semantic control and improved modeling of human play while preserving the latent representations of the original model. By combining a pretrained text encoder with a frozen chess policy network, UniMaia achieves competitive performance on human move prediction benchmarks with relatively limited training. This approach enables control over player strength and style, supports prompt-driven behaviors such as opening selection, and facilitates analysis of emergent behaviors while retaining the inductive biases of the pretrained chess model. This may enable more personalized training and analysis tools, allowing players to practice specific openings, opponent strengths, or playing styles through natural language.

We further investigate auxiliary temporal conditioning and behavioral prediction objectives through UniMaia-Aux, an extension that incorporates temporal game metadata and auxiliary supervision related to game outcome, remaining plies, move delay, and termination behavior. These auxiliary objectives improve behavioral modeling and expected accuracy across several evaluation settings, while revealing trade-offs between controllability, top-move accuracy, and expected accuracy.

Through this process, we make the following contributions:

1. A parameter-efficient framework for **prompt-conditioned modulation** of pretrained Lc0 chess policy networks, enabling controllable, human-like gameplay while preserving the base policy (Section 6.5.1).

2. An extension, UniMaia-Aux, that incorporates auxiliary temporal conditioning and auxiliary behavioral prediction objectives to improve policy calibration and behavioral modeling (Section 6.6).

3. A systematic empirical study of conditioning mechanisms for policy modulation, including ControlNet-style conditioning, fusion strategies, auxiliary objectives, and parameter-efficient fine-tuning methods (Section 6.3).

4. An empirical analysis of prompt-conditioned behavior, including sensitivity to prompt structure, controllable opening selection, temporal conditioning effects, and variation in model predictions across player skill levels (Sections 8.1 and 7.3).

5. A large-scale, metadata-augmented version of the Lichess dataset, along with a reproducible pipeline for generating natural language prompt templates (Chapter 5).

6. An open-source research framework for loading, conditioning, and evaluating neural chess policies under a unified interface, enabling reproducible comparison across models and conditioning regimes.

7. A suite of evaluation benchmarks spanning **prompt-conditioned** and **metadata-conditioned** settings, enabling systematic comparison of controllability, auxiliary behavioral modeling, and human move prediction performance in chess (Section 7.1).

Although evaluated in chess, the proposed framework generalizes to structured decision-making domains where strong inductive biases coexist with the need for high-level semantic control.

Chapter 2

Background

This chapter introduces the key technical background needed to understand UniMaia, including chess notation, rating systems, and the architecture of Leela Chess Zero (Lc0). Many papers that use Lc0 cite its homepage rather than a formal technical report. Recent work such as Maia-1, which is a fine-tuned Lc0-CNN model, demonstrated that neural policies can model human move distributions conditioned on rating. UniMaia itself combines an Lc0-CF transformer with an LLM to provide semantic control.

2.1 Chess

2.1.1 Brief Overview of Chess

Chess is a two-player, zero-sum board game played on an 8-by-8 board with alternating light and dark-coloured squares [33]. Players take turns to make moves, starting with the player with the white pieces, followed by the player with black pieces [33]. A single player's move is called a *ply* (or half-move), while a full move consists of one move by White and one by Black. In this zero-sum game, players can either win, lose, or draw [33]. A player wins if they "checkmate" the enemy king, if their opponent resigns, or if their opponent runs out of time on the clock [33]. A draw is forced if there is "stalemate", insufficient material on the board to deliver checkmate, five-fold repetition, or 75 reversible chess moves have been made [33]. A draw can be claimed if a player accepts a draw offered by the opposing player, if there is three-fold repetition, or if 50 reversible chess moves have been made [33].

On Lichess, an online chess website, the game is automatically drawn upon meeting the 50-move rule [10], and Lichess users can configure their settings to automatically claim a draw upon threefold repetition.

2.1.2 Chess Notation

Chess positions are represented either by the positions of each chess piece or the list of moves that the two players made leading up to the current position. The Forsyth–Edwards Notation (FEN) encodes the current position as a string [31].

Chess moves can either be encoded with the Standard Algebraic Notation (SAN) or for the Universal Chess Interface (UCI). Moves encoded with Standard Algebraic Notation (SAN) are meant to be read by people and encode (1) the piece type, (2) the destination square, and (3) any disambiguating information, if necessary [32]. For example, *Nf3* indicates that a knight moved to the *f3* square. On the other hand, moves encoded for the Universal Chess Interface are meant to be read by chess engines and only encode the starting and ending squares, and what piece a pawn was promoted to, if applicable [44]. For example, if the white knight moved to *f3* from the starting position, the Universal Chess Interface (UCI) move would be encoded as *g1f3*.

In Portable Game Notation (PGN), metadata about the chess game is included above the list of moves in Standard Algebraic Notation, possibly followed by the result of that game [23]. In Portable Game Notation (PGN), the moves in the movelist are separated by spaces, with the full move count appearing before each white player's move, like this: *1. e4 e5 2. Nf3*. The public dataset of Lichess games [102] is stored as compressed PGN files, with a single file per month. Each game PGN is concatenated together, separated by newlines.

2.1.3 Rating System

In chess, a numerical score called a "rating" is used to track the strength of a player, given their performance on past games. Two rating systems are typically used: the Elo rating system and the Glicko rating system [98].

In the Elo rating system, new players are initially assigned a fixed numerical value as their rating. Whenever two chess players play a game, both of their ratings are updated depending on the outcome of the game, whether it is a win, loss, or a draw [24].

The Glicko rating system enhances the Elo rating system with a rating deviation score that is used to determine how uncertain the rating is, which is lower when more games are played and is higher when a player is inactive or is a new player [35]. Glicko-2 builds on Glicko by adding rating volatility, which measures how much the player's rating is expected to change under normal play [36].

Lichess uses the Glicko-2 rating system but awards more points to players with the black pieces [98]. Although Lichess uses the Glicko-2 system, ratings are still referred to as "Elo" in the PGN headers in their open games database [102], following common parlance. Depending on the time control, Lichess games are classified as Ultrabullet, Bullet, Blitz, Rapid, Classical, or Correspondence games [104]. Players have a unique rating for each time control. New Lichess players start with a rating of 1500 [98]. The minimum rating was historically 600 but was lowered to 400 in March 2023, and the maximum possible rating is 4000 [29]. As of early 2026, ratings span approximately 400–3400, with a median rapid rating near 1400 and a heavy concentration in the 900–1900 range [97, 103].

While the Glicko-2 rating system is used by Lichess, we refer to player ratings as "Elo" throughout for simplicity.

2.2 LeelaChess Zero

AlphaZero is a neural network that was trained to play several board games like Go, Shogi, and chess through self-play reinforcement learning [86]. It uses Monte Carlo Tree Search (MCTS) with the value head to evaluate the long-term strength of playing each move. The learned policy is trained to match the MCTS visit counts and the value predicts the game outcome. The Q-value, or action-value, encodes the strength of each intermediate node in the search tree. Leela Chess Zero is an open reproduction of AlphaZero [86], specifically tailored to chess. Although training is centralized, the creation of self-play games is crowdsourced.

The Lc0 models released earlier are Squeeze-Excitation convolutional neural networks (CNNs) [43] with the same dimensions as the model used in AlphaZero. Models released later use a transformer backbone [110] following the ChessFormer (CF) architecture [67, 68] and are stronger than the CNN models.

2.2.1 Input Encoding

In Lc0, the input consists of an encoding of the last 8 chess board positions, along with auxiliary metadata describing the current state. Specifically, the representation comprises 112 feature planes for each of the 64 board squares [100]. The first $13 \times 8 = 104$ planes encode the 8 most recent board positions, ordered from most recent to oldest [100].

For each position, the first six planes encode the locations of the white player's pawns, knights, bishops, rooks, queens, and king, respectively [100]. The next six planes encode the positions of the black player's pieces in the same order [100]. The thirteenth plane indicates whether the position has occurred previously, taking value 1 if a repetition is present and 0 otherwise [100]. If fewer than 8 positions are available, the missing history is padded with all-zero planes [100].

The remaining 8 auxiliary planes assign a single value to all squares within each plane (with different planes encoding different values) [100]. These encode: white queenside castling rights, white kingside castling rights, black queenside castling rights, black kingside castling rights, whether it is black's turn to move, the halfmove clock normalized by 99, a deprecated all-zero plane, and an all-ones plane that helps the network identify board boundaries [100].

Like AlphaZero [86], Lc0 encodes a fixed history of 8 positions, but uses 112 planes instead of 119 [100]. In particular, it uses a single repetition plane per position rather than two [100]. This encoding is sufficient to detect threefold repetition, but does not distinguish higher-order repetitions (e.g., fivefold repetition). In practice, this limitation has negligible impact, as such events are extremely rare in real games (see Section 6.1.4 and Appendix A).

Following AlphaZero, Lc0 orients the boards to be from the perspective of the current player [100, 86].

2.2.2 Output Encoding

There are 1,858 legal UCI moves that can be predicted by Lc0's policy head, which covers both ordinary moves and pawn promotions as distinct tokens [101]. Instead of introducing a new token for knight promotions, regular moves from the seventh to eighth rank are treated as knight promotions by default if the moved piece was a pawn [101]. These cover all of the possible legal moves that can be made by any piece from the perspective of the current player, thus omitting the opponent's pawn promotions.

Unlike AlphaZero, which outputs a scalar value head [86], most newer Lc0 models output a 3-dimensional vector encoding the probability of a win, draw, or loss. Instead of using the scalar output of the value head for the leaf nodes in MCTS, Lc0 uses the net win probability $\mathbb{P}(w) - \mathbb{P}(l)$.

The Lc0 models also output a moves-left head that predicts the expected number of plies remaining in the game [30]. In value-only formulations without such a signal, positions that are evaluated as decisively winning or losing can lead to degenerate behaviour, where the search becomes indifferent among many equivalent continuations and may shuffle moves until a draw occurs. The moves-left head mitigates this issue by guiding MCTS: when the absolute value of the best child's Q-value exceeds a threshold, it provides a bonus to moves that lead to shorter wins or longer losses, thereby encouraging decisive play [30].

Chapter 3

Related Work

This chapter reviews prior work on chess engines, human move prediction, prompt-conditioned policies, and multimodal conditioning methods for neural networks.

We organize this literature through the lens of **conditioning mechanisms**: traditional engines and policy networks rely on structured representations of game state, metadata-conditioned models explicitly incorporate player and game attributes, and prompt-conditioned models use natural language as a flexible control interface. This framing highlights the trade-offs between inductive bias, controllability, and predictive performance that motivate the approach proposed in this thesis.

3.1 Chess Engines

Since the 1980s, chess has been used as a testbed for developing artificial intelligence. In 1985, Hsu began work on ChipTest [41], the predecessor to Deep Thought and later Deep Blue. Deep Blue defeated the reigning world chess champion at the time, Garry Kasparov, in 1997 by winning two games and drawing three, scoring 3.5 points against Kasparov's 2.5 and thus winning the match [11]. Deep Blue ran on a specialized hardware designed for searching through chess move trees. The system used three processes, or "layers," to search for desirable leaf nodes during play.

Since Deep Blue, engines such as Stockfish have risen to prominence, using a traditional move-tree search to play chess [105]. These engines perform a minimax search over the game tree with alpha-beta pruning, which allows the search to skip branches that cannot produce optimal values [51]. Heuristic evaluation functions are used to estimate the values of the

leaf nodes. At the time, alpha-beta search was widely believed to be the superior approach for chess engines, as Stockfish was the 2016 Top Chess Engine Competition (TCEC) world chess engine champion [86].

After deep learning achieved several breakthroughs in computer vision [52] and playing Atari games [66], deep neural networks began to impact other two-player board games, such as Go. AlphaGo was the first program to defeat professional human Go players without handicap [85]. It used a convolutional neural network pre-trained on a dataset of human games and further improved through self-play and MCTS-based reinforcement learning [85].

AlphaGo was succeeded by AlphaGo Zero [87] and later AlphaZero [86], both of which were trained exclusively from scratch using self-play and MCTS-based reinforcement learning. AlphaZero was generalized such that one instance could be specialized to play chess and another instance to play shogi. The AlphaZero chess model defeated Stockfish 8, the strongest chess engine at the time, in a 1,000-game match with 155 wins, 6 losses, and 839 draws. Although the official weights for any of AlphaZero's board-game models were never released, Lc0 is an open-source project that reproduced the AlphaZero model for chess [96]. The project later developed its own chess transformer architecture to increase playing strength [67, 68].

After its loss to AlphaZero, Stockfish adopted the use of Efficiently Updatable Neural Networks (NNUEs) for chess, starting with version 12 [106]. NNUE is a neural network architecture designed for CPU-only engines and was originally developed for shogi [69]. Stockfish's NNUE networks are trained on datasets derived from the Leela Chess Zero (Lc0) training data, according to the Stockfish training documentation [118, 105]. Ruoss et al. later distilled Stockfish's policy into a transformer trained on the board states encoded as text, using FEN notation [81].

While these systems provide highly effective structured policy representations, they are optimized for playing the strongest moves rather than predicting human decisions or supporting semantic control. UniMaia builds upon Lc0's strong pretrained representations while introducing mechanisms for controllable human-like play.

3.2 Metadata-Conditioned Human Move Prediction

While traditional chess engines focus on optimal play, a separate line of work aims to model human decision-making. These approaches typically condition on structured metadata, such as player skill level, to predict human moves.

Although previous chess engines focused on playing chess well, most engines prior to Maia Chess were unable to emulate human playing styles [65]. Maia Chess is a fine-tuned version of Lc0 trained on human Lichess games played at different Elo rating levels [65]. It was able to predict human moves more accurately than previous models [65]. Using a family of these models, the researchers were also able to predict which blunders were commonly made by newer players and at what skill levels those mistakes disappeared [65].

Jacob et al. found that using KL-regularized MCTS with a Maia Chess engine improved its ability to predict the top human chess move, particularly for stronger players [46]. In the open-source community, several approaches have been explored to produce human-like chess bots, such as using random sampling instead of greedy sampling for Maia Chess [18], or training stronger Maia-style networks with different Lichess datasets [107].

While Maia produced a family of models for different skill levels, follow-up work trained a single unified model capable of playing across a range of skill levels. Tang et al. introduced Maia-2, which trains a single model to imitate players across multiple skill levels using binned Elo embeddings [91]. Zhang et al. later proposed Allie, which instead uses interpolated skill embeddings to represent arbitrary skill levels [127], as does Maia-3 [68].

Allie could also play games with different time controls and predict how long players would take to make a move [127].

Maia4All adapts a Maia-2 model to learn the style of an individual player from as few as 20 games [92].

Another line of work models chess games directly as language sequences. Several recent approaches have attempted to imitate human play by training large language models to generate PGNs of Lichess games without additional metadata [50, 124, 16]. Allie is also based on an LLM initialized with GPT-2 weights and trained to model sequences of chess moves [127]. These LLM-based approaches typically rely on specialized chess tokenizers.

These approaches demonstrate that conditioning on structured metadata substantially improves human move prediction. However, they remain limited to predefined attributes such as Elo, time control, or player identity, making it difficult to express richer behavioral concepts or novel combinations through a flexible interface. UniMaia replaces these discrete conditioning variables with natural language while preserving the strong inductive biases of structured chess policies.

3.3 Prompt-Conditioned Policies

Prompt-conditioned models use natural language as a flexible interface for controlling behavior across a wide range of tasks. Over the years, many different approaches have been explored to address this problem.

One early direction involved applying machine learning to text-based games. Some approaches used information retrieval [109], others used reinforcement learning [17], and some trained separate policy and language models [26]. After the release of ChatGPT [71], many researchers began using LLMs to both communicate and act within environments, whether through text [119, 84], graphical user interfaces [39, 121], real-world robotics [20, 34, 63], or open-world video games [112, 77, 9, 90, 64].

Specifically for chess, Carlini found that GPT-3.5-turbo-instruct was able to play chess more successfully than GPT-4's chat models [12]. Feng et al. introduced ChessGPT, a prompt-conditioned model capable of both answering questions about chess and generating human-like gameplay [27]. They also introduced ChessCLIP, a model for jointly representing chess prompts and board states [27].

Existing prompt-conditioned chess systems typically generate moves directly as text or rely on general-purpose language models, sacrificing the domain-specific inductive biases that make specialized chess policies highly effective. UniMaia instead uses language to modulate a pretrained chess policy, combining semantic controllability with the accuracy of a dedicated policy network.

3.4 Multimodal Conditioning Methods

Early vision–language models used cross-attention between each text decoder layer and latent image representations [3, 54]. More recently, projecting vision representations as text tokens for the language model has become a mainstream approach [59, 7, 94].

Image generation models typically use cross-attention between prompt embeddings and image representations to generate images [80, 74, 25]. ControlNet allows users to specify additional conditioning signals, such as edge maps, when generating images using text-to-image diffusion models [125]. This is achieved by training a copy of the base model whose outputs are conditioned on both the control signal and the layer-wise outputs of the original model through zero-initialized convolutions [125].

Although several subject-conditioned text-to-image models selectively train portions of the model to support multiple conditioning modalities [120, 57], others follow a ControlNet-like approach to generate personalized images [116].

Taken together, these lines of work highlight a trade-off between structured inductive biases, controllability, and flexibility. Metadata-conditioned models achieve strong performance on human move prediction, while prompt-conditioned models offer greater flexibility but often lack domain grounding. This thesis explores a hybrid approach that combines these strengths through prompt-conditioned modulation of a structured policy network.

Chapter 4

Problem Formulation

Our goal is to learn a prompt-conditioned mechanism for move selection while leveraging the representations learned during chess pretraining. Specifically, given a board state s and a conditioning prompt p, we aim to learn a conditioned policy $\pi(a \mid s, p)$ that predicts the next move a. Example prompts include *"Play chess as White, opening with the Alapin Sicilian"* or *"Play like a 1600-rated blitz player"*. During inference, this prompt serves as a persistent system instruction that remains fixed throughout the game, while the board state evolves after each move. For models with auxiliary temporal conditioning, a dynamic suffix containing information such as the remaining clock time and previous move delay is updated after every ply, while the original prompt remains unchanged. We investigate how well the model's behavior aligns with the intended prompt, how effectively it can be steered across different aspects of play, whether any emergent behaviors arise during training, and how well it generalizes to out-of-distribution positions.

The strongest neural-network-based chess engines are derived from the Leela Chess Zero (Lc0) engine, which we use as the base policy for our model. Accordingly, the input follows the same format: the eight most recent board states (including the current position), castling rights for both sides, the active player, and the number of moves until an automatic draw due to the 50-move rule. Boards are oriented toward the current player, and actions are represented as UCI move tokens.

We do not use Stockfish's NNUE network, as it is designed for efficient position evaluation within a search-based engine rather than for strong standalone representation learning. While Stockfish achieves high playing strength by evaluating many positions efficiently, our focus is on leveraging a strong learned policy representation, making Lc0 a more suitable foundation for this work.

Chapter 5

Dataset Construction

5.1 Data Sources

All of the training data was derived from the Lichess Open Database [102]. The database consists of chess games encoded as PGN strings, where each game contains a header with metadata, a list of moves played in the game, and, if the game ended, the result of the game. The PGNs for each game played in the same month are concatenated together into a single file, delimited by newlines, and compressed. For the Lichess Open Dataset, we used all games from 2013 to 2023, inclusive, for a total of 5.185 billion games.

The following steps were performed to create the datasets used to train the model:

1. The raw Lichess games were processed into a Parquet dataset that was organized and could be easily filtered by metadata (LichessGames).
2. The games in LichessGames were then formatted using one of 160 randomly selected templates to create LichessTemplates-Pretrain for metadata-rich pretraining.
3. The games in LichessGames were also formatted using one of 2,048 randomly selected templates to create LichessTemplates-Instruct for instruction fine-tuning.

During training, each example was formatted using a template from either LichessTemplates-Pretrain or LichessTemplates-Instruct with equal probability.

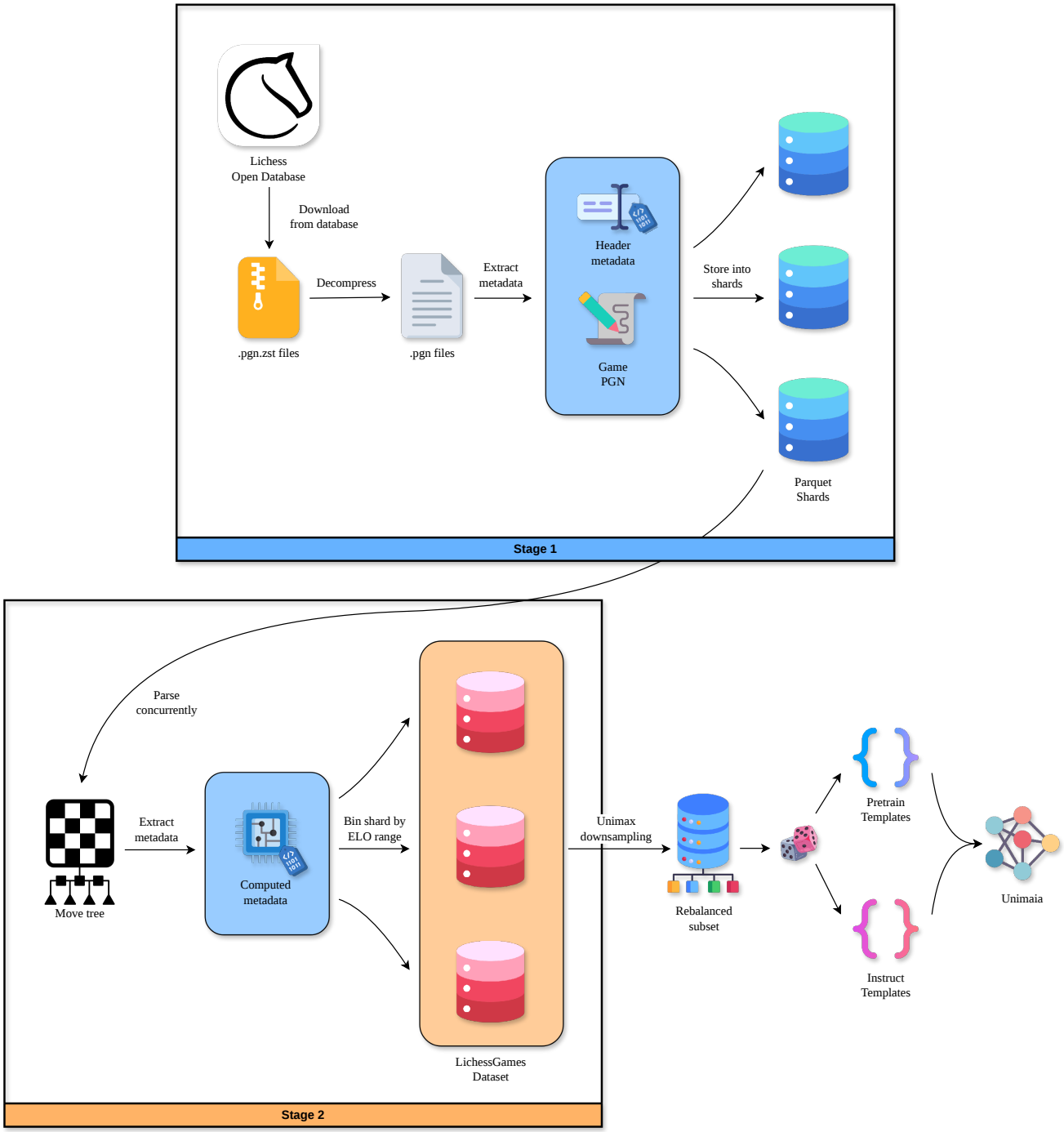

Figure 5.1: Data processing occurs in two stages: the first performs lightweight string-based parsing of the PGNs, while the second concurrently parses them into move trees. For training UniMaia, the LichessGames dataset is downsampled and formatted using random prompt templates.

Statistic	Value
Total games	5.185×10^9
Total positions	2.65197×10^{11}
Years covered	2013–2023
Source	Lichess Open Database
License	CC0
ELO range	600–$\approx$ 3400
Templates (pretrain)	160
Templates (instruct)	2048

Table 5.1: Summary statistics of the LichessGames dataset.

5.2 Creating LichessGames

As shown in Figure 5.1, the data processing used to create LichessGames is performed in two stages. The first stage uses string-only operations to quickly organize the PGN data into a series of Parquet files containing metadata. The second stage performs more in-depth processing, extracting additional game-related information and splitting the Parquet files by the players' Elo ranges.

The PGN games were parsed using the `python-chess` [28] package on a SLURM cluster. `python-chess` provides two functions for reading chess PGNs: a cheaper function, `chess.pgn.read_header`, which extracts the PGN metadata, and a more expensive function, `chess.pgn.read_game`, which fully parses the PGNs into move trees. Both functions advance the file pointer to the next PGN. To enable concurrent calls to `chess.pgn.read_game` in the second stage, we rewind the file pointer after calling `chess.pgn.read_header` in the first stage to recover the full PGN string.

Splitting the games by Elo range ensures that games from all skill levels can be sampled during training. Games with very low or very high Elo ratings are rarely observed when sampling uniformly from the dataset.

Stage 1: String-only processing

The first stage does not parse the list of PGN moves and instead extracts only the metadata from the PGN header and the PGN itself, storing them in a Parquet file. Because this stage relies only on string operations, the processing can be performed quickly, even on a single thread.

A single thread was used because all games played in the same month are stored in a single text file and concatenated sequentially. To reduce I/O latency, updates to the Parquet file are performed in batches of 1,024 games. A maximum of 4,096 batches (4,194,304 games) are stored in each Parquet file to prevent out-of-memory errors when loading the file into memory.

To concurrently process PGNs in Stage 2, the number of bytes in the game PGN was tracked so that the file pointer could be rewound and the original PGN extracted. Because Unicode characters may occupy more than one byte, the number of bytes requested must be adjusted accordingly to recover the correct PGN, as shown in Listing 5.1. Opening the file as UTF-8 text or as raw bytes alone does not resolve this issue.

```
diff = len(pgn.encode("utf-8")) - len(pgn)
f.seek(prev_offset)
pgn = f.read(offset - prev_offset - diff)
```

Listing 5.1: Python pseudocode for handling PGN encoding differences

The fields that were extracted from the game metadata are described in Table 5.2.

Stage 2: Game processing

In this stage, the PGN games are parsed into move trees and additional game-level information is extracted. Because the games have already been separated into an array, it is possible and beneficial to use concurrency, as move-tree parsing is the primary CPU-bound bottleneck.

We also extract additional information used in later stages of the pipeline, such as the length of the game and more detailed termination information. The fields extracted in Stage 2 are described in Table 5.3.

The other task performed in Stage 2 is splitting the shards into 100-point bins based on the mean Elo and the Elo difference between players, allowing the dataset to be rebalanced by skill level during training. This partitioning step is performed during Stage 2 rather than Stage 1 because it is simpler to implement efficiently.

Field	Description
Event	The event name
TimeControlName	The time control, if provided
UTCDate	The date that the game was played
Year	The year that the game was played
Month	The month that the game was played
Day	The day that the game was played
TcDelay	The time limit per player for the game, in seconds, if provided
TcIncrement	The time limit increase given to each player after making a move, in seconds, if provided
WhiteElo	The Lichess Elo rating of the white player, if provided
BlackElo	The Lichess Elo rating of the black player, if provided
WhiteRatingDiff	The rating change of the white player after the conclusion of the game
BlackRatingDiff	The rating change of the black player after the conclusion of the game
Round	The round of the tournament, if applicable
Result	The game result
Variant	The variant of chess
ECO	The chess opening code, as described by the ECO
Opening	The name of the chess opening
Termination	Some rough information about how the game ended, such as whether it was through the normal rules of chess, due to a time forfeiture, or because of rule-breaking
Annotator	Who annotated the PGN (usually lichess.org)
IsFischerRandom	Whether the game is a Fischer random chess game
EvalDepth	The Stockfish evaluation depth of the first comment (if available) in the PGN. This can be used to help identify if Stockfish was used to analyze the game
OriginalPGN	The original PGN

Table 5.2: The fields extracted from the Lichess PGN games in Stage 1.

Field	Description
Length	The length of the game, in halfmoves
TerminationOutcome	If the game ended through the rules of chess, how the game ended
TerminationWinner	The winning player
LastComment	The last comment in the game, which occasionally contains information about how the game ended
TerminationReason	The way the game ended. Combines information from the Termination, TerminationOutcome and Result fields.
MeanElo	The mean player Elo
DiffElo	The absolute difference between the players' Elos

Table 5.3: The fields extracted from the Lichess PGN games in Stage 2.

5.3 Creating the Prompt Templates

To ensure that the model can generate a wide variety of chess games, we use synthetic prompt templates that were manually reviewed and edited. Given the scale of the dataset, it is infeasible to obtain human annotations for each game or to prompt a model to generate a unique prompt for every example. Instead, we use prompt templates, a strategy previously applied in training LLMs for low-resource languages [88].

We accept a reduction in surface-level diversity because we use parameter-efficient fine-tuning, which is less susceptible to overfitting.

To generate the synthetic templates, we prompt an LLM (`gpt-4-turbo-2024-04-09` [2]) to produce natural-language prompts that describe or request a chess game using game metadata as input. For example, the opening "Sicilian Defense: Alapin Variation" may appear as "the Alapin variation of the Sicilian Defense." These prompts are then converted into Jinja2 templates by replacing metadata spans with programmatically generated snippets that can be instantiated for arbitrary games.

Template construction is performed iteratively. In each pass, a script replaces all currently supported metadata spans with Jinja2 fields. The resulting prompts are then reviewed to identify remaining metadata references, and additional snippet-generation rules are implemented as needed. This process yields the final template sets used for pretraining and instruction fine-tuning.

A key challenge in this pipeline is maintaining correctness under iterative modification. Because templates combine multiple independently generated metadata snippets, even small changes to snippet logic or template structure can introduce subtle grammatical

or formatting errors across many instantiations. Ensuring correctness therefore requires re-validating the fully instantiated templates after each modification, which becomes increasingly costly as the template space grows.[1]

As a result, some instantiated templates contain minor grammatical artifacts (e.g., duplicated words such as "player player"). These artifacts are retained in both training and evaluation. Empirically, they do not appear to materially affect model behavior, and preserving them ensures full reproducibility of the data pipeline.

5.3.1 Creating LichessTemplates-Pretrain

To generate the synthetic prompts, we generate two prompts for each combination of the following configurations:

1. 5 random Lichess games, with players of varying skill levels and a mix of titled and untitled players. The chess metadata that isn't overwritten by the other settings originate from these seed games.
2. Four different time controls: bullet, blitz, rapid, and classical
3. Two different openings, one common and one uncommon
4. Two different verbosity settings: concise and regular

This results in $2 \times 5 \times 4 \times 2 \times 2 = 160$ prompts. These prompt templates retain all metadata so that the model can learn associations between each piece of metadata and the subsequent move.

5.3.2 Creating LichessTemplates-Instruct

To create the synthetic instruction prompts, we once again generate two prompts for each combination of the following configurations:

[1] Substantial effort was invested in designing metadata formatting rules that produce well-formed text across edge cases (e.g., correct pluralization, handling zero vs. nonzero increments, and decomposing time controls into hours, minutes, and seconds). Early versions of the templates produced consistently well-formed English. However, as additional metadata fields and variations were introduced, the combinatorial growth in template instantiations made it difficult to guarantee that all edge cases remained correct under subsequent modifications.

1. Two random Lichess games between titled players, one with a common opening, and one with an uncommon opening
2. Whether to include the players' titles in the prompt
3. Whether to include the players' Elos in the prompt
4. Whether to include a textual description of the player's skill level in the prompt
5. Whether to include the players' usernames in the prompt
6. Whether to include the players' real names (if available) in the prompt
7. Whether the current player is black or white. Although most prompts can be recycled as being for white or for black, passing in a slightly different prompt helps ensure that the output is more original
8. Whether to include the name of the opening
9. Whether to include the opening moves
10. Whether to format the prompt as a request to play this game or as a description of the players.

This results in $2 \times 2^{10} = 2,048$ prompts. Because these prompt templates omit some game metadata, they help the model learn to generate games even when metadata is partially or entirely absent.

5.4 Metadata Preprocessing for Prompt Templates

To ensure that metadata snippets can be generated for any chess game, we supplement the data in the PGN headers with additional information, such as the real names of well-known chess players. While some snippets are straightforward to implement, such as formatting the time control as text with correct pluralization, extracting the official opening move sequence for a game requires considerably more effort.

Additional auxiliary metadata sources are stored separately and joined dynamically when constructing training examples or prompt-conditioned benchmarks.[2] Table 5.4 summarizes the auxiliary metadata sources used in this work.

[2]The version of the openings database used in this work contains 3,464 openings, although newer releases contain additional entries.

Table 5.4: Auxiliary metadata sources used during training-example and benchmark construction. These sources are stored separately from LichessGames and joined dynamically as needed.

Dataset	Purpose	Entries	License
Historical Peak Elo and Identity Data for Notable Lichess Users	Player name augmentation	5,793	CC0
Lichess Chess Openings Database	Opening normalization and canonical move sequences	3,464	CC0

5.4.1 Enriching Player Metadata with Real Names

The data used for player identity mapping was obtained from a publicly distributed, anonymized file (`famous_lichess.tsv`) shared on an online chess community forum. Because the original hosting platform and author are no longer active or identifiable, the dataset has been independently preserved and hosted for replication purposes [108].

5.4.2 Obtaining Opening Move Sequences

Two main pieces of information in the PGN header are used to identify the opening move sequence for most openings: (1) the ECO code and (2) the opening name. In some cases, the PGN move history is also used to disambiguate between variations that share the same ECO code and legacy opening name. This is handled via rule-based heuristics after considering the ECO code and opening name alone.

Our goal is to recover the exact sequence of moves that are officially considered part of the opening, rather than arbitrarily taking the first n moves from the PGN, and to make this sequence available in the prompt. Because Lichess has changed its opening classifications and naming conventions over time, additional processing is required to recover the canonical move sequence for each opening. The official Lichess Chess Openings repository [99] provides the reference data used by Lichess.

To recover the move sequences, we perform the following steps:

1. Normalize the ECO code and opening name for each opening across the dataset.
2. Look up the best match in the Lichess chess openings database.

To identify how to resolve the missing openings, a script was used to collect all of the openings in LichessGames whose ECO code and name were missing from the Lichess chess openings database. To allow for using fuzzy string matching algorithms to compare both values, they are first concatenated together using the tab character as a delimiter. Using the `thefuzz` package [83], the openings with the three highest similarity scores were returned for each opening.

To resolve missing openings, we first collect all entries in LichessGames whose ECO code and opening name do not appear in the Lichess chess openings database. For fuzzy string matching, these two fields are concatenated using a tab character as a delimiter. Using the `thefuzz` package [83], we retrieve the three highest-scoring matches for each query.

The `thefuzz` library is a convenience wrapper around `rapidfuzz` [6], which provides several similarity metrics. The Levenshtein distance counts the number of insertions, deletions, and substitutions required to transform one string s_1 into another s_2 [55]. This distance can be normalized by dividing by the maximum possible distance, $\max(|s_1|, |s_2|)$ [6]. The InDel distance, as defined in `rapidfuzz`, is similar but does not account for substitutions. Its normalized form divides the distance by $|s_1| + |s_2|$. The token sort ratio is computed by sorting tokens in each string, calculating the InDel distance, and scaling the result by 100 [6].

Empirically, the token sort ratio produced the best results, outperforming the normalized InDel distance (the default metric in `thefuzz`). This is because the normalized InDel distance can favor strings with high character overlap despite semantic differences. Additionally, restricting comparisons to openings with the same ECO code performed poorly, as some openings have been reassigned different ECO codes over time.

All unresolved openings were manually reviewed. If the top match was incorrect but one of the next two candidates was correct, rule-based heuristics were introduced to remap the ECO code and opening name to the current database values. If none of the top three matches were correct, we used one of the following approaches:

1. Run `git log -S "<STRING>"` on the repository to identify commits where the opening name changed, allowing us to recover the updated name and ECO code.

2. Search for the opening name online to find its move sequence, then input this sequence into the Lichess opening analysis board to determine the corresponding Lichess opening name and ECO code.

This process yields a complete mapping from legacy opening names to their current equivalents.

When formatting prompts, we retain the legacy opening name when inserting it into the template, but map it to its updated equivalent when looking up the corresponding move sequence. Because multiple move sequences of varying lengths may correspond to the same ECO code and opening name, we identify the longest common prefix of moves consistent with the PGN move history. This longest common prefix is then used as the opening move sequence in the metadata snippet.

Chapter 6

Model Architecture and Training

To determine the optimal configuration for the final training run, we ran a series of ablations on a subset of the data. Although adding out-of-domain steerability to a specialized Lc0 model is unprecedented, we configure the initial baseline with an educated best guess. We then examine how the model improves over time in the full training run.

6.1 Experimental Setup

To determine the optimal configuration for the final training run, we conduct a series of ablations on a subset of the data. All ablations are performed on Lichess games from January to December 2013, allowing for rapid iteration while preserving realistic game distributions.

6.1.1 Training Data and Procedure

Due to the scale of the dataset, we do not globally shuffle all games. Instead, each monthly partition is divided into shards, which are shuffled independently and processed sequentially. While this may introduce mild temporal data drift, it enables efficient checkpointing and incremental training without restarting from scratch.

Each shard is further subdivided into Elo buckets, according to the Elo mean and difference for the two players. We apply Unimax downsampling [15] across Elo buckets in all experiments to allocate a fixed training budget while ensuring that extreme Elo ranges remain represented. The data sampling ablations described in Section 6.3.5 are applied within each Elo bucket after this step.

To mitigate overfitting to prompt templates, we apply simple data augmentation: each prompt is lowercased with probability 0.1, and diacritics are removed with probability 0.5.

Each ablation is trained with 2^{14} plies per shard, for a total of 196,608 plies. Since the 2013 dataset contains only one shard per month, training proceeds sequentially from January to December.

To ensure hardware stability during long training runs, we insert a cooldown period after processing each month to prevent GPU thermal throttling and system crashes.

6.1.2 Optimization

Unless otherwise specified, all experiments use the following optimization setup.

We use a linear warmup schedule over the first 2,048 steps to a learning rate of 5×10^{-5}. From April 2013 to June 2014, the learning rate decreases linearly to 1.5×10^{-5} through monthly updates. It is then held constant at 1.5×10^{-5} until 2022, after which it decays monthly, approaching 5×10^{-7} by the end of training.

All models are trained with a weight decay of 0.01 and $\epsilon = 1 \times 10^{-8}$.

6.1.3 Evaluation Protocol

Model variants are compared using both training loss and downstream benchmark performance. Due to high variance in the loss curves, we report the mean loss over December 2013 ($\mu_{\text{loss, Dec 2013}}$) for each configuration.

We evaluate each model on a subset of benchmarks (see Chapter 7, Section 7.1) selected to capture complementary aspects of model behavior:

1. **Lichess Openings Benchmark (LOB–P):** Measures the model's ability to follow opening-specific instructions under partial information.

2. **Lichess Instruction-Following Diagnostic (LIF-D):** A controlled diagnostic benchmark used during ablations to evaluate instruction-following behavior under a restricted Elo range.

3. **Maia-1 Subset (M1-S):** A stratified subset of the Maia-1 benchmark used to evaluate alignment with human move choices across skill levels.

LIF-D is used only during ablations, as it overlaps with the training distribution. Final evaluation instead uses the full Lichess Instruction-Following Benchmark (LIF) defined in Chapter 7.

For each ablation, we select the best configuration based on a combination of final loss and average benchmark performance.

6.1.4 Baseline Configuration

Our initial configuration reflects an informed starting point for combining language-conditioned control with a specialized chess policy model.

We apply the opening Unimax downsampling strategy and filter out:
- bot games,
- Lichess Master games,
- correspondence games,
- games with missing Elo or time control, and
- games with fewer than 32 plies unless they end in checkmate or a forced draw.

Lichess Master is a rare honorific title, no longer awarded, that was granted at the discretion of Lichess [1]. In contrast, all other player titles are officially awarded by FIDE.

For the model, we use:
- the final layer of ChessGPT-base [27] to generate prompt embeddings and
- the Lc0-ChessFormer model [67, 68] initialized with `t3` weights as the policy backbone.

We retain the standard Lc0 input representation without modification, despite its omission of an explicit five-fold repetition plane (Section 2.2.1). This is justified by the extremely low prevalence of five-fold repetitions in practice. In a representative shard from October 2022, five-fold repetition accounts for only 0.094% of terminations, and decreases further to 0.019% in January 2023 following changes to draw-claim behavior in November 2022 (see Appendix A for full termination statistics).

We initially use the AdamW optimizer [62].

All ablations are designed to be computationally accessible and can be run on consumer-grade GPUs such as the RTX 3090 or RTX 5090.

6.2 Design Evolution and Key Improvements

To provide a high-level overview of our experimental findings, we summarize the sequence of design decisions that led to the final model in Table 6.1. Each step reflects a modification to the previous configuration, along with its impact on benchmark performance and training loss. The combined effect of these contributions is substantial, as seen in Figure 6.1.

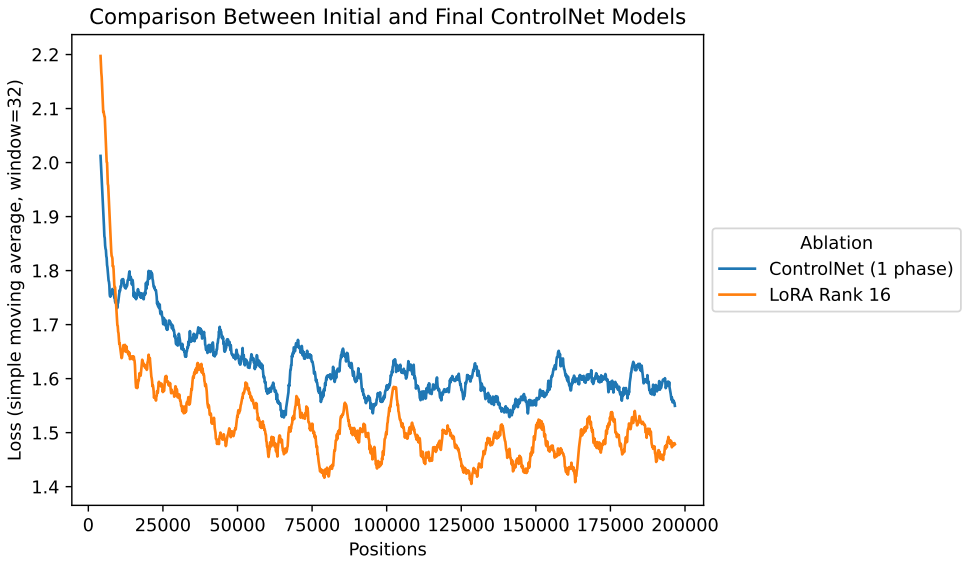

Figure 6.1: Comparison of training losses between the initial single-phase, layer-coupled ControlNet and the final model.

While most modifications improve performance, their efficiency varies considerably. In our setting, architectural and data-centric changes account for the majority of the observed gains with minimal parameter overhead. In contrast, later scaling decisions, such as adopting the BT4 backbone and increasing the LoRA rank, introduce substantial parameter increases for comparatively modest improvements. These latter changes are retained to maximize absolute performance, but exhibit clear diminishing returns in terms of parameter efficiency.

Table 6.1: Sequential improvements over the course of the ablation study. Each step builds on the previous configuration. The early-fusion, single-step cross-attention model is used as the baseline.

Aspect	**Change**	**Δ Benchmark Score**	**Δ $\mu_{\text{loss, Dec13}}$**
Base	Initial configuration	–	–
+ Architecture	ControlNet + controllable policy head	**+0.0684**	**−0.1005**
+ Optimizer	NorMuon (vs AdamW)	+0.0110	−0.0349
+ PEFT	LoRA (vs DoRA)	+0.0016	−0.0049
+ Lc0 Scaling	BT4 backbone	+0.0059	−0.0219
+ Data Sampling	Unimax (time control)	+0.0195	−0.0295
+ LoRA Config	Rank 16	+0.0017	+0.0011
Final	Full configuration	**+0.1061**	**−0.1811**

Beyond performance, several design choices also affect model parameterization. Table 6.2 summarizes the corresponding changes in LLM and Lc0 parameter counts, providing context for the efficiency trade-offs discussed above.

Table 6.2: Parameter changes associated with each design decision. Notably, the BT4 backbone accounts for the majority of the parameter increase (+279M of +404M total), while contributing only a modest improvement in benchmark score.

Aspect	**Change**	**Trainable LLM Parameters**	**Lc0 Parameters**
Base	Initial configuration	–	–
+ Architecture	ControlNet + controllable policy head	–	+124.486M
+ Optimizer	NorMuon (vs AdamW)	–	–
+ PEFT	LoRA (vs DoRA)	−0.328M	–
+ Lc0 Scaling	BT4 backbone	–	+279.038M
+ Data Sampling	Unimax (time control)	–	–
+ LoRA Config	Rank 16	+3.932M	–
Final	Full configuration	**+3.604M**	**+403.524M**

We now examine each of these design choices in isolation through controlled ablation studies.

6.3 Ablation Studies

To determine the optimal configuration, we ran a series of ablation studies. In this section, we go over each ablation study that resulted in an improvement, as well as the ones that led to negative results.

6.3.1 Architecture Design

We hypothesize that the effectiveness of language conditioning depends on where and how cross-modal interactions are introduced in the network. In particular, we study the impact of *interaction depth* by comparing early fusion, deep fusion, and ControlNet-style conditioning.

Fusion-based conditioning We first consider two standard fusion strategies:

1. **Single-step cross-attention (three-stage training, early fusion):** Inspired by early-fusion approaches in vision-language models such as LLaVA [59], this variant injects language information once before the main Transformer stack. Because the Lc0 encoder operates on a fixed set of 64 square tokens, we perform a single cross-attention operation between the prompt embeddings and the shallow board representation. The resulting fused representation is then processed by all 15 Lc0 Transformer layers and the output heads.

   For stability, we adopt a three-stage training procedure:

   (i) both the text encoder and the Lc0 encoder are frozen,

   (ii) the text encoder is fine-tuned using DoRA,

   (iii) all non-policy components are frozen while the policy outputs are fine-tuned.

   Each stage is trained on disjoint month ranges (Jan–Mar, Apr–Jun, Jul–Dec).

2. **Per-layer cross-attention (deep fusion):** This variant introduces cross-attention at every Transformer layer, enabling iterative refinement of the board representation conditioned on the prompt. To stabilize training, cross-attention output projections are zero-initialized, and the base Lc0 parameters are frozen.

ControlNet-based conditioning We next evaluate structured conditioning via ControlNet, which introduces a parallel, trainable pathway:

3. **ControlNet (two-stage training, layer-coupled):** This variant follows the ControlNet architecture [125], adapted to the Lc0 setting. A trainable copy of the encoder is added in parallel to the frozen base model, with its outputs injected into the base network after each encoder layer. The final hidden representation is mapped through a cloned policy embedding and policy head to produce a policy residual, which is added to the base policy logits.

   Cross-attention is inserted between the self-attention and feed-forward networks within each layer of the cloned encoder, connecting the hidden board representation and the prompt embeddings. The output projections of these cross-attention modules are zero-initialized for training stability.

   All ControlNet contributions (both layer-wise injections and the final policy residual) are passed through separate zero-initialized projection layers, following the ControlNet-Transformer design used in PixArt-δ [13]. Unlike the original ControlNet-Transformer, both the base layer and the ControlNet layer at depth n take as input the hidden representation from the base layer at depth $n-1$.

   The text encoder is frozen during an initial training stage (Jan–Apr) and subsequently fine-tuned using a DoRA adapter [61] in a second stage (May–Dec).

4. **ControlNet (single-stage training, layer-coupled):** Identical to the above, but the text encoder is trained with DoRA throughout.

5. **ControlNet with a controllable policy head (CPH) (single-stage training, layer-coupled):** We insert a pre-normalized cross-attention block between the cloned policy embedding and the cloned policy head. Given a policy embedding x, we apply LayerNorm and cross-attention over the prompt embeddings to obtain an update y, which is added residually before the cloned policy head: $x \leftarrow x + y$. This allows direct modulation of the action distribution.

6. **ControlNet with controllable policy head and gated attention (single-stage training, layer-coupled):** Extends the previous variant with gated attention [76] to mitigate attention sinks.

7. **ControlNet with controllable policy head (single-stage, sequential):** We also evaluate a variant that follows the original ControlNet connectivity, where the ControlNet layer at depth 0 takes the same input as the base layer, and subsequent ControlNet layers operate sequentially on the hidden representations of the preceding ControlNet layer.

Training regime All architecture ablations are conducted with the base Lc0 parameters frozen, and only newly introduced components are trained. This choice was initially made to stabilize training, as early experiments with full joint optimization exhibited large loss spikes.

Although these stability issues were later mitigated, we retain the frozen-base setting for consistency and to isolate the effect of the conditioning mechanisms.

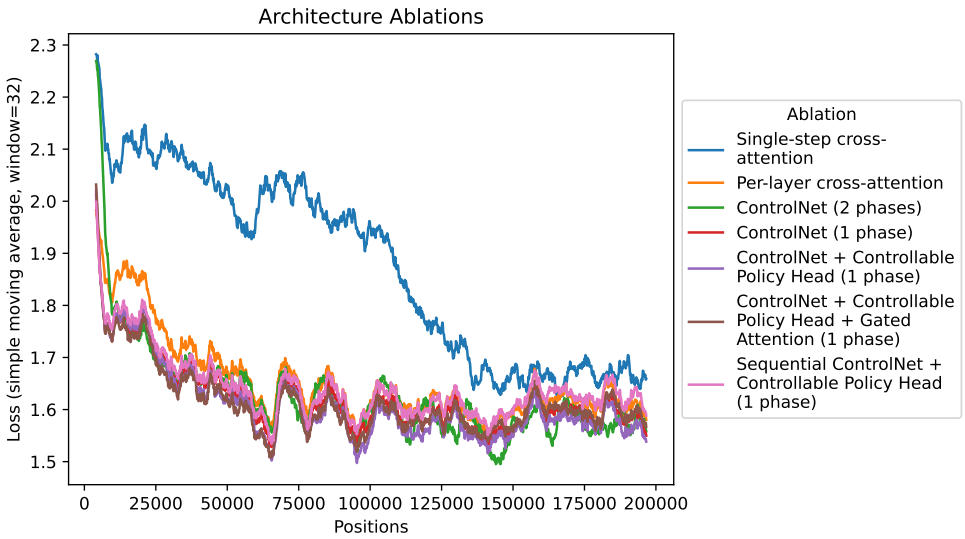

Figure 6.2: The simple moving averages of the training losses for the architecture ablations. Due to the small batch size (8), the raw losses are highly volatile, typically fluctuating between approximately 1.2 and 1.9.

Results Figure 6.2 shows the training losses for all variants. Fusion-based methods exhibit consistently higher losses than ControlNet-based approaches, particularly early in training.

Table 6.3 summarizes benchmark performance.

Table 6.3: Architecture ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Single-step cross-attention (early fusion)	0.4696	0.4176	0.4315	0.4396	1.6685
Per-layer cross-attention (deep fusion)	**0.5745**	0.4325	0.3379	0.4483	1.6092
ControlNet (2 phases)	0.5384	0.4332	0.5118	0.4944	<u>1.5753</u>
ControlNet	0.5469	<u>0.4351</u>	0.5122	0.4981	1.5865
ControlNet + controllable policy head	<u>0.5704</u>	**0.4358**	<u>0.5179</u>	**0.5080**	**1.5680**
ControlNet + CPH + gated attention	0.5059	0.4253	**0.5228**	0.4847	1.5942
Sequential ControlNet + CPH	0.5653	0.4311	0.5116	<u>0.5027</u>	1.6223

ControlNet-based variants outperform both early and deep fusion in terms of loss and mean benchmark performance. While deep fusion achieves the highest score on the openings benchmark (LOB-P), it performs substantially worse on M1-S, indicating weaker alignment with human move distributions.

In contrast, ControlNet variants achieve more consistent performance across benchmarks. This suggests that increasing the frequency of cross-modal interaction alone is less effective than introducing a structured conditioning pathway in the frozen-base setting.

Adding a controllable policy head further improves performance, consistent with its role in directly modulating the output distribution. Gated attention improves performance on M1-S but degrades instruction-following performance. The sequential ControlNet variant performs worse than the layer-coupled design and yields higher training loss, although it still outperforms the gated attention variant overall.

Overall, the ControlNet variant with a DoRA-adapted text encoder and a controllable policy head achieves the best trade-off between loss and benchmark performance in the frozen-base regime, and is therefore selected as the baseline for subsequent experiments.

Full fine-tuning (retrospective study). We conduct a limited follow-up study in which selected fusion architectures are trained with full joint fine-tuning (Table 6.4).

Under this regime, both early and deep fusion improve substantially, particularly on M1-S, indicating improved alignment with human play. Deep fusion with a controllable policy head achieves the strongest overall performance.

Table 6.4: Full fine-tuning results for selected fusion architectures. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Early fusion (frozen)	0.4696	0.4176	0.4315	0.4396	1.6685
Early fusion (full FT)	0.5393	0.4294	0.5094	0.4927	1.6329
Deep fusion (frozen)	<u>0.5745</u>	0.4325	0.3379	0.4483	**1.6092**
Deep fusion (full FT)	0.5742	**0.4400**	<u>0.5115</u>	<u>0.5086</u>	1.6205
Deep fusion + CPH (full FT)	**0.5822**	<u>0.4389</u>	**0.5122**	**0.5111**	<u>1.6189</u>

These results are not directly comparable to the main ablation study, which assumes a frozen base model. Instead, they highlight that the relative effectiveness of conditioning strategies depends strongly on whether the backbone is fixed or jointly optimized.

Finally, fusion-based methods are significantly more parameter-efficient than ControlNet, which approximately doubles the parameter count, highlighting a trade-off between parameter efficiency and conditioning modularity.

6.3.2 Optimization Strategy

Next, we investigate whether other optimizers can improve the model's performance.

1. **AdamW:** AdamW [62] is an adaptive optimizer widely used across a variety of machine learning applications.

2. **Muon:** Some recent frontier LLMs, such as Kimi K2 [95] and GLM-4.5 [122], are trained using the Muon [48] optimizer, which orthogonalizes gradient updates for 2D non-embedding and non-output parameters. Moonshot AI showed that scaling the RMS norm of Muon's updates to match that of AdamW [60] makes its hyperparameters compatible with AdamW [62] without additional tuning. Following Jordan et al. [48], we use AdamW for the remaining parameters.

3. **NorMuon:** Since Muon produces highly non-uniform neuron norms, NorMuon improves upon it by tracking second-order momentum statistics for each neuron and performing row-wise normalization after orthogonalization [58].

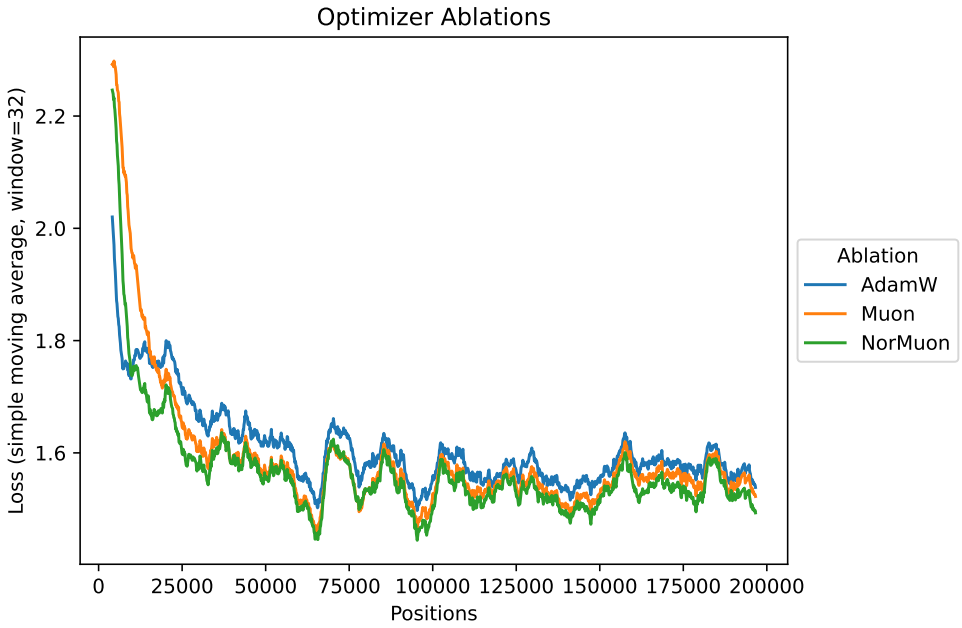

Figure 6.3: Simple moving averages of the losses for each optimizer.

As shown in Figure 6.3, the Muon and NorMuon models exhibit higher initial losses; however, both quickly drop below the AdamW model's loss for the remainder of training. This reduction in loss translates directly into improved benchmark performance, as shown in Table 6.5. Since NorMuon achieves the highest benchmark scores and the lowest loss, we use it for all subsequent experiments.

Table 6.5: Optimizer ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
AdamW*	0.5704	<u>0.4358</u>	<u>0.5179</u>	0.5080	1.5680
Muon	<u>0.5906</u>	0.4346	0.5171	<u>0.5141</u>	<u>1.5525</u>
NorMuon	**0.5919**	**0.4432**	**0.5219**	**0.5190**	**1.5331**

6.3.3 Parameter-Efficient Fine-Tuning

Next, we assess different configurations of parameter-efficient fine-tuning applied to the text encoder. The baseline configuration uses DoRA with rsLoRA and LoRA+, based on the assumption that this configuration would improve performance with minimal computational overhead.

1. **DoRA**: DoRA extends LoRA by decomposing low-rank updates into magnitude and direction components [61].
2. **LoRA**: We determine whether the additional parameters introduced by DoRA are necessary for model performance.
3. **No rsLoRA**: We examine whether removing rsLoRA improves performance. LoRA adapters consist of low-rank matrix products scaled by a rank-dependent factor [42]. rsLoRA uses a modified scaling rule that performs better at higher ranks [49].
4. **No LoRA+**: We examine whether removing LoRA+ improves performance. In LoRA+, the learning rate ratio between the down-projection matrix B and the up-projection matrix A, $\frac{\eta_B}{\eta_A}$, is configured to a constant greater than 1 [37]. The DoRA baseline uses a ratio of 2.
5. **Frozen text encoder**: This follows the ControlNet architecture, except that the text encoder is fully frozen.

Table 6.6: PEFT ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
DoRA*	<u>0.5919</u>	0.4432	<u>0.5219</u>	<u>0.5190</u>	<u>1.5331</u>
LoRA	**0.6015**	0.4408	0.5196	**0.5206**	**1.5282**
No rsLoRA	0.5799	<u>0.4441</u>	0.5200	0.5147	1.5333
No LoRA+	0.5781	**0.4471**	**0.5246**	0.5166	1.5401
Frozen text encoder	0.5545	0.4350	0.5145	0.5013	1.5501

As shown in Figure 6.4, all LoRA-based variants perform similarly, although the frozen text encoder eventually reaches a comparable loss. The LoRA variant achieves a slightly higher mean benchmark score than DoRA, while attaining a lower loss and using fewer parameters (Tables 6.6 and 6.7). Since rsLoRA and LoRA+ introduce minimal computational overhead while providing slight performance gains, we retain both.

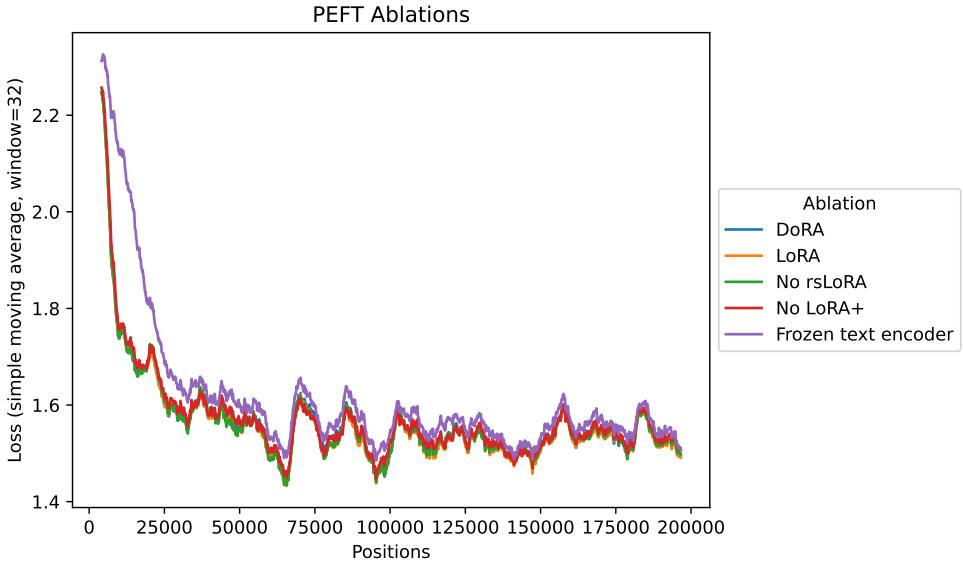

Figure 6.4: Simple moving averages of the training losses for the PEFT ablations.

Table 6.7: PEFT parameter counts.

Experiment	Trainable LLM parameters
DoRA*	4.260M
LoRA	3.932M
Frozen text encoder	0

6.3.4 Lc0 Scaling

Several Lc0-ChessFormer [67, 68] models are available at different parameter scales. We investigate whether increasing the size of the Lc0 backbone improves the performance of our model. Because each variant is used within a ControlNet architecture, any increase in parameter count is effectively doubled in the final model.

All variants consist of 15 transformer layers with 32-dimensional attention heads; larger models increase the number of attention heads, resulting in higher-dimensional embeddings. Deeper Lc0 models are not currently available, so we leave depth scaling to future work. All checkpoints are trained with stochastic weight averaging [45].

1. **t3:** An Lc0-CF model with 0.10B parameters. We use the pretrained `t3-512x15x16h-swa-2815000` weights.

2. **BT3:** An Lc0-CF model with 0.12B parameters. We use the pretrained `BT3-768x15x24h-swa-2790000` weights.

3. **BT4:** An Lc0-CF model with 0.21B parameters. We use the pretrained `BT4-1024x15x32h-swa-5000000` weights.

Table 6.8: Lc0 ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Lc0-CF-t3 (0.10B)*	0.6015	0.4408	0.5196	0.5206	1.5282
Lc0-CF-BT3 (0.12B)	**0.6154**	<u>0.4432</u>	**0.5249**	**0.5278**	<u>1.5136</u>
Lc0-CF-BT4 (0.21B)	<u>0.6071</u>	**0.4449**	<u>0.5244</u>	<u>0.5255</u>	**1.5063**

Increasing the Lc0 width generally improves performance. While BT3 achieves the highest mean benchmark score in this ablation, BT4 attains the lowest loss and remains competitive across all benchmarks.

We note that BT3 also outperforms BT4 in the standalone Lc0 evaluation (see Section 7.3), particularly on human move matching. This likely contributes to its stronger benchmark performance in the ControlNet setting.

Nevertheless, we select BT4 as the baseline for subsequent experiments, prioritizing lower loss and higher capacity despite a small drop in benchmark performance. We do not further disentangle the effect of backbone quality versus ControlNet adaptation and leave this to future work.

6.3.5 Data Sampling

We use Unimax downsampling [15] to allocate a fixed budget of plies (half-moves) across the dataset. Unimax combines uniform allocation with a cap on the number of samples drawn from any individual source, preventing small datasets from being oversampled while limiting the dominance of large datasets. We restrict the maximum to one epoch per dataset, meaning that smaller groups are fully utilized when possible, while larger groups are truncated to satisfy the global budget.

In this ablation, we study how to perform downsampling *within each Elo bucket*. Since our sampling operates at the level of plies rather than full games, each method determines how many plies to extract from each game under a fixed turn budget.

There are two independent design choices:

1. **Group-level balancing**: whether to partition the data into groups (e.g., by opening or time control) and allocate a separate turn budget to each group using Unimax.

2. **Budget allocation**: whether to allocate per-game turn budgets using Unimax (which enforces a capped, approximately uniform allocation) or proportionally to game length.

Intuitively, Unimax reduces the dominance of long games by capping their contribution, whereas proportional allocation preserves the empirical distribution of plies across games. In all cases, plies are ultimately sampled uniformly at random within each game given an allocated budget; the distinction between methods lies in how these per-game budgets are assigned. This decomposition separates dataset-level balancing (across groups) from per-game budget allocation, allowing these two effects to be controlled independently.

By default, we apply Unimax downsampling grouped by opening.

1. **Proportional (global)**: No group-level balancing is applied. The turn budget is allocated across games in proportion to their length.

2. **Proportional (opening)**: The turn budget is first allocated across openings using Unimax to ensure balanced representation. Within each opening, budgets are assigned to games proportionally to their length.

3. **Proportional (opening + time control)**: The budget is allocated across (opening, time control) groups using Unimax. Within each group, budgets are assigned proportionally to game length.

4. **Proportional (time control)**: The budget is allocated across time controls using Unimax, followed by proportional allocation within each group.

5. **Unimax (global)**: No grouping is applied. The global turn budget is allocated across games using Unimax, capping the number of plies per game to encourage uniform contributions.

6. **Unimax (opening)**: The budget is first distributed across openings using Unimax, and then further allocated across games within each opening using Unimax.

7. **Unimax (opening + time control)**: The budget is distributed across (opening, time control) groups using Unimax, and then across games within each group using Unimax, resulting in the most fine-grained balancing.

Table 6.9: Downsampling ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Proportional (global)	0.6357	0.4469	0.5312	0.5380	1.5500
Proportional (opening)	0.6154	0.4492	0.5319	0.5321	1.5651
Proportional (opening + time control)	0.6384	<u>0.4503</u>	0.5260	0.5382	1.6006
Proportional (time control)	<u>0.6503</u>	**0.4503**	0.5312	<u>0.5440</u>	1.5192
Unimax (global)	0.6228	0.4472	**0.5322**	0.5341	**1.4716**
Unimax (opening)*	0.6071	0.4449	0.5244	0.5255	1.5063
Unimax (opening + time control)	0.6182	0.4494	0.5272	0.5316	1.5562
Unimax (time control)	**0.6580**	0.4450	<u>0.5320</u>	**0.5450**	<u>1.4768</u>

8. **Unimax (time control)**: The budget is distributed across time controls using Unimax, and then across games within each time control using Unimax.

As shown in Table 6.9, strategies that balance across time controls consistently outperform those that balance across openings. This suggests that equalizing variation in time control improves generalization, whereas enforcing a uniform distribution over openings may distort the natural opening frequencies observed in Lichess. In particular, balancing by opening can introduce a distribution shift away from the empirical data, which may negatively impact performance. This is consistent with the fact that opening frequencies on Lichess follow a highly skewed distribution, where a small number of openings dominate play. While proportional sampling slightly increases training loss, Unimax budget allocation yields better overall benchmark results. Therefore, we adopt Unimax (time control) for subsequent experiments.

6.3.6 LoRA Rank Ablations

We keep the LoRA scaling factor (α) fixed while varying the LoRA rank across experiments. Specifically, we evaluate powers of two, 2^k, for $k \in [0, 7]$.

From Table 6.10, we observe that the loss does not vary monotonically with increasing rank. In contrast, the mean benchmark score exhibits a clear peak, with the best performance at rank 16 and the second-best at rank 8. Since increasing the LoRA rank also increases the number of trainable parameters, higher ranks should only be used when they yield meaningful performance improvements.

Table 6.10: LoRA rank ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
LoRA rank = 1	0.6494	0.4433	0.5264	0.5397	1.4863
LoRA rank = 2	0.6340	0.4425	0.5299	0.5355	1.4874
LoRA rank = 4	0.6504	0.4442	<u>0.5300</u>	0.5415	1.4866
LoRA rank = 8*	<u>0.6559</u>	**0.4457**	**0.5304**	<u>0.5440</u>	1.4863
LoRA rank = 16	**0.6625**	0.4448	0.5297	**0.5457**	1.4874
LoRA rank = 32	0.6540	<u>0.4453</u>	<u>0.5300</u>	0.5431	1.4892
LoRA rank = 64	0.6545	0.4428	0.5299	0.5424	**1.4832**
LoRA rank = 128	0.6467	0.4429	0.5247	0.5381	<u>1.4843</u>

We hypothesize that only a limited amount of information from the text encoder is required to achieve a meaningful performance gain. Increasing the LoRA rank yields relatively modest improvements. Separately, the PEFT ablations (Table 6.6) show that introducing LoRA provides a substantial improvement over a frozen text encoder. While these results are not directly comparable due to differences in experimental setup, they are consistent with diminishing returns as the rank increases.

6.3.7 Learning Rate Schedule Ablations

We investigate whether a simplified Warmup–Stable–Decay (WSD-S) learning rate schedule outperforms our custom schedule. Wen et al. [114] report that WSD-S outperforms cosine decay schedules. In our setting, where training is limited to data up to December 2013, our custom schedule behaves similarly to a cosine-like decay.

1. **WSD-S (monthly):** Warmup and decay are applied every month.
2. **WSD-S (annual):** Warmup and decay are applied once per year. Since our ablations only cover 2013, this results in a single warmup–decay cycle.

Because the original warmup phase spans the entirety of January 2013, we shorten it to 512 steps and allocate an additional 512 steps for decay, leaving 1,024 steps for stable training.

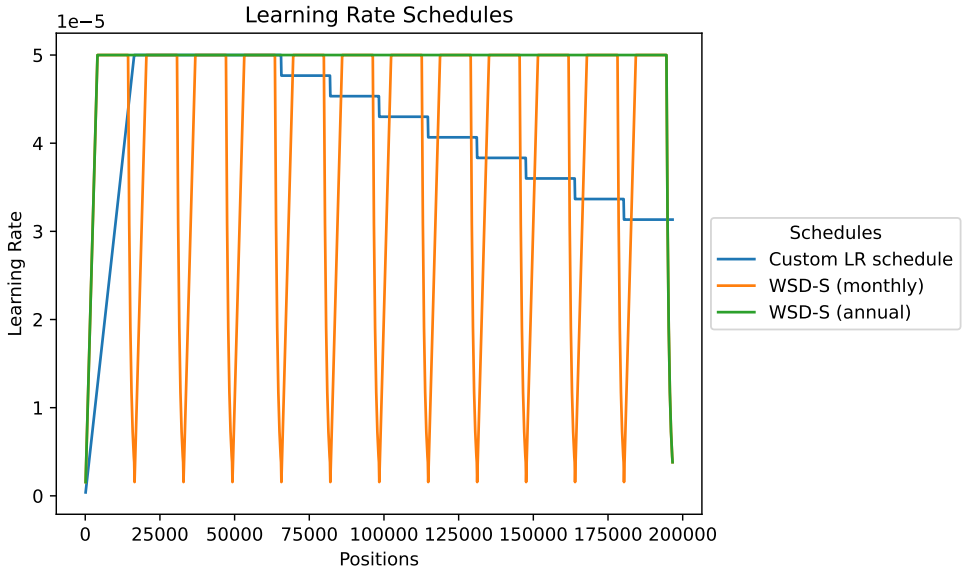

Figure 6.5: The three learning rate schedules.

Table 6.11: Learning rate schedule ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Custom LR schedule*	**0.6580**	**0.4450**	**0.5320**	**0.5450**	**1.4768**
WSD-S (monthly)	0.6019	0.4407	0.5222	0.5216	1.5014
WSD-S (annual)	<u>0.6389</u>	<u>0.4434</u>	<u>0.5226</u>	<u>0.5350</u>	<u>1.4845</u>

As shown in Figure 6.6, the shorter warmup in the WSD-S variants leads to faster initial loss reduction. However, the custom schedule catches up as training progresses, particularly when the WSD-S variants enter their decay phase and the custom schedule continues training at a relatively higher learning rate. Ultimately, as seen in Table 6.11, the custom schedule achieves the lowest final loss.

Although we retain the custom learning rate schedule, the loss curves suggest that the original warmup duration is unnecessarily long. For the full training run, we reduce the number of warmup steps from 2,048 to 1,024.

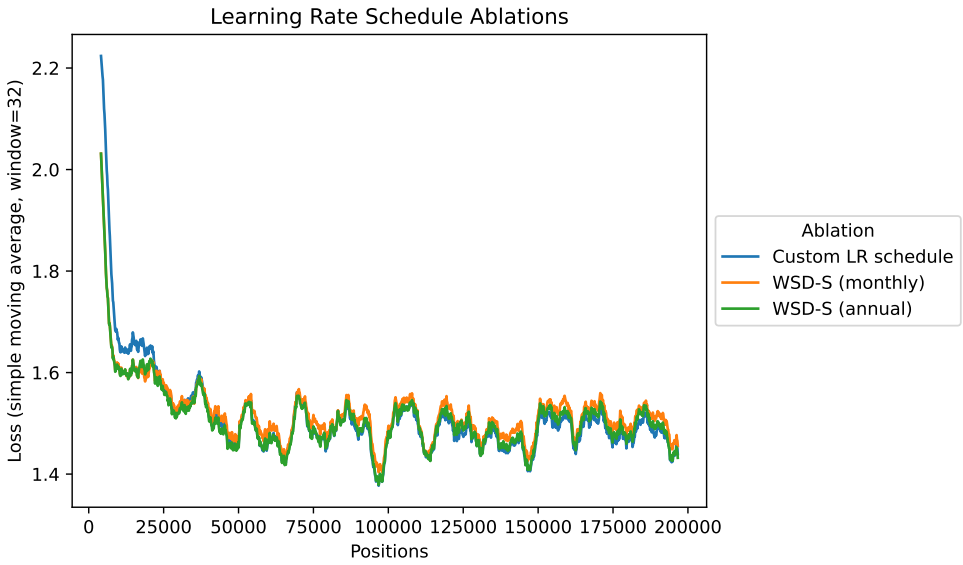

Figure 6.6: Simple moving averages of the training losses for the learning rate schedule ablations. Because the WSD-S variants use fewer warmup steps, their losses are initially lower. However, the custom schedule catches up over time, particularly during the decay phase of the WSD-S variants.

6.3.8 Negative Results

This section presents ablation studies that did not impact the final training configuration, but provide insight into model behavior. Negative results are included to highlight design decisions that were explored but ultimately not retained.

Data Ablations

We evaluate the following template sampling strategies:

1. **50/50 pretrain, instruct templates**: Sample from LichessTemplates-Pretrain and LichessTemplates-Instruct with equal probability.
2. **Pretrain templates**: Sample exclusively from LichessTemplates-Pretrain.
3. **Instruct templates**: Sample exclusively from LichessTemplates-Instruct.

Table 6.12: Data ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
50/50 pretrain, instruct templates*	<u>0.5704</u>	0.4358	<u>0.5179</u>	<u>0.5118</u>	<u>1.5680</u>
Pretrain templates	0.4401	**0.4422**	0.5104	0.4794	**1.5507**
Instruct templates	**0.5819**	<u>0.4402</u>	**0.5221**	**0.5173**	1.5842

The 50/50 configuration achieves comparable mean benchmark performance to using only instruct templates, while yielding lower training loss. We therefore adopt this setting. However, because the instruct template pool (2,048) is substantially larger than the pretrain pool (160), the reduced loss may partially reflect overfitting effects on the smaller pretrain subset.

Despite this, we retain the 50/50 split, as pretrain templates contain richer metadata and may provide a stronger semantic signal during training. A more principled exploration of mixture ratios or template design is left to future work.

Following Maia-2 Rapid [91], we also experiment with restricting training to rapid games and omitting the first 10 plies. While this significantly reduces training loss, it yields only marginal improvements on the Maia-2 rapid benchmark. Since our objective is to train a unified, prompt-controllable model across time controls, we do not pursue this direction further. Prior work such as Allie [127] demonstrates that unified models across time controls are not only feasible, but can even outperform specialized models such as Maia-2 Rapid on their own benchmarks.

Text Model

We next evaluate the impact of the text encoder, varying model size, training data, and architecture family.

Within the GPT-NeoX [8] family:

1. **RedPajama INCITE Base (3B)**: A fully open-source GPT-NeoX model released under the Apache 2.0 license [113].
2. **ChessGPT-base**: Fine-tuned from RedPajama INCITE Base (3B) on a mixture of natural language and chess data [27].

3. **ChessGPT-chat**: Further fine-tuned from ChessGPT-base on chess Q&A data [27].

From the Gemma family:

1. **Gemma 3 (1B)**: A 1B-parameter autoregressive model [94].
2. **T5 Gemma 2 (1B–1B)**: An encoder-decoder model initialized from Gemma 3 (1B) and pre-trained with the UL2 objective [93, 123]. We use only the encoder.
3. **T5 Gemma 2 (270M–270M)**: A smaller variant initialized from Gemma 3 (270M), again using only the encoder.
4. **Embedding Gemma (300M)**: A Matryoshka embedding model derived from T5 Gemma 2 (270M–270M) [53, 111].

From the Qwen family:

1. **Qwen 3 Embedding (0.6B)**: A Qwen 3 model fine-tuned for text embeddings [126].

Table 6.13: LLM ablation results. Best values are in bold and second-best values are underlined. †Because the LoRA model produced NaN losses, we instead train a DoRA model to report the benchmark scores and the mean loss for December 2013.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Gemma 3 (1B)	0.4817	0.4275	0.5049	0.4714	1.6049
T5Gemma 2 Encoder (1B)	0.4555	0.4287	0.5056	0.4632	1.6111
T5Gemma 2 Encoder (0.27B)	0.5168	0.4329	0.5161	0.4886	1.5873
Embedding Gemma (0.30B)	0.5273	0.4381	**0.5212**	0.4955	<u>1.5512</u>
Qwen 3 Embedding (0.60B)	0.5772†	**0.4441†**	<u>0.5207</u>†	<u>0.5140</u>†	1.5733†
Redpajama Incite Base (3B)	0.5108	0.4341	0.5100	0.4850	1.5916
ChessGPT-base v1 (3B)*	**0.6015**	0.4408	0.5196	**0.5206**	**1.5282**
ChessGPT-chat v1 (3B)	<u>0.5778</u>	<u>0.4433</u>	0.5147	0.5119	1.5617

Increasing model size does not consistently improve performance. For example, the larger T5 Gemma 2 model underperforms its smaller counterpart (Table 6.13). Instead, training data composition appears to be the dominant factor: models trained on both chess and natural language data perform best, while embedding-focused models also yield competitive results.

Given the relatively simple prompt structure, the full capacity of larger LLMs is likely underutilized, suggesting that smaller, domain-specialized encoders may suffice. We leave this direction to future work.

As none of the evaluated alternatives outperform ChessGPT-base, we retain it for all subsequent experiments.

Fine-tuning ChessGPT-base

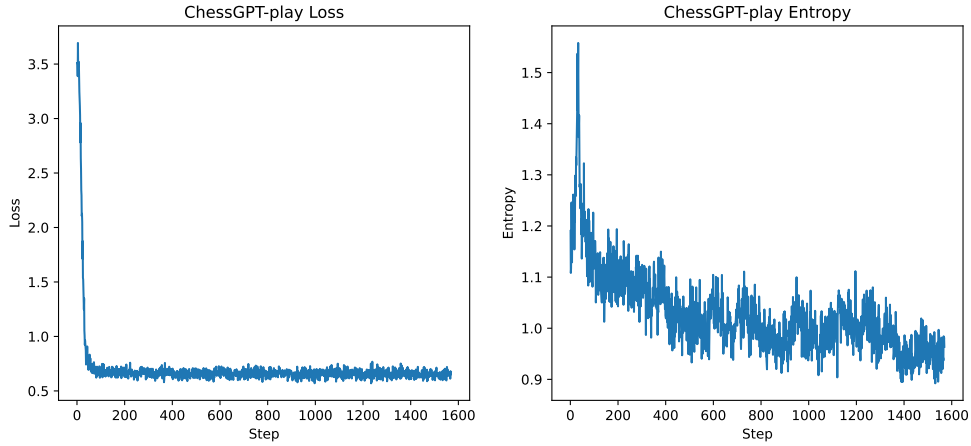

Figure 6.7: Training loss (left) and entropy (right) over the course of training. While loss and mean token accuracy plateau, entropy continues to decrease, indicating increasing prediction confidence.

1. **ChessGPT-play**: We fine-tune ChessGPT-base on the 2013 training split using the same hyperparameters, without incorporating Lc0. The model directly predicts the next move using the template `prompt<|endoftext|>pgn`. We refer to this variant as ChessGPT-play.

   Training loss decreases rapidly within the first 50 steps and then stabilizes (Figure 6.7). Interestingly, entropy continues to decrease even after the loss plateaus, suggesting increasing confidence without corresponding improvements in predictive accuracy. This behavior was not tracked in the ControlNet experiments, but may be informative for future analysis.

2. **ControlNet (ChessGPT-base)**: Our baseline ControlNet model, incorporating improvements such as the BT4 backbone, NorMuon optimizer, and rank-16 LoRA. As in other ablations, we use 2,048 warmup steps.

3. **ControlNet (ChessGPT-play)**: A variant in which the text encoder is replaced with ChessGPT-play.

4. **ControlNet (ChessGPT-play), end-of-text PGNs**: To reduce distribution shift, we modify the prompt format to {`prompt`}<|`endoftext`|>{`pgn`}.

Table 6.14: Fine-tuned base model ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
ChessGPT-play	**0.8080**	0.3975	0.4343	**0.5466**	**0.6530**
ControlNet (ChessGPT-base)*	<u>0.6625</u>	<u>0.4448</u>	<u>0.5297</u>	<u>0.5457</u>	1.4874
ControlNet (ChessGPT-play)	0.6355	0.4445	**0.5314**	0.5371	1.4975
ControlNet (ChessGPT-play), end-of-text PGN	0.6494	**0.4453**	0.5281	0.5409	<u>1.4872</u>

Table 6.15: Fine-tuned base model ablations evaluated on opening instruction-following benchmarks. Best values are in bold and second-best values are underlined. †Prompted with the PGN header.

Experiment	LOB-P	LOB-C	LOB-P†	LOB-C†	Mean
ChessGPT-play	**0.8080**	**0.8799**	**0.8305**	**0.8327**	**0.8378**
ControlNet (ChessGPT-base)*	<u>0.6625</u>	<u>0.6844</u>	<u>0.6569</u>	<u>0.6608</u>	<u>0.6661</u>
ControlNet (ChessGPT-play)	0.6355	0.6523	0.6119	0.6151	0.6287
ControlNet (ChessGPT-play), end-of-text PGN	0.6494	0.6715	0.6388	0.6420	0.6504

As shown in Table 6.14, ChessGPT-play performs strongly on LOB-P but degrades substantially on the LIF-D and M1-S benchmarks. ControlNet variants that use ChessGPT-play as the text encoder similarly underperform relative to those using ChessGPT-base. Notably, improvements on the openings benchmark do not carry over when the fine-tuned model is used as the ControlNet text encoder. We further evaluate the ablations on additional opening instruction-following benchmarks (Table 6.15) to assess whether using ChessGPT-play within ControlNet improves performance on these tasks, but no such gains are observed.

For the end-of-text variant, we provide both the end-of-text token and PGN during inference. Performance remains nearly unchanged even when these inputs are omitted, although we do not report these results.

LoRA Target Modules Ablations

We vary the set of modules targeted by LoRA in the text encoder:

1. **QKV projections:** Apply LoRA only to the query, key, and value projections.
2. **QKV + O projections:** Additionally adapt the output projection.
3. **QKV + O + FFN:** Further extend LoRA to the feedforward layers.

Table 6.16: LoRA target modules ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
LoRA target = W_{QKV}	**0.6455**	**0.4445**	0.5260	<u>0.5387</u>	<u>1.4867</u>
LoRA target = W_{QKV}, $W_O{}^*$	<u>0.6443</u>	<u>0.4426</u>	**0.5315**	**0.5394**	1.4871
LoRA target = W_{QKV}, W_O, FFN	0.6407	0.4419	<u>0.5288</u>	0.5371	**1.4855**

The performance differences between configurations are negligible, with all variants falling within a narrow margin. While targeting both QKV and O projections yields the highest mean score, the gains over the QKV-only configuration are minimal and likely within the variance induced by random initialization.

These results suggest that expanding the set of target modules beyond attention projections provides limited benefit in this setting. In particular, adding LoRA to the feedforward layers increases parameter count without yielding consistent improvements, indicating diminishing returns.

LoRA Rank with Scaled α Ablations

Here, we scale the α value with the rank while keeping the same rsLoRA multiplier used in the rank 16 experiment in Section 6.3.6. We evaluate ranks 2^k with $\alpha = 4 \cdot 2^{k/2}$, for $k \in [2, 6]$.

Table 6.17: LoRA rank with scaled α ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
LoRA rank = 4, $\alpha = 8$	0.6511	0.4438	0.5272	0.5407	**1.4866**
LoRA rank = 8, $\alpha = 8\sqrt{2}$	0.6441	0.4427	0.5288	0.5385	1.4908
LoRA rank = 16, $\alpha = 16^*$	**0.6625**	<u>0.4448</u>	0.5297	**0.5457**	1.4874
LoRA rank = 32, $\alpha = 16\sqrt{2}$	<u>0.6546</u>	**0.4456**	**0.5338**	<u>0.5447</u>	1.4870
LoRA rank = 64, $\alpha = 32$	0.6468	0.4406	<u>0.5327</u>	0.5400	<u>1.4868</u>

Table 6.17 shows that performance again peaks at moderate ranks, with rank 16 achieving the highest mean score. Increasing the rank beyond this point provides only marginal or inconsistent improvements despite the additional parameters.

Compared to the fixed-α setting (Table 6.10), scaling α with rank produces a similar trend, suggesting that the effective capacity of the adapters saturates in this regime. Although scaling α stabilizes training across ranks, it does not substantially change the optimal rank, reinforcing that moderate-rank adapters provide the best trade-off between performance and parameter efficiency.

NorMuon Hyperparameter Ablations

We also try adding Nesterov momentum [19], but the performance difference was negligible (slightly worse). We also try using the Polar Express algorithm [4] instead of Newton-Schulz to approximate polar$(\cdot)$ when trying to orthogonalize the gradient updates, but we find that it (1) occasionally causes gradient spikes in the LoRA adapters for the text encoder, which increases the loss slightly thereafter and (2) makes it more likely for the loss to become NaN in some execution environments, so we don't use it. As mentioned by the Kimi Team, Muon tends to have exploding attention logits more frequently than AdamW [95]. Using a DoRA adapter tends to be more stable when using the Polar Express algorithm compared to LoRA.

Key Padding Mask Ablations

We also experimented with masking padding tokens in the cross-attention keys, which resulted in a slight performance degradation. Adding gated attention [76] did not recover this loss.

Table 6.18: Key padding mask ablation results. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Without key padding mask*	**0.6625**	<u>0.4448</u>	**0.5297**	**0.5457**	**1.4874**
With key padding mask	<u>0.6496</u>	**0.4459**	<u>0.5287</u>	<u>0.5414</u>	<u>1.4900</u>
With key padding mask and gated attention	0.5338	0.4298	0.5256	0.4964	1.5354

6.4 Reproducibility and Sources of Non-Determinism

Despite fixing random seeds for Python, NumPy, and PyTorch (CPU and CUDA), we observe slight differences in training loss trajectories across execution environments. In particular, runs performed locally and on university servers exhibit consistent high-level behavior, with major peaks and dips aligning with dataset shard boundaries (e.g., the beginning and end of each month).

However, finer-grained variations within each shard differ systematically between environments. Since each month corresponds to a single shard in earlier portions of the dataset (e.g., 2013), this suggests that while shard-level ordering is preserved, the ordering of samples within each shard varies depending on the data loading pipeline (e.g., streaming versus local parquet access with an `IterableDataset`).

Within a fixed environment, repeated runs produce nearly identical trajectories, with only minor numerical differences attributable to known sources of non-determinism in GPU computation and multi-worker data loading. Overall, these results indicate that reproducibility is preserved at a coarse level, but exact training dynamics remain sensitive to data ordering and system-level factors.

Table 6.19: Results for repeated runs across local and server execution environments. Best values are in bold and second-best values are underlined.

Experiment	LOB-P	LIF-D	M1-S	Mean	$\mu_{\text{loss, Dec13}}$
Local run 1	**0.6580**	0.4450	**0.5320**	**0.5450**	**1.4768**
Local run 2	0.6429	**0.4470**	0.5254	0.5385	<u>1.4813</u>
Server run 1	<u>0.6559</u>	<u>0.4457</u>	0.5304	<u>0.5440</u>	1.4863
Server run 2	0.6443	0.4426	<u>0.5315</u>	0.5394	1.4871

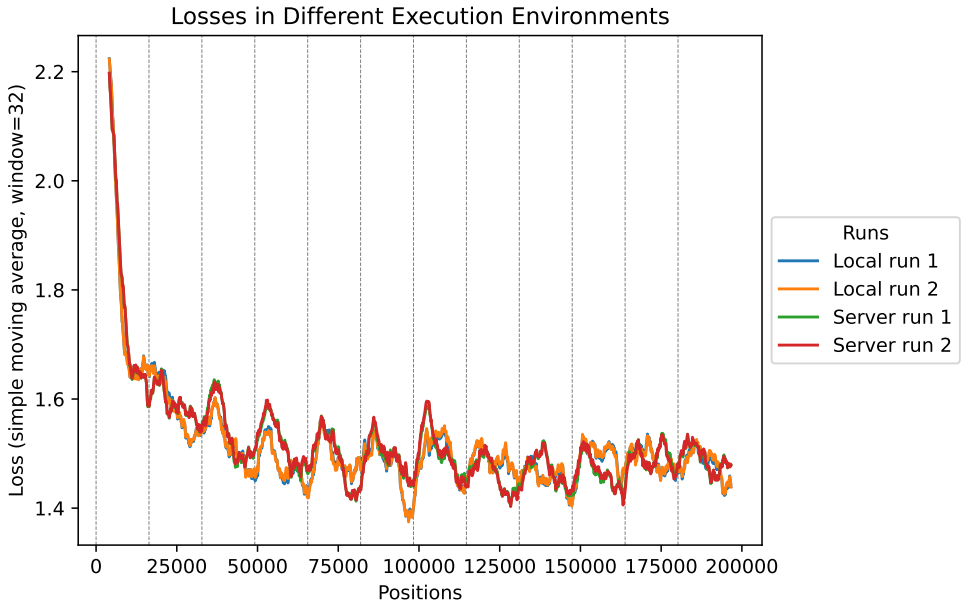

Figure 6.8: The training losses for two local and two server runs, all with the same training configuration. Runs within the same execution environment exhibit highly similar loss trajectories, while runs across environments differ slightly. In these experiments, each shard consists of 2^{14} plies.

Figure 6.8 illustrates this behavior: runs within the same execution environment exhibit highly similar loss trajectories, while runs across environments differ in their within-shard dynamics. This pattern is also reflected in Table 6.19, where runs from the same environment yield similar losses and benchmark scores. While the local runs achieve slightly lower final losses on average, the differences are modest and do not materially affect the overall conclusions.

6.5 Full Training Run

6.5.1 Final Model Architecture

We present the final model architecture in Figure 6.9, which integrates the key design decisions identified throughout Sections 6.3.1–6.3.7. In particular, the model combines (i) a frozen Lc0-CF backbone initialized with the BT4 weights, (ii) a text encoder augmented with

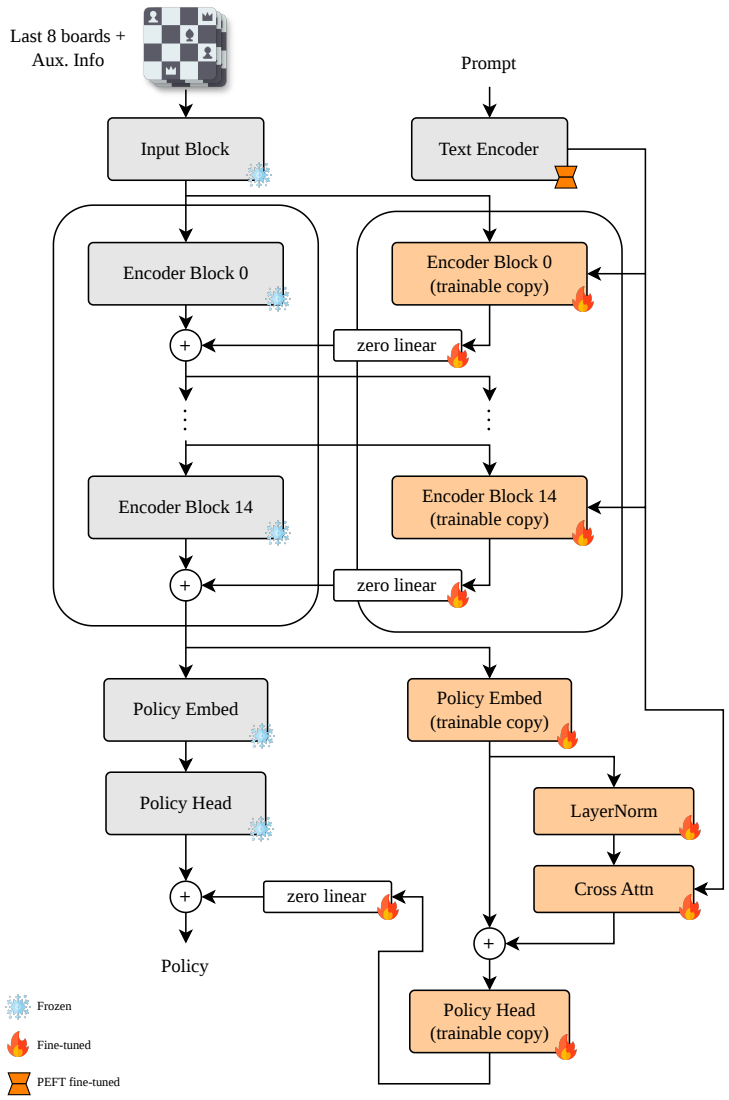

Figure 6.9: Unlike ControlNet architectures used in image generation models, both the base layer and ControlNet layer at depth n take as input the hidden representation from base layer $n-1$. This coupling keeps the ControlNet branch aligned with the base model and prevents representations from drifting out of distribution.

parameter-efficient fine-tuning (LoRA with rsLoRA and LoRA+), and (iii) a ControlNet-style conditioning mechanism that injects text-derived features into the policy network via cross-attention.

The Lc0 backbone processes board states to produce intermediate representations, while the text encoder maps natural language prompts to semantic embeddings. These embeddings are incorporated through cross-attention in the ControlNet branch, allowing the model to modulate its policy predictions based on the provided textual context. The resulting architecture enables controllable, human-like move generation across a range of skill levels, while preserving the strong inductive biases of the underlying chess engine.

This configuration reflects the combination of architectural choices (Section 6.3.1), parameter-efficient fine-tuning strategies (Section 6.3.3), and optimization refinements (Sections 6.3.2 and 6.3.7) that yielded the best empirical performance.

6.5.2 Final Training Configuration

We summarize the key settings used in the full training run for reproducibility.

Data. We train on the Lichess dataset spanning January 2013 to January 2022, corresponding to a total of 66.978M plies. Training samples are allocated using Unimax downsampling grouped by time control (Section 6.3.5), ensuring balanced coverage while limiting over-representation from large groups.

Model. The model uses a BT4 Lc0 backbone with a frozen policy network and a text encoder augmented with LoRA adapters, combined with rsLoRA and LoRA+, at the selected rank determined in Section 6.3.6.

Optimization. We use NorMuon on the 2D non-embedding, non-output parameters and AdamW on the remaining parameters, with weight decay 0.01 and $\epsilon = 1 \times 10^{-8}$. The learning rate follows the schedule described in Section 6.3.7, with a linear warmup over the first 1,024 steps to a peak learning rate of 5×10^{-5}, followed by staged decay throughout training. For the full training run, we use a batch size of 64.

Training procedure. Training is performed sequentially over monthly dataset partitions, each divided into independently shuffled shards. This design enables efficient streaming from disk and robust checkpointing. Since training is performed for a single epoch, the training loss is equivalent to the validation loss; we therefore report the average loss over each month. To track performance over time, we evaluate each monthly checkpoint

on the benchmarks described in Section 6.1.3. Experiments are conducted on a multi-GPU SLURM setup using two RTX Pro 6000 GPUs, with fixed random seeds for Python, NumPy, and PyTorch.

6.5.3 Training Dynamics

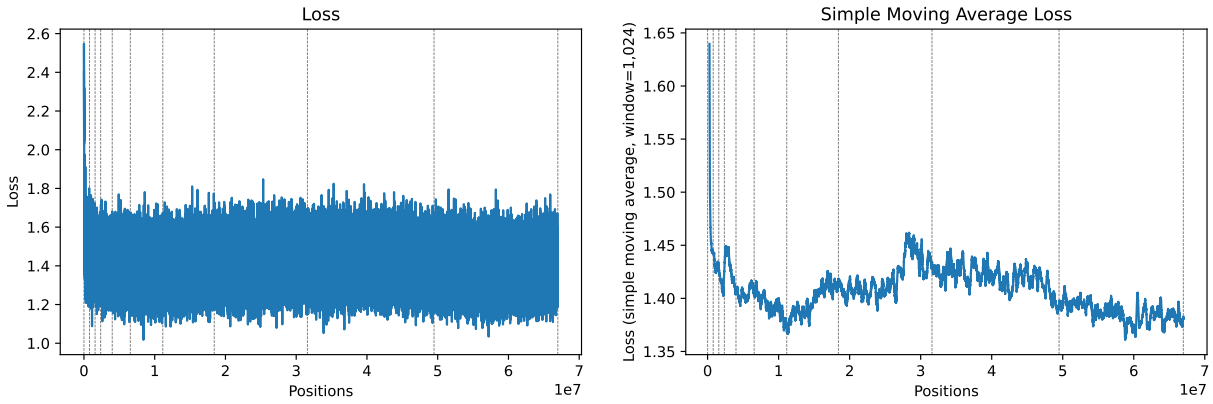

Figure 6.10: Left: raw training loss. The loss decreases rapidly early in training and then plateaus, while maintaining noticeable variance. Right: simple moving average of the loss (window size 1,024). A clear year-dependent trend is visible, with vertical lines indicating year boundaries.

To track model performance throughout training, we plot the training loss alongside five benchmark scores (defined in Chapter 7, Section 7.1) covering (i) instruction-following for chess openings (LOB–P, LOB–C), (ii) general instruction-following (LIF-D, LIF), and (iii) human move prediction (M1-S).

Despite year and month-dependent fluctuations in the training loss, all benchmark scores except LOB-P exhibit an overall upward trend over the course of training. While the curves do not exhibit a clear plateau, the rate of improvement diminishes over time, indicating diminishing returns as additional data is incorporated. These loss fluctuations are consistent with the shard-dependent variation discussed in Section 6.4.

The two LOB variants differ in the amount of opening information provided in the prompt. LOB–C includes the full canonical opening sequence, which strongly constrains the valid continuation, whereas LOB–P provides only partial information. As a result, LOB–C consistently achieves higher scores.

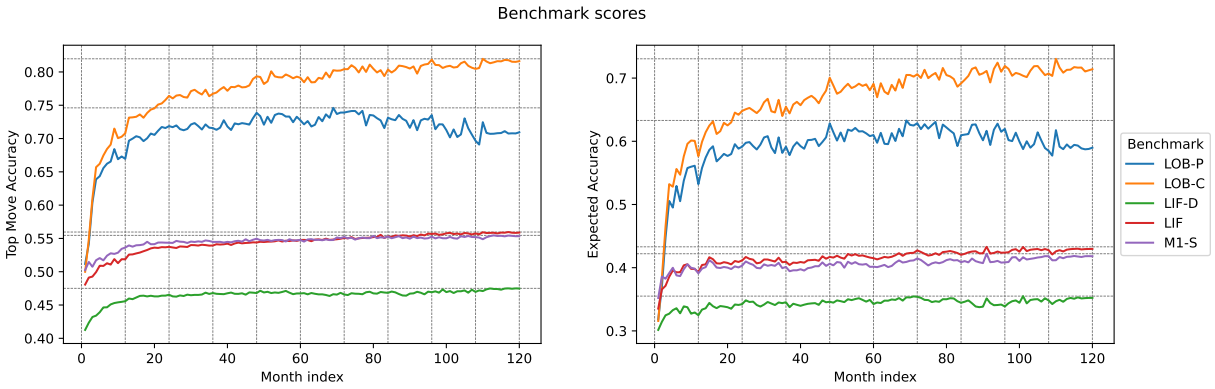

Figure 6.11: Top move accuracy and expected accuracy for five benchmarks across monthly checkpoints during training. Vertical lines indicate year boundaries, and horizontal lines denote the maximum value attained for each benchmark. LOB–C consistently achieves higher scores than LOB–P, reflecting the stronger constraints imposed by the inclusion of canonical opening sequences. While most benchmarks exhibit an overall upward trend, LOB–P plateaus and slightly declines in later stages of training. The January 2013 scores for LIF and LOB-C are computed from a recomputed checkpoint, as the original was corrupted. See Section 6.4 for a discussion of reproducibility effects observed across runs.

As training progresses, performance on LOB–C continues to improve, while LOB–P plateaus and eventually declines slightly. This suggests that the model increasingly benefits from explicit move sequences when they are available, while improvements under weaker constraints are more limited.

To better understand this behavior, we partition templates based on their final top move accuracy using a threshold of 0.74, which largely separates the two groups. This reveals a clear bifurcation (Figure 6.12): templates above the threshold continue to improve throughout training, whereas those below the threshold degrade over time.

We find that this split aligns closely with whether the prompt includes an explicit opening PGN . Nearly all high-performing templates omit the PGN and instead rely on semantic descriptors (e.g., opening names), while nearly all low-performing templates include the PGN sequence.

The worst-performing template corresponds to a prompt that effectively reduces to a general move prediction task conditioned only on the partial PGN, without specifying an opening.

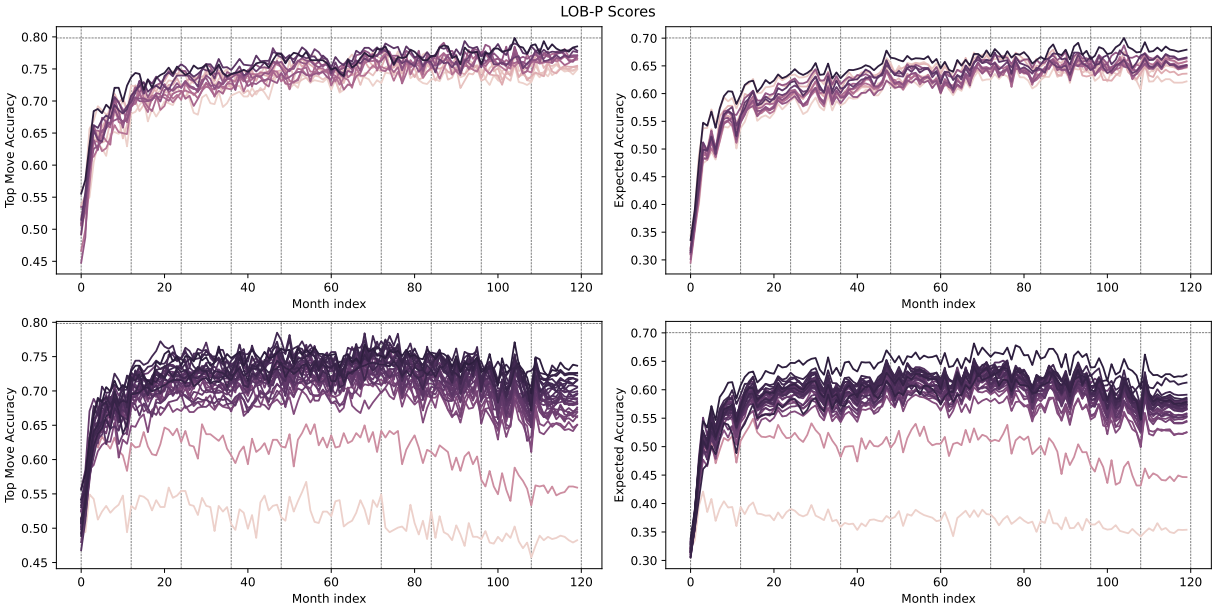

Figure 6.12: Training dynamics for LOB-P templates, grouped by final performance. Templates are partitioned using a top move accuracy threshold of 0.74. The top row shows high-performing templates, while the bottom row shows low-performing templates. Left: top move accuracy over training. Right: expected accuracy over training. High-performing templates (predominantly without PGN conditioning) exhibit consistent improvement, whereas low-performing templates (predominantly with PGN conditioning) degrade over time.

Transient increases in the training loss tend to affect expected accuracy more than top move accuracy, although the magnitude of this effect is modest.

Table 6.20: Top move accuracy for five benchmarks at December checkpoints for each year of training, along with the cumulative number of positions processed up to each checkpoint.

Month	LOB-P	LOB-C	LIF-D	LIF	M1-S	Cumulative Positions
December 2013	0.6695	0.7079	0.4557	0.5187	0.5363	786,432
December 2014	0.7187	0.7641	0.4654	0.5374	0.5435	1,572,864
December 2015	0.7157	0.7672	0.4680	0.5394	0.5448	2,359,296
December 2016	0.7386	0.7934	0.4680	0.5439	0.5496	3,997,696
December 2017	0.7274	0.7912	0.4674	0.5477	0.5477	6,553,600
December 2018	0.7390	0.8042	0.4658	0.5508	0.5522	11,141,120
December 2019	0.7313	0.8036	0.4683	0.5532	0.5528	18,415,616
December 2020	0.7351	0.8185	0.4692	0.5567	0.5526	31,588,352
December 2021	0.6966	0.8046	0.4697	0.5575	0.5520	49,479,680
December 2022	0.7094	0.8162	0.4748	0.5587	0.5534	66,977,792

6.6 Fine-tuning on Auxiliary Targets

6.6.1 Motivation and Dataset

While UniMaia is primarily trained for move prediction conditioned on natural language prompts, the Lichess Open Database additionally contains temporal metadata that may help model human chess behavior. In particular, since April 2017, games in the database include `%clk` annotations that record the remaining clock time for both players after each move.

To investigate whether UniMaia can incorporate this temporal metadata, we further fine-tune the final December 2022 checkpoint on games from 2018, which we refer to as **UniMaia-Aux**. We select 2018 because it contains the required clock annotations while remaining sufficiently lightweight for a targeted fine-tuning experiment.

6.6.2 Prompt Augmentation

To incorporate temporal metadata into the prompt-conditioned setting, we append a suffix to each prompt containing information about the time control, the remaining clock time for both players, and the move delay from the previous turn when available. The exact formatting is provided in Appendix C.5.

To preserve backwards compatibility when this metadata is unavailable, the suffix is omitted with probability 0.1 during training.

6.6.3 Auxiliary Prediction Heads

In addition to the policy head, UniMaia-Aux predicts several auxiliary targets related to game metadata and temporal behavior.

The auxiliary targets consist of:

(i) the final game result, represented as a categorical terminal-outcome distribution Z over $\{-1, 0, 1\}$, where a white win corresponds to 1, a draw corresponds to 0, and a black win corresponds to -1;

(ii) the number of plies remaining until the end of the game, represented as $N - i - 1$, where N is the total number of plies in the game and i is the current ply index;

(iii) the termination reason (e.g., checkmate, timeout, resignation, or draw by threefold repetition); and

(iv) the move delay of the active player, denoted by Δt_a.

The scalar value estimate V can then be recovered from the predicted terminal-outcome distribution by taking its expectation:

$$V = \mathbb{E}[Z].$$

To support these predictions, we add trainable copies of the value and moves-left heads that predict residual updates to the corresponding frozen Lc0 heads. We additionally introduce a termination prediction head and a move-delay prediction head.

The value and termination heads predict categorical targets using a cross-entropy loss, while the moves-left and move-delay heads predict scalar targets using a Huber loss.

6.6.4 Training Dynamics

Figure 6.13 shows the optimization dynamics throughout auxiliary-target fine-tuning.

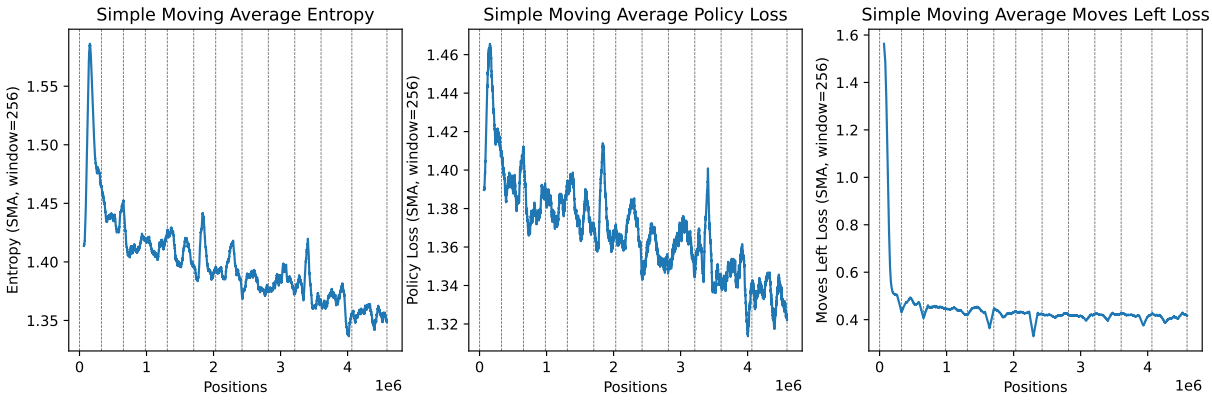

Figure 6.13: Left: simple moving average of the policy entropy. Middle: simple moving average of the policy loss. Right: simple moving average of the moves-left loss.

The policy loss initially decreases until the model exits a local policy minimum, temporarily increasing the policy loss and sharply increasing the policy entropy. This transition coincides with decreases in most auxiliary losses except for the move-delay loss, which remains comparatively stable throughout training with occasional spikes. The policy loss subsequently decreases again as training progresses.

For the benchmark trajectories in Figures 6.14 and 6.15, the prompts include the time-control metadata but omit the remaining clock information and previous move delay. In contrast, the auxiliary benchmarks in Figures 6.16 and 6.17 use the full temporal metadata suffix. The first pair of figures corresponds to the original evaluation benchmarks, whereas the second pair corresponds to LIF-Aux and LGB-Aux.

Interestingly, unlike UniMaia, the expected accuracy of UniMaia-Aux continues to improve even when the top-move accuracy remains comparatively stable. This suggests that the auxiliary temporal metadata may improve the calibration and distributional structure of the policy more strongly than the exact top-move prediction itself.

Although the auxiliary-target fine-tuning improves performance on the augmented benchmarks, performance on the original benchmarks tends to decline somewhat later in training despite continued exposure to the temporal metadata. This suggests that the model may partially overfit to the auxiliary supervision setup or to the prompt formatting introduced during fine-tuning.

Finally, while the temporal metadata improves behavioral modeling, the inclusion of the auxiliary suffix may dilute the prompt-conditioning signal during the opening phase of the game, potentially contributing to the slight decrease in opening memorization performance

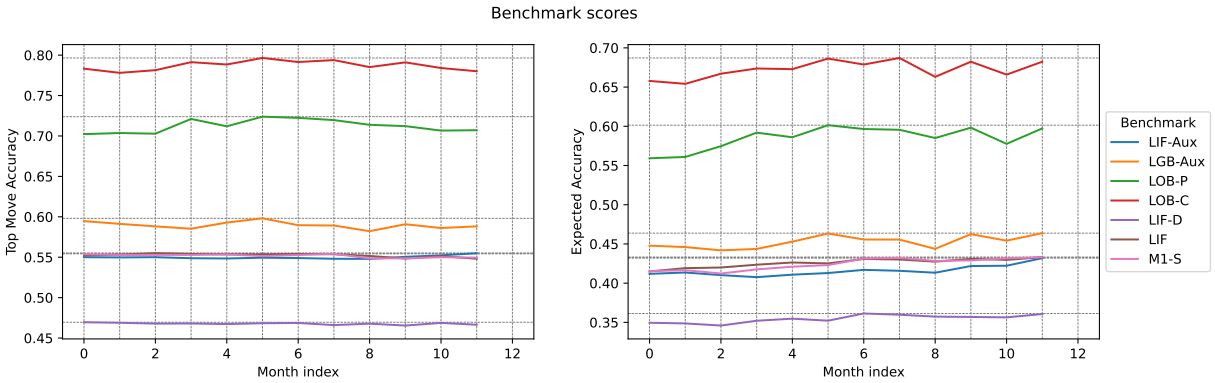

Figure 6.14: Top move accuracy and expected accuracy for seven benchmarks across monthly checkpoints during auxiliary-target fine-tuning. Vertical lines indicate year boundaries, and horizontal lines denote the maximum value attained for each benchmark. While the top-move accuracies remain relatively stable, the expected accuracies tend to improve slightly.

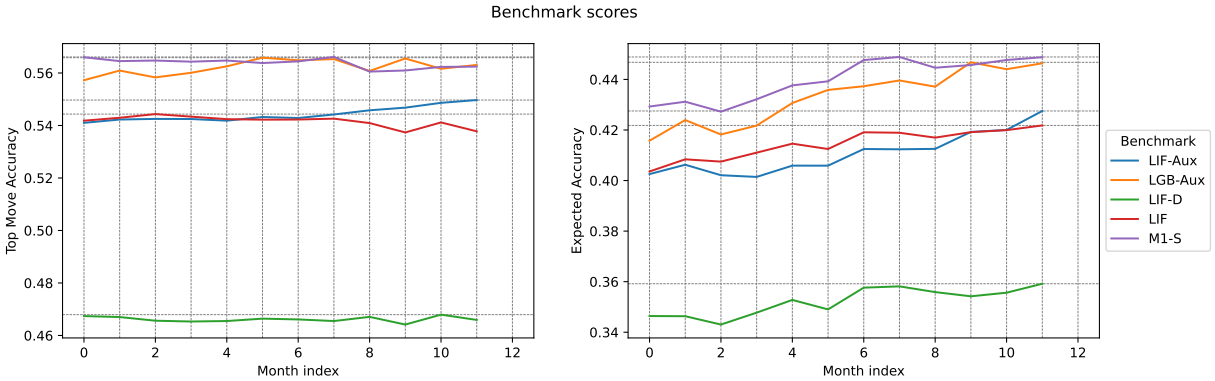

Figure 6.15: Top move accuracy and expected accuracy for seven benchmarks across monthly checkpoints during auxiliary-target fine-tuning, where the first 10 plies are omitted from evaluation. Vertical lines indicate year boundaries, and horizontal lines denote the maximum value attained for each benchmark. The expected accuracies generally improve more strongly when the opening plies are omitted.

observed after fine-tuning. Additionally, the inclusion of move-delay information changes the prompt embeddings even for templates containing only opening-specific information,

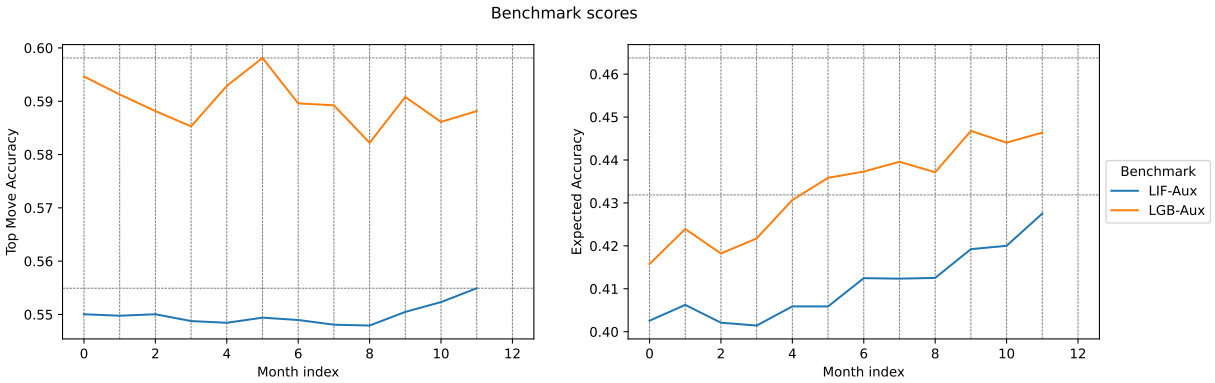

Figure 6.16: Top move accuracy and expected accuracy for LIF-Aux and LGB-Aux across monthly checkpoints during auxiliary-target fine-tuning. Vertical lines indicate year boundaries, and horizontal lines denote the maximum value attained for each benchmark. The expected accuracies for both benchmarks generally improve throughout training.

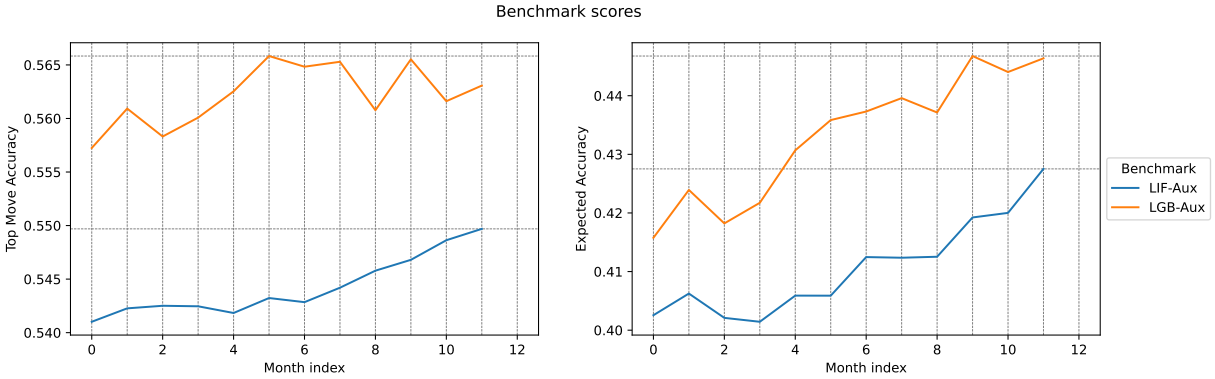

Figure 6.17: Top move accuracy and expected accuracy for LIF-Aux and LGB-Aux across monthly checkpoints during auxiliary-target fine-tuning, where the first 10 plies are omitted from evaluation. Vertical lines indicate year boundaries, and horizontal lines denote the maximum value attained for each benchmark. The gains in top-move accuracy are generally stronger when the opening plies are omitted.

since games following the same opening no longer necessarily share identical prompt text. This may reduce the model's ability to memorize fixed prompt–opening associations by encouraging it to rely on a broader distribution of contextual signals.

6.7 Training Cost and Accessibility

UniMaia's frozen 3B-parameter LLM backbone introduces substantial computational overhead. The resulting batch size of 64 requires many more optimization steps per training position than prior structured-input chess models. Full training required 31.6 days on two RTX Pro 6000 GPUs. Later training years also contain substantially more data, increasing training cost despite diminishing benchmark improvements over time.

In contrast, smaller-scale experiments were considerably more accessible. The 2013-only ablation runs completed in roughly 7 hours on a single consumer-grade RTX 5090, while fine-tuning UniMaia-Aux on 2018 data required approximately 1 day on two RTX Pro 6000 GPUs. These results suggest that ablation studies and downstream adaptation remain feasible on modern hardware despite the high cost of full-scale training.

Nevertheless, the preprocessing pipeline and large-scale multi-GPU training setup may still limit reproducibility and accessibility. Future work could reduce this overhead through smaller language model backbones, more efficient conditioning mechanisms, or more aggressive quantization.

Chapter 7

Evaluation

7.1 Benchmarks

We evaluate model performance using a suite of benchmarks designed to measure complementary aspects of behavior under different conditioning mechanisms, including opening-specific tasks, prompt-conditioned tasks, and metadata-conditioned tasks. These benchmarks are used both for ablation studies and training analysis (Chapter 6) and for final evaluation against competing models.

Prompt-conditioned benchmarks evaluate the model's ability to adapt predictions to natural language inputs, while metadata-conditioned benchmarks evaluate performance given structured player and game information. In all cases, performance is measured by agreement with human moves.

7.1.1 Opening Tasks

- **Lichess Openings Benchmark (LOB):** Positions derived from a curated database of chess openings, specified by ECO codes, opening names, and canonical move sequences. Each position is paired with a prompt that instructs the model to follow a particular opening, using templates sampled from LichessTemplates-Instruct.

  We consider two variants of this benchmark that differ in the amount of opening information provided in the prompt:

- **LOB-Partial (LOB-P; 31,169 positions):** The prompt includes the opening name and/or moves up to the current position. One template effectively reduces to a general move prediction task by conditioning only on the partial PGN up to the current position and is therefore less aligned with the opening-following objective.
- **LOB-Canonical (LOB-C; 32,736 positions):** The prompt additionally includes the full canonical opening sequence, including moves beyond the current position.

LOB–C imposes a stronger constraint on the model's output through the inclusion of the full canonical sequence, whereas LOB–P avoids potential answer leakage by conditioning only on partial information, resulting in a more challenging evaluation setting.

7.1.2 Prompt-Conditioned Tasks

- **Lichess Instruction-Following Benchmark (LIF; 25,000 positions):** A large-scale benchmark constructed from positions sampled from the 2023 LichessGames dataset and formatted using templates from LichessTemplates-Pretrain and LichessTemplates-Instruct. This benchmark reflects a broad distribution of player skill levels and game contexts, and serves as the primary evaluation of the model's ability to follow natural-language instructions in realistic gameplay settings.
- **Lichess Instruction-Following Diagnostic (LIF-D; 19,606 positions):** A controlled diagnostic benchmark constructed from January 2013 positions in LichessGames, restricted to the 900–1100 ELO range and formatted using the same template families as LIF. Because it is drawn from the same temporal distribution as the training data, LIF-D is used only for ablation studies and training diagnostics, and not for final evaluation.
- **Lichess Instruction-Following (Top-10 Active Players) (LIF-T10; 25,000 positions):** A variant of LIF constructed using positions from January 2023 games in LichessGames, restricted to games in which at least one player is among the most active players on Lichess. The benchmark is formatted using the same template families as LIF.

This benchmark is designed to evaluate instruction-following performance in settings where player identities are more likely to be recognizable, potentially allowing models to leverage implicit knowledge of player-specific tendencies when such information is included in the prompt.

- **Lumbra's Gigabase Benchmark (LGB; 25,000 positions):** A benchmark constructed from games drawn from Lumbra's Gigabase [47], a large curated collection of deduplicated PGN files aggregated from multiple sources. The database includes both over-the-board (OTB) and online games, spanning a wide range of time periods, player ratings, and data sources.[1]

  We format the games with templates from LichessTemplates-Instruct and append time control information. To account for the difference between FIDE and Lichess ratings, we increment the ratings of all players by 500 for evaluation.

  In this work, we define **LGB (OTB 2020–2024)** as the subset constructed from the corresponding OTB files, and use it to evaluate generalization beyond online Lichess data.

We additionally define LIF-Aux and LGB-Aux, variants of LIF and LGB, respectively, with the same number of positions, that append temporal metadata describing the time control, remaining clock information, and previous move delay when available. These benchmarks additionally evaluate auxiliary prediction targets introduced during auxiliary-target fine-tuning (Section 6.6).

Although these benchmarks are framed as instruction-following tasks, they are ultimately evaluated based on agreement with human moves. As such, they can be interpreted as human move prediction under prompt-conditioned inputs, bridging the gap between instruction-following and conventional imitation benchmarks.

[1] Lumbra's Gigabase is a curated aggregation of games from multiple sources, including national federations, online platforms, and historical archives. Games are deduplicated and filtered (e.g., excluding very short games and low-rated play), and are distributed as separate PGN files grouped by time period, source, and game type (OTB vs. online).

7.1.3 Metadata-Conditioned Tasks

- **Maia-1 (M1; 11.3M positions):** A benchmark derived from Lichess games (December 2019) introduced by McIlroy-Young et al. [65], designed to evaluate how well a model predicts human moves across a range of skill levels. The evaluation is organized into Elo bins, enabling analysis of model performance as a function of player strength.[2]

- **Maia-1 Subset (M1-S; 11,225 positions):** A stratified random subset of M1, grouped by player Elo (rounded to the nearest hundred) and time control. This subset is used for rapid evaluation during ablation studies and training analysis. We use it to evaluate prompt-based models when full M1 evaluation is computationally prohibitive.

- **Maia-2 Rapid Benchmark (M2R; 127,852 positions):** A benchmark constructed from Lichess Rapid games (December 2019), following the evaluation protocol introduced in Maia-2 [91].

- **Allie Blitz Benchmark (ABB; 119,738 positions):** A benchmark of Lichess Blitz games from 2022 [127], with early moves and low-time moves filtered out to reduce noise.

Taken together, these benchmarks distinguish between prompt-conditioned and metadata-conditioned settings while maintaining a common evaluation objective of matching human moves.

[2]We use the publicly released "Maia KDD Testing Set" (`https://csslab.cs.toronto.edu/datasets/`), which differs from the construction described in the original paper. The paper describes test sets consisting of 10,000 games per Elo bin (1100–1900), with filtering to remove the first 10 plies and moves made under severe time pressure, yielding roughly 500,000 positions per bin.

In contrast, the released dataset spans a wider range of Elo bins (approximately 1000–2600) and contains between approximately 550,000 and 850,000 positions per bin for most rating ranges, with a much smaller number of positions at the highest ratings. Additionally, while the paper specifies filtering of low-time moves, the released dataset contains a small number of such positions (64 out of approximately 11 million total), which does not materially affect evaluation results.

We report results using this released dataset for reproducibility.

7.1.4 Evaluation Protocol Details

For benchmarks derived from human games (e.g., M1, M2R, ABB), prior work commonly discards the first 10 plies and moves made under severe time pressure (e.g., less than 30 seconds on the clock) to reduce noise from opening preparation and time scrambles.

In our evaluation, we report results both with and without these filtering criteria where applicable. This allows us to assess the sensitivity of benchmark scores to early-game moves and time pressure effects. We additionally comment on cases where differences in filtering protocols lead to discrepancies with previously reported results.

Unless otherwise stated, benchmark scores are reported to four decimal places. As each benchmark contains at least 10,000 positions, this precision is sufficient to distinguish reproducible differences without excessive rounding.

7.1.5 Evaluation Metrics

We report both **top move accuracy** (Acc@1), which measures whether the highest-probability move matches the ground-truth move, and **expected accuracy** ($\mathbb{E}$[Acc]), which measures the probability of selecting the correct move when sampling from the predicted distribution at temperature 1.

Expected accuracy is only reported for models that produce normalized probability distributions over legal moves.

In addition to move prediction accuracy, we evaluate several auxiliary targets. $\mathrm{F1_w}$ denotes the weighted F1 score for categorical targets, while MAE denotes the mean absolute error for scalar targets.

7.2 Models

We evaluate a diverse set of models spanning classical chess engines, metadata-conditioned models, and prompt-conditioned models. These models differ substantially in their input representations, conditioning mechanisms, and predicted outputs.

7.2.1 Engine-Based Models

Lc0 serves as a strong baseline for near-optimal play, in contrast to the human-like or prompt-conditioned models described in subsequent sections.

Lc0. Lc0 is an open-source neural network chess engine originally developed as a reproduction of AlphaZero [86]. It predicts both a policy over legal moves and a scalar value representing the expected game outcome. In addition, modern Lc0 models include an auxiliary head that estimates the number of plies remaining in the game.

Architecture and inputs. Two main architectures exist: earlier models use a Squeeze-and-Excitation CNN [43], while more recent models adopt the ChessFormer architecture [67, 68], which is substantially stronger. All ChessFormer (Lc0-CF) models take as input 112 feature planes encoding a sequence of 8 board states, including the current position and the previous 7 states (see Section 2.2.1).

Model variants. We evaluate several pretrained Lc0-CF models that vary in parameter count and architectural details:

- **t3:** A 0.10B parameter model using the `t3-512x15x16h-swa-2815000` weights.
- **t82:** A 0.10B parameter model using the `t82-768x15x24h-swa-7364000` weights. This model uses a simplified "arc encoding" input block.
- **BT3:** A 0.12B parameter model using the `BT3-768x15x24h-swa-2790000` weights.
- **BT4:** A 0.21B parameter model using the `BT4-1024x15x32h-swa-5000000` weights.
- **BT4-spsa:** A 0.21B parameter model refined using simultaneous perturbation stochastic approximation (SPSA) [89], using the `BT4-spsa-1740` weights.

7.2.2 Metadata-Conditioned Models

Human imitation models aim to reproduce the human player behavior rather than optimize for maximal playing strength. Unlike engine-based approaches such as Lc0, these models incorporate explicit conditioning on player skill and game context.

Maia family. The Maia models are a family of neural networks trained to predict human moves from Lichess games, conditioned on player skill. They differ primarily in how skill is represented and how game state is encoded.

Maia-1. Maia-1 consists of a family of nine fine-tuned Lc0-CNN models trained on Lichess games from 2017–2019 [65]. Each model corresponds to a specific Elo range (1100–1900, in increments of 100), and no explicit time control information is provided. An additional unofficial model trained on games from players rated 2200–2299 is also available [107].

Evaluation setup. For Maia-1, we match each position to the model corresponding to the closest rounded Elo of the active player, following the evaluation protocol used in prior work. We evaluate two configurations: one using only the official Maia-1 models, and another including the unofficial 2200 model.

Maia-2. Maia-2 is trained from scratch using a hybrid ResNet–Transformer architecture [38, 110, 91]. It encodes the current board state and conditions on player skill using discrete Elo tokens for both players. Separate models are trained for rapid and blitz time controls. In addition to predicting the next move, Maia-2 includes auxiliary prediction heads for move attributes (e.g., moved piece, origin and destination squares, captures, and checks) and a scalar value head for game outcome prediction.

Maia-3. Maia-3 adopts the Lc0-ChessFormer architecture and replaces discrete skill tokens with linearly interpolated Elo embeddings for both players [68]. It takes as input a history of recent board states, similar to Lc0, and predicts both policy and value. Maia-3 is trained exclusively on blitz games, in contrast to Maia-2, which uses separate models for rapid and blitz. At the time of writing, the code and pretrained weights for Maia-3 are not publicly available.

Allie. Allie is a language-model-based approach to human imitation, initialized from GPT-2 Medium weights and adapted to chess through a custom tokenization scheme [79, 127]. Unlike the Maia models, which operate on structured board representations, Allie represents games as sequences of tokens encoding player skill, time control, moves, and game outcomes. In our benchmark tables, we report results using the Allie-Policy configuration, which performs next-move prediction via direct sampling from the model distribution without search.

Input representation. A game is represented as a token sequence consisting of: (i) two tokens encoding the Elo of each player via linear interpolation between weak and strong skill embeddings, (ii) a token representing the time control (or an unknown token if unavailable), and (iii) a sequence of tokens representing the moves played in UCI format. If the game terminates normally (e.g., resignation or checkmate), a termination token is included, followed by an end-of-sequence token.

Outputs. Allie uses three prediction heads: (i) a next-token prediction head (typically corresponding to the next move), (ii) a scalar value head for predicting the game outcome, and (iii) a head for predicting normalized move delay (ponder time), defined as the move time truncated to a maximum of one minute and normalized using dataset statistics.

7.2.3 Prompt-Conditioned Models

Prompt-conditioned models approach chess as a sequence modeling problem over text, relying on natural-language prompts and serialized game representations rather than structured board encodings. As a result, game state, player skill, and other contextual information must be conveyed implicitly through text.

For autoregressive language models (e.g., ChessGPT and GPT-3.5 Turbo Instruct), generated moves are constrained to valid formats where possible. If a predicted move is not legal in the given position, we replace it with a uniformly sampled legal move. This post-processing step does not apply to scoring-based models such as ChessCLIP, which only consider legal moves during inference.

ChessCLIP. ChessCLIP is a CLIP-based model that scores the compatibility between board states, candidate moves, and textual prompts [27, 78]. It encodes board positions using the 112 Lc0 feature planes and conditions on textual input via a modified FiLM layer [73].

Inference. For a given position, we enumerate all legal moves, apply each move to obtain a candidate next state, and select the move corresponding to the highest similarity between the (state, move) pair and the prompt.

ChessGPT v1. ChessGPT is a large language model trained for chess reasoning and move prediction [27]. It is based on the RedPajama INCITE 3B model [113]. ChessGPT-base is obtained by continual pretraining on a mixture of text, PGN, and FEN data, while ChessGPT-chat is further fine-tuned on conversational chess data.

ChessGPT-base. We evaluate ChessGPT-base using two prompting strategies unless otherwise specified:

1. **Prompt**: The prompt followed by the PGN move list without headers: `{prompt}\n\n{pgn}`.
2. **Elo and time control PGN header**[3]: A PGN header describing both players' Elos and the time control (Appendix C.4), followed by two newlines and the PGN move list.

[3]We use *PGN header* to refer to PGN-style headers that may not fully conform to the PGN specification.

To ensure reliable move extraction, we use `outlines` [115] to constrain the generated output with a predefined regular expression. For ChessGPT-base, the move number may optionally be preceded by a space, and both the move number and trailing punctuation are optional.

ChessGPT-chat. We evaluate ChessGPT-chat using three prompting strategies unless otherwise specified:

1. **Simple, duplicate**: We prompt ChessGPT-chat with `{prompt}\n\nPGN: {pgn}\n\nJust write your move.` Applying the chat template twice—once without the generation prompt and once with it—improves performance on the Lichess openings benchmark.
2. **Simple, original**: The same prompt as above, except the chat template is applied only once with the generation prompt.
3. **General policy**: A prompt adapted from Feng et al., consisting of a prompt to play chess in a given position, a PGN header with certain fields masked by "??", the PGN move list, and a final instruction to produce the next move.

As with ChessGPT-base, we use `outlines` [115] to constrain generation. For ChessGPT-chat, the output must match a move number, one or three periods, a space, and a valid SAN move.

GPT-3.5 Turbo Instruct. GPT-3.5 Turbo Instruct [70] is a proprietary language model that demonstrates strong chess performance when prompted appropriately.

Prompting strategies. We evaluate the following configurations:

1. **GM PGN header**: A PGN header indicating a game between strong players (*i.e.*, Magnus Carlsen and Garry Kasparov), followed by the move list. We use the prompt from the open-sourced implementations accompanying Carlini [12] and Allie [127], which differs slightly from the version described in the Carlini blog and reproduced in the Allie appendix (Appendix C.3).
2. **GM PGN header + prompt**: A natural-language prompt inserted between the header and the move list.
3. **Original PGN header**: The original PGN header, followed by the move list.
4. **Elo and time control PGN header**: A PGN header specifying player ratings and time control (Appendix C.4), followed by the move list.

UniMaia. UniMaia is the final prompt-conditioned policy model obtained from the full training run described in Section 6.5. The model combines a frozen Lc0-based chess policy network with a LoRA-adapted text encoder and a ControlNet-style conditioning mechanism, enabling natural-language control over gameplay while preserving the inductive biases of the underlying chess representation.

UniMaia-Aux. UniMaia-Aux is obtained by further fine-tuning the final UniMaia checkpoint using auxiliary temporal metadata and auxiliary prediction heads, as described in Section 6.6. In addition to next-move prediction, UniMaia-Aux is trained to predict auxiliary behavioral and game-state targets, including the terminal value, termination reason, move delay, and remaining plies. During evaluation, we append time-control metadata to the prompts used for all benchmarks. During evaluation, UniMaia-Aux uses temporally augmented prompts for LIF-Aux and LGB-Aux, as described in Section 7.1.2.

Model-specific prompting strategies. For prompt-conditioned benchmarks, different prompting strategies are used depending on the model. For ChessGPT-base, we use the standard prompt templates, while for metadata-conditioned benchmarks we instead provide Elo and time control information through the PGN header. For ChessGPT-chat, we use the duplicate simple prompt for opening benchmarks and the general policy prompt for all other benchmarks. For GPT-3.5 Turbo Instruct, we use a grandmaster-style PGN header for opening benchmarks and an Elo- and time control-conditioned header for all other benchmarks.

7.2.4 Model Comparison

The models described above differ in how they represent game state, incorporate contextual information such as player skill and time control, and produce outputs. Tables 7.1 and 7.2 compare model inputs and outputs, while Tables 7.3 and 7.4 summarize training scale and configuration.

Table 7.1 highlights three distinct approaches to incorporating game context. Engine-based models such as Lc0 rely exclusively on structured board representations and do not condition on player-specific information. Human imitation models introduce explicit control signals, such as Elo and time control, either through discrete tokens (Maia-2) or continuous interpolation in embedding space (Allie and Maia-3), enabling them to model

Table 7.1: Comparison of model inputs. Here, "board" denotes a full game state, including piece placement, castling rights, and en passant. The 112 Lc0 feature planes (see Section 2.2.1) encode a sequence of 8 board states, including the current position and the previous 7 states.

Model	Game state	Elo	Time control	Prompt	Move delay
Lc0	112 Lc0 planes	–	–	–	–
Maia-1	112 Lc0 planes	Separate models	–	–	–
Maia-2	Current board	Discrete tokens	Separate models	–	–
Maia-3	112 Lc0 planes	Interp. embeddings	–	–	–
Allie	Move history	Interp. embeddings	Discrete tokens	–	✓
ChessCLIP	112 Lc0 planes	Via prompt	Via prompt	✓	–
ChessGPT	Via prompt	Via prompt	Via prompt	✓	–
GPT-3.5	Via prompt	Via prompt	Via prompt	✓	–
UniMaia	112 Lc0 planes	Via prompt	Via prompt	✓	–
UniMaia-Aux	112 Lc0 planes	Via prompt	Via prompt	✓	✓

Table 7.2: Comparison of model outputs, including predicted quantities such as move policy, game result (value), and auxiliary behavioral signals.

Model	Policy	Value	Moves left	Move delay	Resignation	Termination	Text
Lc0	✓	✓	✓	–	–	–	–
Maia-1	✓	✓	–	–	–	–	–
Maia-2	✓	✓	–	–	–	–	–
Maia-3	✓	✓	–	–	–	–	–
Allie	✓	✓	–	✓	✓	–	–
ChessCLIP	✓	–	–	–	–	–	–
ChessGPT	✓	–	–	–	–	–	✓
GPT-3.5	✓	–	–	–	–	–	✓
UniMaia	✓	–	–	–	–	–	–
UniMaia-Aux	✓	✓	✓	✓	✓	✓	–

Table 7.3: Comparison of training configurations for structured-input models, including batch size, dataset scale, and parameter count. For UniMaia, parameters for the text and board encoders are reported separately.

Model	Batch size	Grad. acc.	# GPUs	# Positions	# Parameters
Maia-1	1,024	–	1	409.6M	92M
Maia-2	8,192	–	2	9.15B	23M
Maia-3	128	4	8	1.024B	5M, 23M, 79M
ChessCLIP	512	–	8	1.3M	67M
UniMaia	64	–	2	66.978M	3B+404M
UniMaia-Aux	64	–	2	71.565M	3B+515M

Table 7.4: Comparison of training configurations for language-based models. ChessGPT is trained for one epoch; however, the exact dataset size is unavailable due to dataset release restrictions.

Model	Batch size	Grad. acc.	# GPUs	Context window	# Tokens	# Parameters
Allie	32	–	8	512	262.144B	355M
ChessGPT-base	3	8	8	1,024	?	3B
ChessGPT-chat	4	8	8	1,024	?	3B

player-dependent behavior. In contrast, language-based models encode all contextual information implicitly through natural-language prompts, requiring the model to infer factors such as skill level and playing style from textual input alone.

UniMaia bridges these paradigms by jointly conditioning on both structured board representations and natural-language prompts. Unlike prior approaches that rely on predefined control signals, this formulation enables flexible and semantically rich control over model behavior while preserving the inductive biases of domain-specific encodings.

Table 7.2 further illustrates that most models focus primarily on predicting the next move (policy), with some additionally estimating game results through a value head. Auxiliary predictions, such as moves-left estimation (Lc0) or behavioral signals like move delay and resignation (Allie), are relatively uncommon. While UniMaia primarily focuses on prompt-conditioned policy prediction, UniMaia-Aux (Section 6.6) additionally incorporates auxiliary heads for game result prediction, moves-left estimation, termination prediction, and move delay prediction conditioned on temporal game metadata.

Tables 7.3 and 7.4 also highlight substantial differences in computational scale across model families. Structured-input chess models generally train with much larger batch sizes and more efficient board encodings, while language-conditioned approaches incur substantially higher computational overhead due to autoregressive text processing and larger backbone models. UniMaia occupies an intermediate position between these paradigms: although it retains efficient structured board representations through the frozen Lc0 backbone, conditioning on a frozen 3B-parameter language model substantially increases training cost relative to prior engine-based and metadata-conditioned approaches.

7.3 Results

Across all benchmarks, the results reveal a consistent trade-off between modeling approaches. Language-based models achieve the strongest performance on opening-heavy tasks, metadata-conditioned models (e.g., Allie) perform best on human move prediction and expected accuracy, while UniMaia attains the highest top-move accuracy on prompt-conditioned benchmarks requiring adaptation to natural-language inputs.

We report results separately for prompt-conditioned and metadata-conditioned benchmarks to reflect differences in input structure and evaluation settings.

Where applicable, we report both scores from the original papers and our reproduced scores. Prior works often evaluate models on different benchmark suites, use distinct prompting strategies, or omit newer benchmarks entirely, making broad comparisons across

Table 7.5: UniMaia training configuration.

Setting	**Value**
Training data	Lichess 2013–2022
Evaluation data	Lichess 2023
Backbone	BT4 Lc0-CF
Architecture	ControlNet + controllable policy head
Frozen backbone	Yes
Text adaptation	LoRA & rsLoRA & LoRA+
LoRA rank	16
Batch size	64
Peak learning rate	5×10^{-5}
Warmup steps	1,024
Optimizer	NorMuon + AdamW
Weight decay	0.01
Optimizer ϵ	1×10^{-8}
Gradient clipping	1.0
Downsampling	Unimax by time control
Hardware	2× RTX Pro 6000

Table 7.6: UniMaia-Aux training configuration.

Setting	**Value**
Initialization	UniMaia 2022 checkpoint
Auxiliary training data	Lichess 2018
Backbone	BT4 Lc0-CF
Architecture	ControlNet + controllable policy head
Frozen backbone	Yes
LoRA rank	16
Batch size	64
Peak learning rate	5×10^{-5}
Warmup steps	1,024
Optimizer	NorMuon + AdamW
Auxiliary losses	Outcome, moves-left, termination, move delay
Move-delay source	PGN `%clk` annotations

model classes difficult. For example, some language-based models are evaluated primarily on opening reconstruction or move-scoring tasks, whereas metadata-conditioned models are often evaluated only on human move prediction benchmarks. Although certain large language models (e.g., GPT-3.5 Turbo Instruct) have also been evaluated on human move prediction, evaluations across model classes remain inconsistent in both benchmark coverage and prompting methodology.

To enable more direct comparison across approaches, all reproduced results are obtained using our unified benchmark harness, which standardizes preprocessing, prompting, legality handling, and evaluation metrics across models whenever possible. This enables broader comparison between language-based, metadata-conditioned, and prompt-conditioned chess models under a common evaluation framework. Full results for certain large language models (e.g., GPT-3.5 Turbo Instruct and ChessGPT variants) are provided in Appendix D.

Comparability across model classes. Prompt-conditioned benchmarks use randomly sampled templates that may condition on varying subsets of metadata, whereas metadata-conditioned benchmarks use fixed templates with a consistent set of metadata fields. Consequently, different model classes receive different input information across benchmark categories, and results should be interpreted as reflecting performance under each conditioning regime rather than as strictly controlled comparisons across all models. In addition, model-specific prompting strategies are used to align each model with the conditioning mechanism being evaluated, further emphasizing that comparisons are made within, rather than across, conditioning regimes.

7.3.1 Prompt-Conditioned

Tables 7.7, 7.8, 7.9, and 7.10 summarize performance on the prompt-conditioned benchmarks. Full expected-accuracy results are reported in Appendix D.2.1.

Overall, UniMaia achieves the strongest performance across prompt-conditioned tasks, attaining the highest Acc@1 on LIF-D, LIF, LIF-T10, and LGB. The only exception is opening instruction-following (LOB-P and LOB-C), where ChessGPT-base achieves the highest scores, with UniMaia consistently ranking second.

When excluding the first 10 plies to reduce the influence of opening memorization, UniMaia retains the best overall performance on over-the-board prediction tasks (LGB) and remains highly competitive across the remaining benchmarks, consistently ranking second and within 0.01 of the top score.

Table 7.7: **Prompt-conditioned** benchmark results, measured by **Acc@1**. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models. †Opening benchmarks evaluated with the regular prompt. Remaining benchmarks evaluated with the Elo and time control PGN header. ‡Opening benchmarks evaluated with the duplicate simple prompt. Remaining benchmarks evaluated with the general policy prompt. §Opening benchmarks evaluated with the GM PGN header. Remaining benchmarks evaluated with the Elo and time control PGN header.

Model	LOB-P	LOB-C	LIF-D	LIF	LIF-T10	LGB
Lc0-CF (t3)	0.5010	0.5027	0.3244	0.3942	0.3180	0.4772
Lc0-CF (t82)	0.5011	0.5024	0.3214	0.3908	0.3148	0.4749
Lc0-CF (BT3)	0.5025	0.5041	0.3252	0.3924	0.3162	0.4771
Lc0-CF (BT4)	0.5025	0.5039	0.3246	0.3924	0.3132	0.4770
Lc0-CF (BT4-spsa)	0.4744	0.4757	0.3217	0.3902	0.3122	0.4643
Maia-1*	0.5582	0.5602	0.4411	0.4797	0.4076	0.4857
Maia-1**	0.5709	0.5729	0.4411	0.4830	0.4085	0.5034
Maia-2 (Blitz)	0.4058	0.4040	0.4144	0.4763	0.4051	0.4452
Maia-2 (Rapid)	0.4129	0.4123	0.4207	0.4758	0.3943	0.4484
Allie-Policy	0.6150	0.6164	0.4658	0.5390	0.4662	0.5572
ChessCLIP	0.2730	0.2696	0.0895	0.0918	0.0747	0.1040
ChessGPT-base†	**0.7998**	**0.8955**	0.3808	0.4464	0.4153	0.4788
ChessGPT-chat‡	0.3417	0.2272	0.1494	0.1637	0.1480	0.1501
GPT-3.5§	0.5748	0.6431	0.4457	0.5312	0.4778	0.5514
UniMaia	<u>0.7094</u>	<u>0.8162</u>	**0.4748**	**0.5587**	**0.4954**	**0.6158**
UniMaia-Aux	0.7071	0.7801	<u>0.4664</u>	<u>0.5481</u>	<u>0.4915</u>	<u>0.6086</u>

Table 7.8: Prompt-conditioned benchmark results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models. †Evaluated with the Elo and time control PGN header. ‡Evaluated with the general policy prompt.

Model	LIF-D	LIF	LIF-T10	LGB
Lc0-CF (t3)	0.3360	0.4052	0.3290	0.5064
Lc0-CF (t82)	0.3329	0.4013	0.3265	0.5012
Lc0-CF (BT3)	0.3364	0.4036	0.3272	0.5054
Lc0-CF (BT4)	0.3357	0.4037	0.3232	0.5067
Lc0-CF (BT4-spsa)	0.3366	0.4046	0.3249	0.5021
Maia-1*	0.4532	0.4845	0.4106	0.4515
Maia-1**	0.4532	0.4871	0.4114	0.4694
Maia-2 (Blitz)	0.4412	0.5021	0.4278	0.4850
Maia-2 (Rapid)	0.4467	0.5019	0.4187	0.4825
Allie-Policy	**0.4812**	**0.5492**	0.4735	0.5452
ChessCLIP	0.0862	0.0813	0.0633	0.0567
ChessGPT-base†	0.3813	0.4440	0.4158	0.4339
ChessGPT-chat‡	0.1490	0.1557	0.1491	0.1564
GPT-3.5†	0.4608	0.5444	**0.4914**	0.5569
UniMaia	<u>0.4712</u>	<u>0.5457</u>	<u>0.4792</u>	<u>0.5638</u>
UniMaia-Aux	0.4657	0.5378	0.4787	**0.5692**

Table 7.9: **Prompt-conditioned** results, measured by $\mathbb{E}$[**Acc**]. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	LOB-P	LOB-C	LIF-D	LIF	LIF-T10	LGB
Allie-Policy	0.4905	0.4906	0.3510	0.4104	0.3525	0.4480
UniMaia	<u>0.5899</u>	**0.7140**	<u>0.3522</u>	<u>0.4297</u>	**0.3678**	<u>0.4834</u>
UniMaia-Aux	**0.5973**	<u>0.6823</u>	**0.3606**	**0.4331**	<u>0.3600</u>	**0.4891**

Table 7.10: **Prompt-conditioned** results, measured by $\mathbb{E}$[**Acc**]. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	LIF-D	LIF	LIF-T10	LGB
Allie-Policy	**0.3652**	<u>0.4215</u>	**0.3607**	<u>0.4393</u>
UniMaia	0.3444	0.4129	<u>0.3507</u>	0.4252
UniMaia-Aux	<u>0.3592</u>	**0.4218**	0.3507	**0.4521**

In terms of expected accuracy, either UniMaia or UniMaia-Aux achieves the strongest performance across nearly all prompt-conditioned benchmarks. When all moves are included, UniMaia-Aux attains the highest expected accuracy on LOB-P, LIF-D, LIF, and LGB, whereas UniMaia performs best on LOB-C and LIF-T10. The largest gap occurs on LOB-C, where UniMaia substantially outperforms UniMaia-Aux, suggesting stronger memorization of canonical opening continuations.

When excluding the first 10 plies, UniMaia-Aux remains highly competitive and frequently attains the strongest expected accuracy, outperforming both UniMaia and Allie-Policy on LIF, LGB, and tying UniMaia on LIF-T10. Although Allie-Policy still achieves the strongest expected accuracy on several benchmarks, the margins remain comparatively small.

Although UniMaia-Aux slightly underperforms UniMaia in terms of Acc@1 on the original prompt-conditioned benchmarks, the gap remains comparatively small across all evaluation settings. This likely reflects the fact that UniMaia-Aux is additionally conditioned and fine-tuned to leverage temporal metadata during prediction. Consequently, when evaluated on benchmarks that do not provide the full temporal information used during auxiliary-target fine-tuning, the model cannot fully utilize the additional conditioning signals it has learned to incorporate. Nevertheless, the expected-accuracy results suggest that the auxiliary temporal conditioning improves the calibration and distributional structure of the policy, even when the corresponding temporal metadata is unavailable during evaluation.

Both ChessGPT models exhibit a tendency to generate illegal moves, which are replaced with uniformly sampled legal moves during evaluation. This replacement introduces additional noise into the predictions, lowering both top-move and expected accuracy.

Finally, although ChessCLIP occasionally assigns moderate probability mass to the correct move, it rarely ranks it as the top prediction. Its output distribution is typically diffuse after the softmax transformation, which leads to low expected accuracy despite assigning non-trivial probability to correct moves.

Taken together, these results highlight a trade-off across model classes: language-based models achieve the strongest performance on opening-heavy tasks, metadata-conditioned models excel at human move prediction, while UniMaia performs best on prompt-conditioned benchmarks. This suggests that different modeling approaches specialize in distinct aspects of the task, rather than uniformly dominating across all evaluation settings.

Auxiliary Temporal Metadata

Tables 7.11 and 7.12 show the benchmark results for UniMaia-Aux. Unlike the previous benchmark results, the LIF-Aux and LGB-Aux evaluations use temporally augmented prompts that append temporal metadata, including the time control, remaining clock information, and previous move delay when available, as described in Section 7.1.2. These additional temporal signals more closely match the conditioning setup used during auxiliary-target fine-tuning.

In addition to move prediction accuracy, we evaluate the auxiliary targets introduced in Section 6.6.3. We report weighted F1 ($F1_w$) for categorical targets and MAE for scalar targets.

Table 7.11: Auxiliary targets results. Best values are in bold and second-best values are underlined.

Benchmark	Model	Acc@1↑	$\mathbb{E}$[Acc] ↑	$F1_w$ ↑			MAE↓		
				Result	Termination	Resigned	$N-i-1$	V	Δt_a
LIF-Aux	Allie-Policy	<u>0.5390</u>	<u>0.4104</u>	–	–	<u>0.6409</u>	–	**0.2176**	<u>4.8651</u>
	UniMaia-Aux	**0.5549**	**0.4318**	**0.6718**	**0.5448**	**0.7279**	**21.3624**	<u>0.3229</u>	**4.0252**
LGB-Aux	Allie-Policy	<u>0.5572</u>	<u>0.4480</u>	–	–	<u>0.0933</u>	–	**0.3711**	<u>127.7715</u>
	UniMaia-Aux	**0.5872**	**0.4637**	**0.3814**	**0.6494**	**0.6643**	**35.6365**	<u>0.4092</u>	**124.9658**

UniMaia-Aux outperforms Allie-Policy on both LIF-Aux and LGB-Aux regardless of whether the first 10 plies are omitted from evaluation. In particular, the gains are stronger when the opening plies are omitted, suggesting that the auxiliary temporal metadata improves human behavioral modeling beyond opening memorization.

Table 7.12: Auxiliary targets results. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined.

Benchmark	Model	Acc@1↑	$\mathbb{E}$[Acc] ↑	$F1_w$ ↑ Result	$F1_w$ ↑ Termination	$F1_w$ ↑ Resigned	MAE↓ $N-i-1$	MAE↓ V	MAE↓ Δt_a
LIF-Aux	Allie-Policy	<u>0.5492</u>	<u>0.4215</u>	–	–	<u>0.6388</u>	–	**0.2139**	<u>5.0358</u>
	UniMaia-Aux	**0.5495**	**0.4275**	**0.6618**	**0.5431**	**0.7270**	**20.4985**	<u>0.3258</u>	**4.3000**
LGB-Aux	Allie-Policy	<u>0.5452</u>	<u>0.4393</u>	–	–	<u>0.1231</u>	–	**0.3337**	<u>122.3217</u>
	UniMaia-Aux	**0.5622**	**0.4464**	**0.3824**	**0.6135**	**0.6223**	**26.3534**	<u>0.3981</u>	**118.3532**

UniMaia-Aux also achieves a substantially higher weighted F1 score for resignation prediction on over-the-board games (LGB), indicating stronger generalization.

Overall, UniMaia-Aux outperforms Allie on all auxiliary targets except value prediction. We note that UniMaia-Aux is trained on substantially less data than Allie, and that its scores continue to improve throughout training. We also note that Allie uses an MSE loss for value prediction, whereas UniMaia-Aux uses a cross-entropy loss over discrete game results, which may contribute to its improved MAE score.

7.3.2 Metadata-Conditioned

Table 7.13 reports results on metadata-conditioned benchmarks, with the first 10 plies excluded to mitigate opening memorization effects. For ABB, we additionally include reported results from prior work where available. Because the Allie and Maia-3 papers use the same evaluation protocol, these values are broadly comparable, although all rankings and comparisons in this table are based on our reproduced results under a unified benchmark harness.

In this setting, UniMaia achieves competitive performance across all benchmarks, consistently ranking second behind Allie. While Allie attains the highest accuracy overall, UniMaia closely follows across ABB, M1-S, and M2R.

Notably, this performance is achieved despite UniMaia being optimized for prompt-conditioned tasks, suggesting that language-conditioned policy modulation does not significantly degrade performance on conventional, metadata-driven prediction tasks.

Table 7.13: **Metadata-conditioned** benchmark prediction results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models. †Evaluated with the Elo and time control PGN header. ‡Evaluated with the general policy prompt. §Evaluated with the GM PGN header.

Model	ABB (ours)	ABB (reported)	M1-S	M1	M2R
Lc0-CF (t3)	0.4179	–	0.4318	0.4351	0.4136
Lc0-CF (t82)	0.4113	–	0.4285	0.4289	0.4062
Lc0-CF (BT3)	0.4157	–	0.4310	0.4340	0.4127
Lc0-CF (BT4)	0.4133	–	0.4255	0.4305	0.4079
Lc0-CF (BT4-spsa)	0.4145	–	0.4223	0.4303	0.4091
Maia-1*	0.5078	0.516 ± 0.001	0.5094	0.5135	0.5135
Maia-1**	0.5089	–	0.5149	0.5173	0.5138
Maia-2 (Blitz)	0.5230	0.520 ± 0.001	0.5246	<u>0.5300</u>	0.5284
Maia-2 (Rapid)	0.5204	–	0.5253	0.5294	0.5311
Maia-3-79M (Blitz)	–	**0.571 ± 0.001**	–	–	–
Allie-Policy	**0.5643**	<u>0.557 ± 0.001</u>	**0.5733**	**0.5717**	**0.5598**
ChessCLIP	0.0813	–	0.0813	0.0803	0.0838
ChessGPT-base†	0.4336	–	0.4375	–	0.4305
ChessGPT-chat‡	0.1697	–	0.1719	–	0.1647
GPT-3.5§	0.5410	0.537 ± 0.001	0.5565	–	0.5390
GPT-3.5†	0.5498	–	<u>0.5667</u>	–	0.5489
UniMaia	<u>0.5548</u>	–	<u>0.5667</u>	–	<u>0.5536</u>
UniMaia-Aux	0.5471	–	0.5624	–	0.5493

Table 7.14: **Metadata-conditioned** results, measured by $\mathbb{E}$[**Acc**]. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	ABB	M1-S	M1	M2R
Maia-2 (Blitz)	0.4102	0.4144	<u>0.4166</u>	0.4137
Allie-Policy	**0.4409**	<u>0.4476</u>	**0.4475**	<u>0.4354</u>
UniMaia	0.4224	0.4317	–	0.4182
UniMaia-Aux	<u>0.4382</u>	**0.4488**	–	**0.4427**

For completeness, results including the first 10 plies are provided in Appendix D.1. Aside from Maia-2 and UniMaia, scores remain largely unchanged. UniMaia benefits slightly from including opening moves, while Maia-2 shows a decrease, likely reflecting differences in training data composition.

For metadata-conditioned benchmarks, either Allie-Policy or UniMaia-Aux achieves the highest expected accuracy across the evaluated settings. In particular, UniMaia-Aux outperforms all other models on M1-S and M2R, while Allie attains the strongest performance on ABB and M1. Although UniMaia slightly trails both models in expected accuracy, it remains competitive despite being primarily optimized for prompt-conditioned tasks. Full expected-accuracy results are reported in Appendix D.2.2.

Interestingly, UniMaia-Aux consistently improves upon UniMaia in terms of expected accuracy on metadata-conditioned benchmarks, despite slightly lower Acc@1 scores. This further suggests that auxiliary temporal conditioning improves the calibration and distributional structure of the policy, even in settings that do not explicitly provide the full temporal metadata used during auxiliary-target fine-tuning.

7.3.3 Performance by Elo Range

As we can see in Figures 7.1 and 7.2, UniMaia has a decent accuracy in all Elo combinations for both prompt-conditioned and metadata-conditioned benchmarks. The plots for the remaining benchmarks, as well as the size heatmaps for all benchmarks, are provided in Appendix D.3.

Across both top-move accuracy and expected accuracy, performance generally increases with the Elo of both players. This trend reflects the greater consistency of higher-rated players, whose moves are more predictable, whereas lower-rated players exhibit more stochastic and diverse play, making accurate prediction inherently more difficult.

This dependence on predictability also helps explain differences in the structure of the heatmaps. In particular, the LIF heatmap exhibits greater variability across Elo combinations compared to the ABB heatmap. This difference arises from the prompt construction: LIF uses randomly sampled prompt templates that condition on varying subsets of metadata, introducing additional stochasticity into the evaluation. In contrast, ABB uses a fixed prompt template with a consistent set of metadata fields, resulting in more stable and structured performance patterns across Elo ranges.

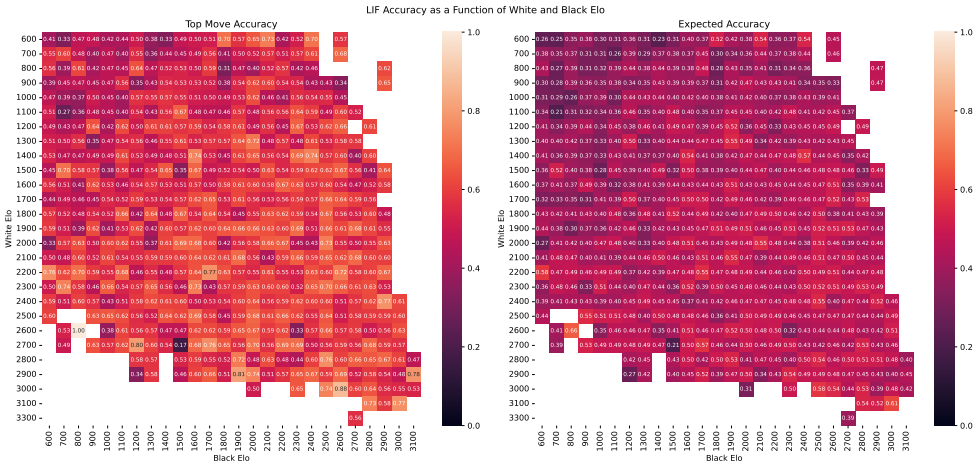

Figure 7.1: LIF accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy.

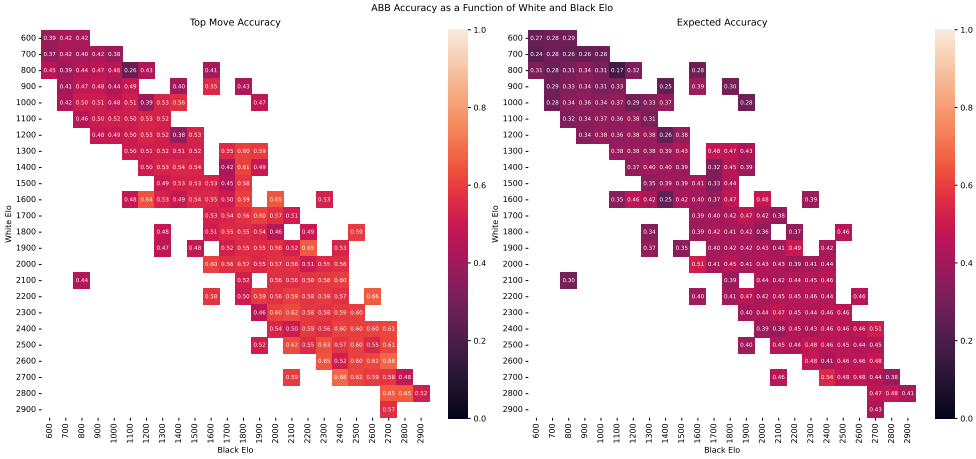

Figure 7.2: ABB accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy.

Chapter 8

Analysis of Trained Model

We analyze the trained UniMaia model to characterize how language conditioning influences its behavior. In particular, we examine whether the model produces coherent, interpretable, and controllable policies across varying inputs, and identify the mechanisms by which these behaviors arise.

8.1 Policy Continuity Across Elo

Since the Elo tokens are discrete, we aim to verify that the policy varies smoothly as Elo increases, while still exhibiting meaningful differences between distant Elo values.

To this end, we analyze the distributions of the top-5 predicted moves across Elo values, which provide a direct and interpretable view of how the policy evolves.

To complement this, we also compute the cosine similarity and the L2 norm of the difference between policies at adjacent Elos, which provide quantitative measures of smoothness. As these metrics largely corroborate the trends observed in the move distributions, we defer the corresponding plots to Appendix B.2.

As mentioned in Section 2.1.3, most of the training data lies between 600 and 3400 Elo. Nevertheless, we evaluate Elo values in the range 0–3400 to assess behavior under out-of-distribution conditions.

We first evaluate policy continuity across positions by considering four representative game stages (early opening, late opening, middlegame, and endgame). Each configuration is defined by a distinct PGN position and a corresponding prompt template. The full PGNs and prompt templates are provided in Appendix B.1.1.

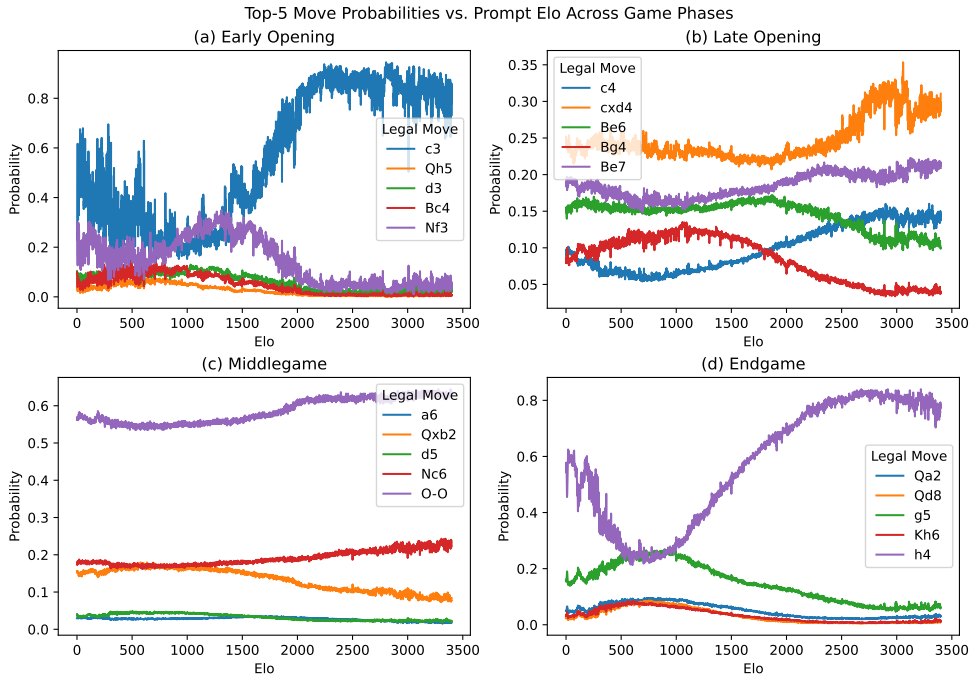

Figure 8.1: Top-5 move probabilities predicted by UniMaia as a function of Elo. Top-left: early opening. Top-right: late opening. Bottom-left: middlegame. Bottom-right: endgame.

The distributions exhibit a clear global trend as a function of Elo, indicating smooth variation (Figure 8.1). However, local fluctuations are present, resulting in a mildly jagged profile, similar to typical training loss curves. Interestingly, for Elo values below 600, the predicted distribution begins to resemble that of stronger players. This likely reflects the influence of the underlying base policy, derived from a strong engine, which dominates in regions with limited training data.

Consistent with this observation, the cosine similarity between adjacent Elo values remains high across the entire range, while the L2 norm of the difference between policies remains low (Appendix B.2). Together, these results support the conclusion that the policy varies smoothly with respect to Elo.

We next evaluate policy continuity under opening conditioning by fixing the initial position and varying the prompt. Given the model's sensitivity to opening choice in the early game, we consider four opening-conditioned prompts corresponding to the King's Pawn, Queen's Pawn, English, and Zukertort openings. The full prompt templates are provided in Appendix B.1.2.

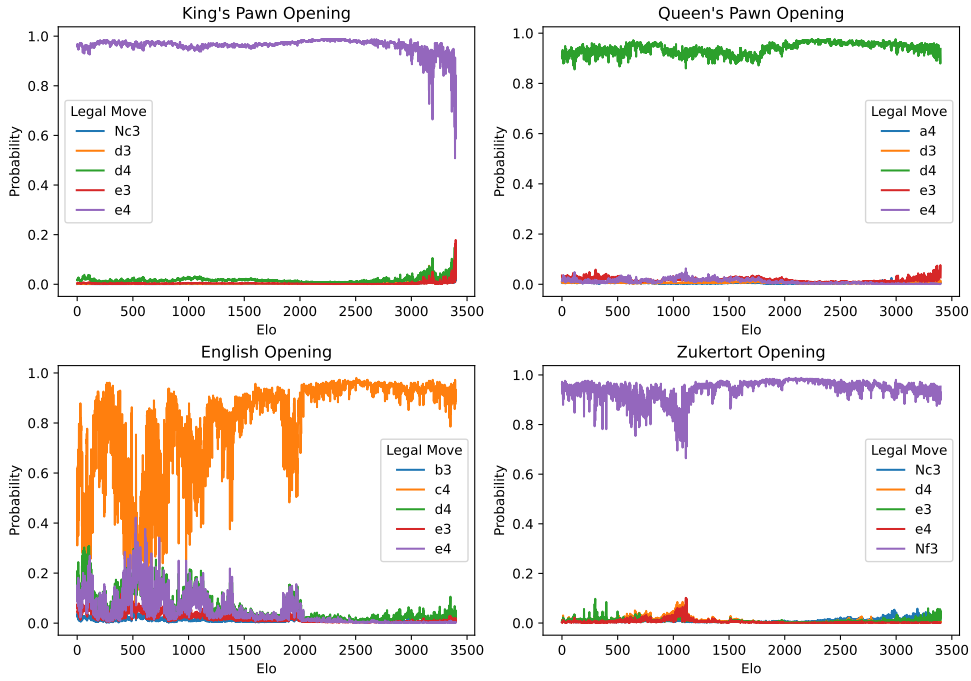

Figure 8.2: Top-5 move probabilities predicted by UniMaia as a function of Elo for opening-conditioned prompts. Top-left: King's Pawn Opening. Top-right: Queen's Pawn Opening. Bottom-left: English Opening. Bottom-right: Zukertort Opening.

For most openings, the top-move probability remains largely invariant with respect to Elo. In contrast, the English Opening exhibits noticeable variation, particularly in the Elo range 400–1300, likely due to the presence of common transpositions. One possible explanation is that lower-rated players may reach such positions via transpositions rather than deliberate opening choice, although this remains speculative.

Overall, these results indicate that UniMaia exhibits consistent variation in its policy as a function of Elo, reflecting continuity across ratings. However, this variation is not uniformly smooth, with noticeable local fluctuations – particularly in early-game positions and certain openings – indicating sensitivity to contextual factors such as position and opening structure.

8.2 Mechanistic Interpretability

To analyze the model's properties, we examine its behavior on the first 256 entries of LIF.

8.2.1 Layer-wise Contribution

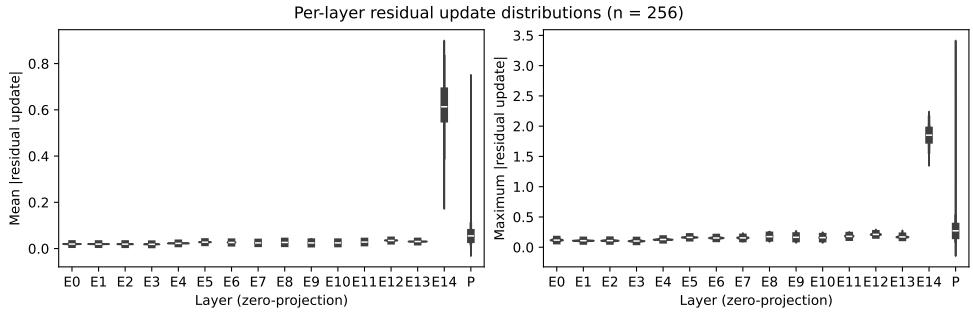

Figure 8.3: Distribution of the mean and maximum absolute residual updates for each encoder layer and the controllable policy head.

We extract the outputs of the zero-initialized projection layers (which we refer to as ControlNet residual updates) for each encoder layer in the ControlNet branch, as well as for the controllable policy head. This allows us to quantify the magnitude of the modifications introduced to the base Lc0 model. We track both the mean and maximum absolute residual updates and plot their distributions in Figure 8.3. While the residual updates for encoder layers 0–13 remain small, those for layer 14 and the controllable policy head are substantially larger.

This pattern suggests that the ControlNet branch operates in a largely *localized* manner, concentrating its influence in the final encoder layer and the policy head rather than uniformly modifying all intermediate representations. This behavior is consistent with the zero-initialization design: earlier layers remain close to the pretrained Lc0 representations, while later layers act as the primary interface for controllable adjustments.

Unlike the earlier layers, both layer 14 and the controllable policy head exhibit relatively high variance in their residual updates. As shown in Figure 8.4, the controllable policy head tends to contribute more during the early opening, while layer 14 becomes increasingly influential later in the game.

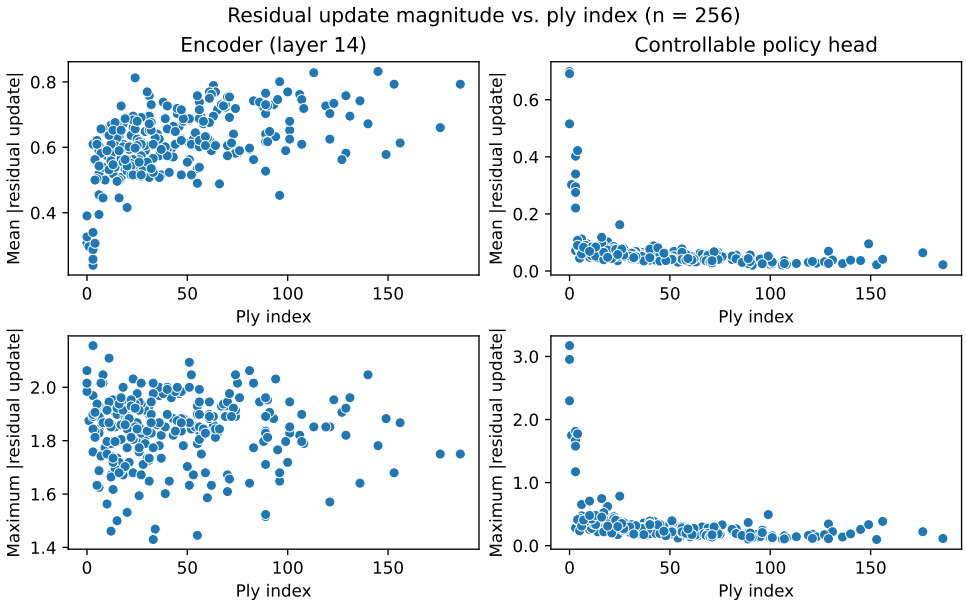

Figure 8.4: Mean and maximum absolute residual updates as a function of ply index for layer 14 and the controllable policy head.

One possible explanation is that the earlier encoder layers are more strongly constrained by the architectural biases of the base Lc0-CF model, such as the geometric attention bias (GAB) [68, 67], which enforces structured, position-dependent representations. As a result, ControlNet residual updates in these layers remain small, preserving pretrained chess-specific features.

In contrast, later components, particularly the final encoder layer and the controllable policy head, interface more directly with the policy output and may therefore admit greater flexibility. This reduced structural constraint could allow the ControlNet branch to introduce larger modifications, effectively concentrating controllable adjustments closer to the output. The observed variation across ply index may reflect differences in how these adjustments are applied at different stages of the game.

While this hypothesis is consistent with the observed distribution of residual updates, we do not directly verify the role of GAB or layer-specific constraints in this work.

We define the residual concentration ratio as the ratio of the maximum absolute residual update to the mean absolute residual update. As shown in Figure 8.5, this ratio is approximately 6 for encoder layers 0–13, approximately 3 for encoder layer 14, and

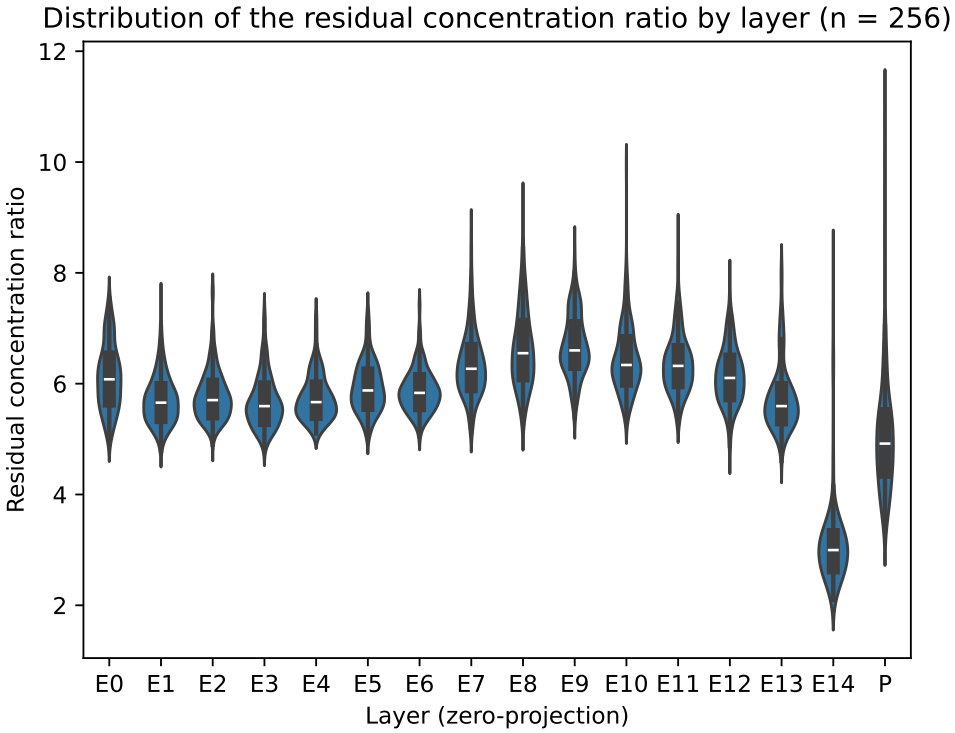

Figure 8.5: Residual concentration ratio for each encoder layer and the controllable policy head.

approximately 5 for the controllable policy head. This suggests that updates in the early encoder layers and the controllable policy head are more concentrated in a small number of dimensions, whereas updates at encoder layer 14 are comparatively more distributed.

This difference suggests that the ControlNet branch applies sparse, high-impact modifications in the early encoder layers and at the controllable policy head, while encoder layer 14 exhibits comparatively more distributed adjustments.

Because the residual updates for layers 0–13 are small in magnitude, it is unclear to what extent the ControlNet branch meaningfully alters the internal representations. To assess this, we measure the cosine similarity between the encoder representations of the base Lc0 model and UniMaia.

The cosine similarities have a mean of 0.551 and a standard deviation of 0.075, indicating moderate similarity between the representations. To analyze the relationship between cosine similarity and mean Elo, we compute the Pearson correlation coefficient.

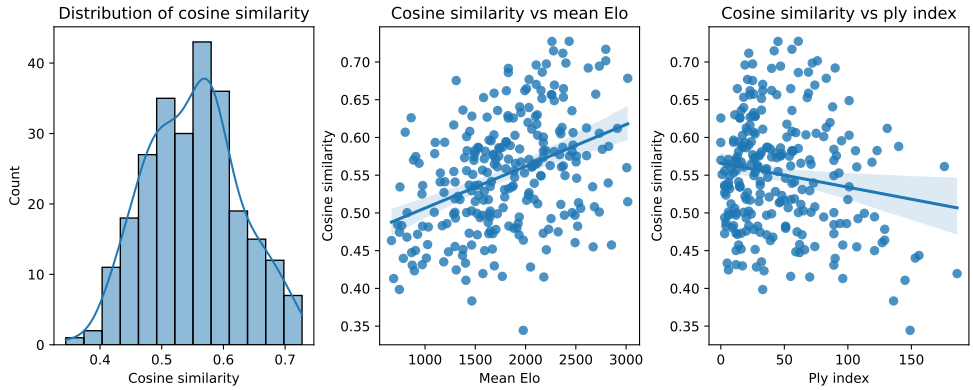

Figure 8.6: Left: Distribution of cosine similarity between flattened encoder layer 13 outputs of UniMaia and Lc0. Middle: Cosine similarity as a function of mean player . Right: Cosine similarity as a function of ply index.

We observe a moderate positive correlation ($r = 0.396$, $p = 4.70 \times 10^{-11}$), indicating that higher-rated games tend to induce representations that are more similar to the base Lc0 model. This is consistent with the interpretation that stronger play aligns more closely with the pretrained policy, while lower-rated or stylistically distinct play requires larger deviations introduced by the ControlNet branch. Interestingly, this correlation is weaker than that observed with the individual player ratings (White: 0.281, Black: 0.349). Additionally, the ply index exhibits a weak negative correlation with cosine similarity ($r = -0.157$, $p = 0.0119$), suggesting that representations diverge slightly from the base model as the game progresses.

8.2.2 Cross-attention Sinks

To assess the impact of attention sinks [117], we compute the proportion of entries for which the largest attention weight is assigned to the first token. We first average attention weights over all heads and square tokens, and then compute this proportion. Note that the ChessGPT tokenizer does not use a beginning-of-sequence token; the first token is part of the prompt itself.

We observe a clear bifurcation across layers: most layers exhibit attention sinks, while a small subset rarely or never does so in our sample of 256 prompts. In particular, encoder layers 2 and 14 in the ControlNet branch almost never exhibit attention sinks ($\approx 1\%$ and

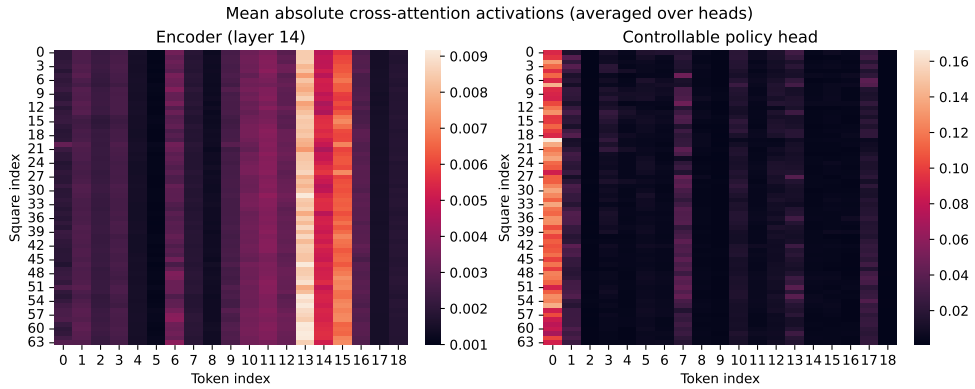

Figure 8.7: Mean absolute cross-attention activations, averaged over attention heads. Left: encoder layer 14. Right: controllable policy head.

0%, respectively), whereas the remaining encoder layers and the controllable policy head do so in nearly all cases (typically $> 90\%$). The controllable policy head (as well as layers 8 and 13) exhibits attention sinks in 100% of the tested prompts.

Cross-attention activations are also highly similar across board squares. That is, for a given text token, the activation from square i is similar to that from square j, as shown in Figure 8.7.

Unfortunately, attempts to mitigate this behaviour did not improve performance. Masking out padding tokens in the cross-attention keys (Section 6.3.8), which removes a potential source of trivial attention targets, resulted in a slight degradation. Similarly, adding gated attention [76] (Section 6.3.1) did not recover this loss. We leave further investigation to future work.

8.3 Failure Modes and Stress Cases

We observe that the model is sensitive to minor variations in prompt phrasing, even when the underlying instruction is semantically equivalent. For example, adding or omitting punctuation can lead to substantial changes in the predicted move distribution. In particular, the prompts "`Open with the English Opening`" and "`Open with the English Opening.`" (with a trailing period) can produce qualitatively different policies.

Similarly, lexical variations that do not alter the intended opening can affect whether the model follows the instruction at all. For example, after the position "`1. e4 c5`", the prompts "`Play the Alapin Sicilian`" and "`Play the Alapin Sicilian Defense`" both assign the highest probability to `c3`, the defining move of the Alapin Sicilian. In contrast, the semantically similar prompt "`Open with the Alapin Sicilian`" assigns the highest probability to `Nf3`, with `c3` becoming only the second-ranked move. Thus, a small change in phrasing can shift the top prediction from an opening-specific move to a more generic developing move.

One contributing factor is the distribution of prompt templates used during training. None of the pretraining templates, and only a subset of the instruction templates ($\approx 33.4\%$, $\frac{684}{2{,}048}$), begin with the verb "Play", while none begin with "Open". In contrast, many instruction templates begin with contextual prefixes such as player names or titles. As a result, prompts that more closely match the training distribution (e.g., those beginning with "Play") tend to elicit more stable behavior, whereas out-of-distribution phrasing (e.g., "Open with") can lead to degraded or inconsistent responses despite semantic equivalence.

From a mechanistic perspective, this sensitivity is consistent with the presence of attention sinks (Section 8.2.2), which are prevalent across most layers of the model. When a small subset of tokens disproportionately captures attention, minor changes in tokenization or phrasing can be amplified through the network, resulting in large downstream changes in the induced policy. While this does not establish a causal relationship, it provides a plausible explanation for the observed instability.

We emphasize that these examples are selected to illustrate failure modes rather than reflect average-case performance. Nevertheless, they highlight a limitation in the robustness of instruction following: small perturbations in prompt form can induce disproportionately large changes in the resulting policy.

Chapter 9

Discussion and Future Work

This chapter summarizes the main findings, limitations, and future directions of this work. UniMaia demonstrates that natural language prompts can steer a pretrained chess policy while remaining competitive with structured-conditioning approaches for human-like and instruction-conditioned behavior, without requiring explicitly structured inputs at inference time. Unlike prior systems that rely on discrete metadata, retrieval-based player embeddings, or language models trained directly over chess text, UniMaia instead emphasizes prompt-based control over a domain-native policy network.

Building on this framework, UniMaia-Aux further demonstrates that auxiliary temporal conditioning can improve behavioral modeling and expected accuracy through the prediction of additional game-state and behavioral targets, including move delay, remaining plies, game result, and termination reason. Together, these results suggest that language-conditioned policy modulation can support both flexible semantic control and competitive human behavioral modeling within a unified framework.

This design enables flexible semantic control, but also introduces several challenges. Prompt wording, template quality, metadata representation, and architectural choices all strongly affect model behavior. In addition, auxiliary temporal conditioning introduces new trade-offs between top-move prediction accuracy, expected accuracy, and dependence on temporally structured information. Addressing these issues is central not only to building more robust and controllable chess agents, but more broadly to enabling reliable language-conditioned control in structured decision-making systems.

9.1 Key Findings

The results show that natural language prompts can effectively steer a pretrained chess policy across multiple behavioral axes, including opening selection and player strength, while remaining competitive with models that rely on structured metadata. UniMaia achieves strong performance across both prompt-conditioned and metadata-conditioned benchmarks, outperforming several prior baselines under the evaluated settings. Furthermore, UniMaia-Aux demonstrates that auxiliary temporal conditioning can improve expected accuracy and behavioral modeling during general gameplay while retaining competitive move-prediction performance.

At the same time, the results reveal several important structural and behavioral properties.

Controllability vs. Behavioral Realism. Prompt conditioning can effectively steer a pretrained chess policy while remaining competitive with metadata-conditioned approaches for predicting human moves. In particular, hybrid models such as UniMaia, which combine a domain-specific policy with natural language prompts, match or exceed the performance of purely metadata-conditioned approaches in many settings. Purely prompt-conditioned models also achieve strong results in certain regimes, particularly in opening-heavy tasks and when modeling highly active players; however, these gains are largely driven by high-frequency patterns in the data, which obscures general performance on human move prediction tasks.

Controllability, however, does not directly translate to behavioral realism. Improvements on prompt-conditioned benchmarks do not consistently transfer to metadata-conditioned benchmarks, revealing a structural tension between prompt adherence and human-like play. These objectives must therefore be balanced rather than jointly optimized. At the same time, auxiliary temporal conditioning improves expected accuracy and behavioral prediction on several benchmarks, suggesting that richer temporal context can improve expected accuracy even when top-move accuracy remains comparatively stable.

Across model classes, distinct performance patterns emerge. Prompt-conditioned models perform well in frequency-dominated regimes, while metadata-conditioned and policy-based models achieve stronger overall move-prediction performance due to their structured inductive biases. As a hybrid model, UniMaia inherits these advantages while introducing flexible semantic control through prompting. At the same time, it captures meaningful skill-dependent behavior across Elo, with policies varying continuously but with noticeable local fluctuations, indicating a context-sensitive representation of player strength.

Data Distribution and Evaluation Effects. Model performance is highly sensitive to both dataset construction and evaluation protocol. Rebalancing strategies produce asymmetric effects: balancing across time controls improves performance, whereas balancing across openings degrades it by distorting naturally skewed distributions. This highlights the importance of preserving informative structure in the data rather than uniformly normalizing it. Evaluation is similarly sensitive to design choices. Including or excluding the first 10 plies significantly alters relative model performance, indicating that opening-phase decisions disproportionately influence benchmark outcomes. Performance metrics therefore reflect not only model quality, but also the specific distributions and regimes emphasized during training and evaluation.

Structure over Scale. Performance gains are driven primarily by architectural and data-centric decisions rather than model scale. Modifications to conditioning structure, auxiliary objectives, and sampling strategy yield substantial improvements, whereas increasing capacity, through larger backbones or higher LoRA ranks, provides only modest gains and exhibits clear diminishing returns. In this templated setting, the limited information content of the input further constrains the utility of additional parameters, with higher-rank adaptations offering minimal benefit. These results indicate that how information is structured and injected into the model is more important than raw model size. Optimization also plays a meaningful role: NorMuon consistently improves training dynamics and downstream performance relative to AdamW and Muon, demonstrating that optimizer choice can materially affect outcomes.

Reliability and Robustness of Language Conditioning. Language-conditioned control remains flexible but brittle. Small changes in prompt phrasing, punctuation, or wording can lead to substantial shifts in the predicted policy, indicating limited robustness to semantically equivalent inputs. In addition, effectiveness depends strongly on domain alignment: text encoders trained on chess-specific data outperform larger general-purpose models, showing that training data composition matters more than scale for extracting useful control signals. While prompt-based conditioning enables expressive and flexible interaction, it also introduces sensitivity that must be addressed to ensure reliable behavior in structured decision-making settings.

9.2 Limitations

9.2.1 Prompt Design and Data Quality

The prompt distribution is based on synthetic templates generated with the assistance of an LLM, followed by a semi-automated conversion into Jinja2 templates and manual review. This approach enables scalable and diverse prompt construction, but it remains template-based and may not fully reflect the variability of naturally occurring user queries.

The two template sets differ systematically in their metadata coverage. LichessTemplates-Pretrain retains all available metadata, whereas LichessTemplates-Instruct selectively includes or omits metadata fields such as player identity, Elo, and opening information. In particular, time control information is entirely absent from LichessTemplates-Instruct. As a result, the model is never exposed to time control as part of the prompt-conditioned distribution, which likely limits its ability to condition on this signal despite its importance in human play.

This omission reflects a practical trade-off in the template construction process. Expanding the template space to include additional metadata dimensions (e.g., time control or clock usage) would significantly increase the number of templates requiring manual validation, making iteration costly. More broadly, each modification to the template generation pipeline requires re-validating the full set of instantiated templates to ensure correctness. While the initial templates were carefully constructed, this iterative process makes it difficult to guarantee the absence of minor artifacts in the final dataset.

As a result, some templates contain small grammatical inconsistencies (e.g., duplicated words). These artifacts are retained for reproducibility, but they highlight the fragility of large-scale template-based data generation.

Finally, the opening instruction-following benchmarks rely on curated opening names and canonical move sequences. This makes evaluation controlled and interpretable, but may also reward memorization or dataset-specific naming conventions rather than robust instruction-following under natural language variation.

9.2.2 Data Representation and Training Objective

The core UniMaia training objective predicts only the next move from the board state and prompt. This objective is simple and directly aligned with policy learning, but it is sample-inefficient and leaves much of the available game information unused. UniMaia-Aux

partially addresses this limitation through auxiliary prediction targets related to game outcome, remaining game length, clock usage, and termination behavior. However, these auxiliary objectives remain comparatively limited and are only introduced during a later fine-tuning stage rather than throughout the full training pipeline.

All conditioning information is expressed through natural language. This is central to the thesis, but it also introduces ambiguity: the same metadata can be phrased in many ways, omitted entirely, or combined with other signals. As a result, the model may underuse metadata that would be easier to exploit in structured form.

The auxiliary temporal conditioning used in UniMaia-Aux introduces additional limitations and trade-offs. Although the temporal prompt suffixes are randomly omitted with probability 0.1 during training, the temporal metadata itself is formatted using a fixed suffix template rather than the broader randomized prompt-template distribution used elsewhere in the dataset. This simplification reduces implementation complexity, but may also encourage overfitting to specific temporal prompt phrasing, limit robustness to natural language variation in temporal descriptions, and contribute to a distribution mismatch between the auxiliary fine-tuning setup and the original benchmark distributions. Although the auxiliary objectives improve expected accuracy and behavioral modeling on several benchmarks, they can also slightly reduce top-move accuracy when temporal metadata is unavailable during evaluation.

The training data is also restricted to Lichess games. This provides scale and rich metadata, but may introduce platform-specific biases in player pool, time controls, opening choices, and game termination patterns. Over-the-board games and games from other online platforms may follow different distributions.

9.2.3 Architectural Trade-offs

The layer-coupled ControlNet architecture enables modular conditioning and preserves the frozen Lc0 policy network, but it is not necessarily the strongest architecture considered. Deep fusion with full fine-tuning achieves stronger empirical performance in later ablations, suggesting that the ControlNet constraints trade off performance for modularity, controllability, and preservation of the base model.

The text encoder may also be larger than necessary. The language input is relatively short and domain-specific, and the ablations in Section 6.3.8 suggest that smaller encoders can remain competitive. More efficient text encoders may therefore provide similar controllability at lower computational cost.

Similarly, the auxiliary temporal conditioning used in UniMaia-Aux introduces additional architectural and optimization complexity while only modestly improving some evaluation settings. Although these auxiliary objectives improve expected accuracy and behavioral modeling on several benchmarks, they can also slightly reduce top-move accuracy when temporal metadata is unavailable. This suggests that the current integration strategy for temporal conditioning may not optimally balance top-move accuracy, expected accuracy, and performance across both opening-specific and general-play settings.

Finally, UniMaia does not include retrieval or player-specific adaptation. Prior work such as Maia4All [92] shows that game-specific embeddings can improve personalized move prediction. Incorporating similar mechanisms could improve human imitation while retaining prompt-level control.

9.2.4 Evaluation and Analysis

The evaluation is limited to automated benchmarks. The model is not evaluated through human studies, live play, or deployment against human players and bots. As a result, the relationship between benchmark performance and perceived playing style remains uncertain.

Several behavioral properties also remain underexplored. For example, this work does not systematically analyze blunder rate as a function of prompted Elo, consistency of style across positions, or robustness to paraphrases and punctuation changes. These analyses would help clarify whether the model has learned stable behavioral control or benchmark-specific correlations.

9.2.5 Mechanistic Interpretability

This work includes several mechanistic interpretability analyses, including layer-wise contribution measurements, cross-attention sink analysis, and comparisons between UniMaia and Lc0 representations. These analyses provide initial evidence of where conditioning signals enter the model. In particular, the controllable policy head is most active in early-opening positions, the layer-14 residuals are larger than those of most other layers, and layers 2 and 14 exhibit substantially fewer attention-sink patterns than the surrounding layers. These observations suggest that conditioning is not uniformly distributed across the network.

However, these analyses do not yet provide a complete causal account of the model's behavior. In particular, while the representation similarity analysis shows that UniMaia remains moderately aligned with the Lc0 representations, it does not explain which dimensions or features differ between the two models. Similarly, the attention and residual

analyses identify where conditioning signals appear to concentrate, but do not fully explain how these signals produce instruction-following, temporal conditioning, opening control, or human move prediction.

9.3 Future Work

9.3.1 Prompt Modeling

Future work should improve the realism, diversity, and robustness of the prompt distribution. While the current pipeline already uses LLMs to generate and paraphrase prompts and converts them into reusable templates, it remains limited by its reliance on a fixed template set and manual validation.

One key direction is to incorporate natural user prompts and systematically introduce variations in punctuation, naming conventions, and phrasing to better reflect real-world usage. This is particularly important for temporally conditioned prompts, where the current auxiliary temporal suffixes use a comparatively fixed phrasing scheme. Expanding prompts with richer descriptions of chess style, strategy, and intent, such as tactical versus positional play, solid versus risky play, or aggressive win-seeking behavior, may further improve controllability and stylistic diversity.

Another important direction is to improve template validation and filtering. Currently, correctness is ensured primarily through manual review, which becomes increasingly difficult as the template space grows. Automated validation methods, such as heuristic checks or LLM-based judges, could be used to detect malformed or inconsistent prompts at scale. This would improve data quality while preserving the flexibility and diversity of the existing generation pipeline.

9.3.2 Prompt and Structured Conditioning

A useful direction is to study the trade-off between natural language conditioning and structured metadata. The goal need not be to replace prompts with structured inputs; rather, future systems could combine both. Natural language can provide flexible high-level control, while structured metadata can provide consistency for signals such as Elo, time control, clock usage, color, and player identity.

The results of UniMaia-Aux further suggest that temporal and behavioral metadata can improve expected accuracy and human behavioral modeling when incorporated appropriately. Future work should therefore explore hybrid conditioning strategies that combine structured temporal signals with flexible natural language control.

9.3.3 Architectural Improvements

The strong performance of deep fusion with full fine-tuning suggests that this architecture warrants further investigation. Future work should systematically compare ControlNet-style conditioning, deep fusion, early fusion, and hybrid approaches under matched training budgets.

Smaller text encoders should also be explored, as the prompt distribution may not require a large language model. A particularly promising and low-cost direction is to fine-tune a smaller, embedding-focused text model on chess data. The ablations in Section 6.3.8 suggest that domain alignment is more important than model scale, and that embedding-oriented models already perform competitively. Fine-tuning a compact embedding model on a chess-specific corpus (e.g., an open subset of the data used to train ChessGPT-base) could yield comparable conditioning quality at significantly lower computational cost. This would enable more efficient training and deployment while retaining effective prompt-based control.

Other conditioning methods, including classifier-free guidance and distilled guidance variants, may provide more direct control over the policy distribution and potentially improve the trade-off between top-move and expected accuracy observed in UniMaia-Aux.

Retrieval-augmented components are another promising direction. Incorporating player-specific or game-specific embeddings could improve personalization and human move prediction while preserving prompt-based semantic control.

9.3.4 Training Objectives

Future work should further extend the training objective beyond next-move prediction. Although UniMaia-Aux introduces auxiliary targets related to game outcome, remaining moves, move delay, and termination behavior, these objectives are only incorporated during a later fine-tuning stage and remain comparatively limited. Additional auxiliary tasks, such as move quality estimation, uncertainty prediction, or richer temporal modeling, could further improve representation learning and sample efficiency.

Bidirectional, multi-task, or self-supervised objectives may also help the model better connect prompts, metadata, temporal information, and chess behavior. More generally, future work should investigate how auxiliary supervision affects the trade-off between top-move accuracy, expected accuracy, and behavioral realism.

9.3.5 Expanded Data Sources

Training and evaluation could be expanded beyond Lichess. Incorporating over-the-board games, other online platforms, and curated engine-analysis datasets would make it possible to study distributional differences and improve robustness across chess contexts.

9.3.6 Evaluation and Deployment

Future evaluation should include human judgments from players, coaches, or titled players, as well as live-play evaluation against humans and existing bots. Deployment in an online environment would provide useful feedback about perceived style, behavioral realism, prompt robustness, and real-world controllability.

9.3.7 Mechanistic Interpretability

Future interpretability work should move from descriptive analysis to causal intervention. Activation patching, residual ablations, prompt counterfactuals, and targeted interventions on the controllable policy head and layer-14 ControlNet residuals could help determine which components mediate opening control, Elo conditioning, and human move prediction. Extending these analyses to UniMaia-Aux may also help clarify how temporal conditioning and auxiliary objectives alter the internal policy representations. Further analysis of the layer-13 representations could additionally clarify which features are preserved from Lc0 and which are modified by language conditioning.

9.4 Conclusion

This thesis demonstrates that natural language prompts can be used to steer a pre-trained chess policy toward controllable, human-like behavior. UniMaia combines the inductive biases of a strong domain-specific policy network with the flexibility of prompt-conditioned modulation, achieving competitive performance across both prompt-conditioned

and metadata-conditioned benchmarks. Furthermore, UniMaia-Aux demonstrates that incorporating auxiliary temporal conditioning and auxiliary behavioral prediction objectives can improve expected accuracy and behavioral modeling while retaining strong top-move prediction performance.

The results show that prompt-conditioned policy models occupy a middle ground between purely prompt-conditioned approaches and conventional policy networks. UniMaia improves controllability relative to policy-only models while maintaining strong performance on human move prediction tasks. At the same time, while improvements in prompt-conditioned performance generally correlate with improvements in human move prediction, current models do not fully match the expected accuracy of the strongest metadata-conditioned approaches. This gap does not appear to be fundamental, and likely reflects limitations in current model design, training, and prompt construction.

Beyond empirical performance, the findings emphasize the importance of architectural design, conditioning strategy, and data curation. Conditioning structure, auxiliary objectives, and sampling strategy account for most of the observed gains, while increasing model capacity yields diminishing returns. In particular, carefully preserving and rebalancing the data distribution is critical for achieving strong results. The results further suggest that domain alignment and conditioning structure are more important than raw parameter scale for controllable policy modulation.

These results point to a broader paradigm for structured decision-making systems: retaining domain-specific inductive biases while introducing controllability through lightweight prompt-conditioned mechanisms offers a flexible alternative to end-to-end approaches. At the same time, this paradigm introduces practical challenges, including sensitivity to prompt formulation, limitations in how structured information is conveyed through prompts, trade-offs between modularity and predictive accuracy, and difficulties balancing top-move accuracy with expected accuracy.

Future work on more robust prompt modeling, improved conditioning architectures, richer auxiliary objectives, and domain-aligned text encoders will be essential for advancing prompt-conditioned control. More generally, this work highlights both the promise and the current limitations of using natural language as a flexible interface for controlling structured decision-making systems.

References

[1] LM Title (Lichess Master Title) — lichess.org. https://lichess.org/forum/general-chess-discussion/lm-title--lichess-master-title. [Accessed 29-03-2026].

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[3] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.

[4] Noah Amsel, David Persson, Christopher Musco, and Robert M Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. *arXiv preprint arXiv:2505.16932*, 2025.

[5] Anthropic. Cowork: Claude Code power for knowledge work — Claude by Anthropic — claude.com. https://claude.com/product/cowork. [Accessed 24-04-2026].

[6] Max Bachmann. GitHub - rapidfuzz/RapidFuzz: Rapid fuzzy string matching in Python using various string metrics — github.com. https://github.com/rapidfuzz/RapidFuzz. [Accessed 22-06-2024].

[7] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song,

Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025.

[8] Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, et al. Gpt-neox-20b: An open-source autoregressive language model. In *Proceedings of BigScience Episode# 5–Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, 2022.

[9] Adrian Bolton, Alexander Lerchner, Alexandra Cordell, Alexandre Moufarek, Andrew Bolt, Andrew Lampinen, Anna Mitenkova, Arne Olav Hallingstad, Bojan Vujatovic, Bonnie Li, et al. Sima 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

[10] Daniel Bächli and Thibault Duplessis. Implement the fifty-move rule correctly · Issue #8681 · lichess-org/lila — github.com. https://github.com/lichess-org/lila/issues/8681#issuecomment-821948591, 2021. [Accessed 10-07-2024].

[11] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[12] Nicholas Carlini. Playing chess with large language models — nicholas.carlini.com. https://nicholas.carlini.com/writing/2023/chess-llm.html, 2023. [Accessed 10-07-2024].

[13] Junsong Chen, Yue Wu, Simian Luo, Enze Xie, Sayak Paul, Ping Luo, Hang Zhao, and Zhenguo Li. Pixart-{\delta}: Fast and controllable image generation with latent consistency models. *arXiv preprint arXiv:2401.05252*, 2024.

[14] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

[15] Hyung Won Chung, Noah Constant, Xavier Garcia, Adam Roberts, Yi Tay, Sharan Narang, and Orhan Firat. Unimax: Fairer and more effective language sampling for large-scale multilingual pretraining. *arXiv preprint arXiv:2304.09151*, 2023.

[16] Hailey Collet. Exploring mamba's capabilities with chess: Ablations, probing inner states, and transformer comparisons. https://haileystorm.github.io/mamba-chess/exploring-capabilities/, 2024. [Accessed 10-07-2024].

[17] Marc-Alexandre Côté, Akos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, et al. Textworld: A learning environment for text-based games. In *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*, pages 41–75. Springer, 2019.

[18] @danegraphics. Humaia: An attempt to make the most human lichess bot. https://www.reddit.com/r/ComputerChess/comments/u7jros/humaia_an_attempt_to_make_the_most_human_lichess/, 2022. [Accessed 10-07-2024].

[19] Timothy Dozat. Incorporating nesterov momentum into adam. *ICLR Workshop Track*, 2016.

[20] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[21] Thibault Duplessis. auto threefold draw setting + improve prefs page · lichess-org/lila@de1ed4f — github.com. https://github.com/lichess-org/lila/commit/de1ed4f0e42bea222c9bc424129b2cad2fbec584, 4 2014. [Accessed 07-04-2026].

[22] Thibault Duplessis. change default threefold setting to always - closes #11812 · lichess-org/lila@c113413 — github.com. https://github.com/lichess-org/lila/commit/c1134134d398be97701672b5d916d5ed91bfbd43, 11 2022. [Accessed 07-04-2026].

[23] Steven J Edwards. Portable Game Notation Specification and Implementation Guide. https://ia802908.us.archive.org/26/items/pgn-standard-1994-03-12/PGN_standard_1994-03-12.txt, 3 1994. [Accessed 02-03-2026].

[24] Arpad E Elo. The uscf rating system - a scientific achievement. *Chess Life*, 16(6):160–161, 1961.

[25] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *Forty-first International Conference on Machine Learning*, 2024.

[26] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[27] Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. Chessgpt: Bridging policy learning and language modeling. *Advances in Neural Information Processing Systems*, 36, 2024.

[28] Niklas Fiekas. GitHub - niklasf/python-chess: A chess library for Python, with move generation and validation, PGN parsing and writing, Polyglot opening book reading, Gaviota tablebase probing, Syzygy tablebase probing, and UCI/XBoard engine communication — github.com. https://github.com/niklasf/python-chess. [Accessed 18-06-2024].

[29] Niklas Fiekas. lower minRating: 600 -¿ 400 by niklasf · Pull Request #12488 · lichess-org/lila — github.com. https://github.com/lichess-org/lila/pull/12488/changes. [Accessed 03-02-2026].

[30] Henrik Forstén. Moves left head by Ttl · Pull Request #961 · LeelaChessZero/lc0 — github.com. https://github.com/LeelaChessZero/lc0/pull/961#issuecomment-587112109, 2020. [Accessed 03-03-2026].

[31] David Forsyth. The shorthand method of noting chess positions. *Glasgow Weekly Herald*, 2 1883.

[32] International Chess Foundation. Appendix C. Algebraic Notation – FIDE Rules Commission — rcc.fide.com. https://rcc.fide.com/appendixc/. [Accessed 03-03-2026].

[33] International Chess Foundation. Fide laws of chess. https://www.fide.com/FIDE/handbook/LawsOfChess.pdf, 2008. [Accessed 10-07-2024].

[34] Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[35] Mark E Glickman. The glicko system. *Boston University*, 16(8):9, 1995.

[36] Mark E Glickman. Example of the glicko-2 system. *Boston University*, 28:2012, 2012.

[37] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. Lora+: Efficient low rank adaptation of large models. *arXiv preprint arXiv:2402.12354*, 2024.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[39] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290, 2024.

[40] Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. Model context protocol (mcp): Landscape, security threats, and future research directions. *ACM Transactions on Software Engineering and Methodology*, 2025.

[41] Feng-Hsiung Hsu. *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press, 2002.

[42] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

[43] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[44] Rudolf Huber and Stefan-Meyer Kahlen. Description of the universal chess interface (uci). https://backscattering.de/chess/uci/, 4 2006. [Accessed 03-03-2026].

[45] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[46] Athul Paul Jacob, David J Wu, Gabriele Farina, Adam Lerer, Hengyuan Hu, Anton Bakhtin, Jacob Andreas, and Noam Brown. Modeling strong and human-like gameplay with kl-regularized search. In *International Conference on Machine Learning*, pages 9695–9728. PMLR, 2022.

[47] Michael Jansen. Free chess game database for Scid and PGN — lumbrasgigabase.com. https://lumbrasgigabase.com/en/. [Accessed 14-04-2026].

[48] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.

[49] Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora. *arXiv preprint arXiv:2312.03732*, 2023.

[50] Adam Karvonen. Emergent world models and latent variable estimation in chess-playing language models. *arXiv preprint arXiv:2403.15498*, 2024.

[51] Donald E Knuth and Ronald W Moore. An analysis of alpha-beta pruning. *Artificial intelligence*, 6(4):293–326, 1975.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[53] Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, Vivek Ramanujan, William Howard-Snyder, Kaifeng Chen, Sham Kakade, Prateek Jain, et al. Matryoshka representation learning. *Advances in Neural Information Processing Systems*, 35:30233–30249, 2022.

[54] Hugo Laurençon, Lucile Saulnier, Léo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander M. Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. Obelics: An open web-scale filtered dataset of interleaved image-text documents, 2023.

[55] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[56] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Prompt repetition improves non-reasoning llms. *arXiv preprint arXiv:2512.14982*, 2025.

[57] Zhen Li, Mingdeng Cao, Xintao Wang, Zhongang Qi, Ming-Ming Cheng, and Ying Shan. Photomaker: Customizing realistic human photos via stacked id embedding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8640–8650, 2024.

[58] Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. Normuon: Making muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025.

[59] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

[60] Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. *arXiv preprint arXiv:2502.16982*, 2025.

[61] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

[62] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[63] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. *arXiv preprint arXiv:2406.01967*, 2024.

[64] Loïc Magne, Anas Awadalla, Guanzhi Wang, Yinzhen Xu, Joshua Belofsky, Fengyuan Hu, Joohwan Kim, Ludwig Schmidt, Georgia Gkioxari, Jan Kautz, Yisong Yue, Yejin Choi, Yuke Zhu, and Linxi "Jim" Fan. Nitrogen: An open foundation model for generalist gaming agents, 2026.

[65] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1677–1687, 2020.

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[67] Daniel Monroe and Philip A Chalmers. Mastering chess with a transformer model. *arXiv preprint arXiv:2409.12272*, 2024.

[68] Daniel Monroe, George Eilender, Philip Chalmers, Zhenwei Tang, and Ashton Anderson. Chessformer: A unified architecture for chess modeling. In *The Fourteenth International Conference on Learning Representations*, 2026.

[69] Yuj Nasu. Efficiently updatable neural-network-based evaluation functions for computer shogi. *The 28th World Computer Shogi Championship Appeal Document*, 185, 2018.

[70] OpenAI. GPT-3.5 Turbo Model — OpenAI API — developers.openai.com. https://developers.openai.com/api/docs/models/gpt-3.5-turbo. [Accessed 22-04-2026].

[71] OpenAI. Introducing ChatGPT. https://openai.com/index/chatgpt/, 2022. [Accessed 10-07-2024].

[72] OpenAI. Introducing Codex — openai.com. https://openai.com/index/introducing-codex/, 5 2025. [Accessed 24-04-2026].

[73] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[74] Pablo Pernias, Dominic Rampas, Mats L Richter, Christopher J Pal, and Marc Aubreville. Würstchen: An efficient architecture for large-scale text-to-image diffusion models. *arXiv preprint arXiv:2306.00637*, 2023.

[75] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

[76] Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, et al. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

[77] Maria Abi Raad, Arun Ahuja, Catarina Barros, Frederic Besse, Andrew Bolt, Adrian Bolton, Bethanie Brownfield, Gavin Buttimore, Max Cant, Sarah Chakera, et al. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

[78] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[79] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[80] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.

[81] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, and Tim Genewein. Grandmaster-level chess without search. *arXiv preprint arXiv:2402.04494*, 2024.

[82] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

[83] SeatGeek. GitHub - seatgeek/thefuzz: Fuzzy String Matching in Python — github.com. https://github.com/seatgeek/thefuzz. [Accessed 22-06-2024].

[84] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

[85] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[86] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[87] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[88] Shivalika Singh, Freddie Vargus, Daniel D'souza, Börje F Karlsson, Abinaya Mahendiran, Wei-Yin Ko, Herumb Shandilya, Jay Patel, Deividas Mataciunas, Laura O'Mahony, et al. Aya dataset: An open-access collection for multilingual instruction tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11521–11567, 2024.

[89] James C Spall. An overview of the simultaneous perturbation method for efficient optimization. *Johns Hopkins apl technical digest*, 19(4):482–492, 1998.

[90] Weihao Tan, Xiangyang Li, Yunhao Fang, Heyuan Yao, Shi Yan, Hao Luo, Tenglong Ao, Huihui Li, Hongbin Ren, Bairen Yi, Yujia Qin, Bo An, Libin Liu, and Guang Shi. Lumine: An open recipe for building generalist agents in 3d open worlds, 2025.

[91] Zhenwei Tang, Difan Jiao, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. Maia-2: A unified model for human-ai alignment in chess. *Advances in Neural Information Processing Systems*, 37:20919–20944, 2024.

[92] Zhenwei Tang, Difan Jiao, Eric Xue, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. Learning to imitate with less: Efficient individual behavior modeling in chess. *arXiv preprint arXiv:2507.21488*, 2025.

[93] Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Jason Wei, Xuezhi Wang, Hyung Won Chung, Siamak Shakeri, Dara Bahri, Tal Schuster, et al. Ul2: Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*, 2022.

[94] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan

Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025.

[95] Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[96] The Leela Chess Zero team. Technical Explanation of Leela Chess Zero — github.com. https://github.com/LeelaChessZero/lc0/wiki/Technical-Explanation-of-Leela-Chess-Zero, 2021. [Accessed 06-07-2024].

[97] The Lichess team. Chess players and leaderboards — lichess.org. https://lichess.org/player. [Accessed 03-02-2026].

[98] The Lichess team. Chess rating systems — lichess.org. https://lichess.org/page/rating-systems. [Accessed 03-02-2026].

[99] The Lichess team. GitHub - lichess-org/chess-openings: An aggregated data set of chess opening names — github.com. https://github.com/lichess-org/chess-openings. [Accessed 22-06-2024].

[100] The Lichess team. lc0/src/neural/encoder.cc at master – LeelaChessZero/lc0 — github.com. https://github.com/LeelaChessZero/lc0/blob/master/src/neural/encoder.cc. [Accessed 03-03-2026].

[101] The Lichess team. lczero-training/tf/lc0_az_policy_map.py at master – LeelaChessZero/lczero-training — github.com. https://github.com/LeelaChessZero/lczero-training/blob/master/tf/lc0_az_policy_map.py. [Accessed 03-03-2026].

[102] The Lichess team. lichess.org open database — database.lichess.org. https://database.lichess.org/. [Accessed 18-06-2024].

[103] The Lichess team. Weekly Rapid rating distribution — lichess.org. https://lichess.org/stat/rating/distribution/rapid. [Accessed 03-02-2026].

[104] The Lichess team. Frequently Asked Questions — lichess.org. https://lichess.org/faq#threefold, 2024. [Accessed 10-07-2024].

[105] The Stockfish team. GitHub - official-stockfish/Stockfish: A free and strong UCI chess engine — github.com. https://github.com/official-stockfish/Stockfish. [Accessed 05-03-2026].

[106] The Stockfish team. Stockfish 12 — stockfishchess.org. https://stockfishchess.org/blog/2020/stockfish-12/, 2020. [Accessed 06-07-2024].

[107] Viet-Anh Tran. GitHub - CallOn84/LeelaNets: All the Lc0 & Lc0-derived nets trained by CallOn84 — github.com. https://github.com/CallOn84/LeelaNets, 2022. [Accessed 10-07-2024].

[108] Unknown. Historical peak elo and identity data for notable lichess users, 5 2026.

[109] Jack Urbanek, Angela Fan, Siddharth Karamcheti, Saachi Jain, Samuel Humeau, Emily Dinan, Tim Rocktäschel, Douwe Kiela, Arthur Szlam, and Jason Weston. Learning to speak and act in a fantasy text adventure game. *arXiv preprint arXiv:1903.03094*, 2019.

[110] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[111] Henrique Schechter Vera, Sahil Dua, Biao Zhang, Daniel Salz, Ryan Mullins, Sindhu Raghuram Panyam, Sara Smoot, Iftekhar Naim, Joe Zou, Feiyang Chen, et al. Embeddinggemma: Powerful and lightweight text representations. *arXiv preprint arXiv:2509.20354*, 2025.

[112] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[113] Maurice Weber, Daniel Y Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, et al. Redpajama: an open dataset for training large language models. *Advances in neural information processing systems*, 37:116462–116492, 2024.

[114] Kaiyue Wen, Zhiyuan Li, Jason Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape perspective. *arXiv preprint arXiv:2410.05192*, 2024.

[115] Brandon T Willard and Rémi Louf. Efficient guided generation for large language models. *arXiv preprint arXiv:2307.09702*, 2023.

[116] Yi Wu, Ziqiang Li, Heliang Zheng, Chaoyue Wang, and Bin Li. Infinite-id: Identity-preserved personalization via id-semantics decoupling paradigm. *arXiv preprint arXiv:2403.11781*, 2024.

[117] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[118] Linmiao Xu. Stockfish NNUE training data — robotmoon.com. https://robotmoon.com/nnue-training-data/. [Accessed 05-03-2026].

[119] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[120] Hu Ye, Jun Zhang, Sibo Liu, Xiao Han, and Wei Yang. Ip-adapter: Text compatible image prompt adapter for text-to-image diffusion models. *arXiv preprint arXiv:2308.06721*, 2023.

[121] Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. *arXiv preprint arXiv:2404.05719*, 2024.

[122] Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

[123] Biao Zhang, Paul Suganthan, Gaël Liu, Ilya Philippov, Sahil Dua, Ben Hora, Kat Black, Gus Martins, Omar Sanseviero, Shreya Pathak, et al. T5gemma 2: Seeing, reading, and understanding longer. *arXiv preprint arXiv:2512.14856*, 2025.

[124] Edwin Zhang, Vincent Zhu, Naomi Saphra, Anat Kleiman, Benjamin L Edelman, Milind Tambe, Sham M Kakade, and Eran Malach. Transcendence: Generative models can outperform the experts that train them. *arXiv preprint arXiv:2406.11741*, 2024.

[125] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3836–3847, 2023.

[126] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025.

[127] Yiming Zhang, Athul Paul Jacob, Vivian Lai, Daniel Fried, and Daphne Ippolito. Human-aligned chess with a bit of search. *arXiv preprint arXiv:2410.03893*, 2024.

APPENDICES

Appendix A

Termination Statistics

This appendix reports the frequencies of termination reasons for representative dataset shards used in the analysis of repetition-related outcomes discussed in Section 6.1.4. We include shards from March 2014, October 2022, and January 2023 to demonstrate that five-fold repetition remains consistently rare across time, including both early data and data collected after changes to Lichess draw-claim behavior. Specifically, Lichess introduced automatic draw claims for threefold repetition under low time conditions in April 2014 and later made automatic claiming the default in November 2022 [21, 22].

Table A.1: Termination reason frequencies for the only shard of March 2014 games in LichessGames, after applying the filtering criteria described in Section 6.1.4.

Termination Reason	Counts	Percentage (%)
Resigned	266,449	37.425
Time forfeit	231,410	32.503
Checkmate	193,718	27.209
Threefold repetition	5,615	0.789
Insufficient material	5,098	0.716
Stalemate	4,868	0.684
Draw by agreement	3,966	0.557
Fivefold repetition	640	0.090
Fifty moves	196	0.028
Total	711,960	100.000

Table A.2: Termination reason frequencies for the first shard of October 2022 games in LichessGames, after applying the filtering criteria described in Section 6.1.4.

Termination Reason	Counts	Percentage (%)
Resigned	1,326,763	34.911
Time forfeit	1,288,942	33.915
Checkmate	1,061,309	27.926
Threefold repetition	50,813	1.337
Stalemate	31,683	0.834
Insufficient material	26,885	0.707
Draw by agreement	9,675	0.255
Fivefold repetition	3,576	0.094
Fifty moves	735	0.019
Abandoned	50	0.001
Rules infraction	27	0.001
Total	3,800,458	100.000

Table A.3: Termination reason frequencies for the first shard of January 2023 games in LichessGames, after applying the filtering criteria described in Section 6.1.4.

Termination Reason	Counts	Percentage (%)
Resigned	1,321,464	34.705
Time forfeit	1,294,684	34.002
Checkmate	1,064,219	27.949
Threefold repetition	59,097	1.552
Stalemate	30,763	0.808
Insufficient material	26,698	0.701
Draw by agreement	9,385	0.246
Fivefold repetition	731	0.019
Fifty moves	626	0.016
Rules infraction	34	0.001
Abandoned	2	0.000
Total	3,807,703	100.000

Appendix B

Policy Continuity Evaluation Details

This appendix provides the evaluation prompts and additional quantitative diagnostics used in the policy continuity analysis presented in Section 8.1.

B.1 Policy Continuity Evaluation Prompts

We provide the full PGN positions and prompt templates used in the two policy continuity evaluation settings: across positions and under opening conditioning.

B.1.1 Policy Continuity Across Positions

We evaluate policy continuity across positions using representative configurations spanning the early opening, late opening, middlegame, and endgame.

Table B.1: PGNs and prompt templates used for policy continuity evaluation across representative positions (early opening, late opening, middlegame, and endgame).

Stage	**PGN**	**Prompt Template**
Early Opening	1. e4 c5	`You are an anonymous white player rated {elo}, playing against an anonymous black player rated {elo}, using the Sicilian Alapin`
Late Opening	1. d4 d5 2. c4 e6 3. Nc3 c5 4. cxd5 exd5 5. Nf3 Nf6 6. g3 Nc6 7. Bg2	`You are an anonymous black player rated {elo}, playing against an anonymous white player rated {elo}, opening with the main line of the Tarrasch Defense's Prague Variation in the QGD.`
Middlegame	1. e4 c5 2. Nf3 Nc6 3. Bc4 g6 4. c3 Bg7 5. Qb3 e6 6. d4 cxd4 7. cxd4 Nxd4 8. Nxd4 Bxd4 9. Qd3 Qb6 10. Be3 Bxe3 11. fxe3 Ne7 12. Nc3	`Username prashu_20091, a beginner with an ELO of {elo}, confidently plays as White, utilizing the Sicilian Defense: Old Sicilian with the opening moves 1. e4 c5 2. Nf3 Nc6.`

Continued on next page

Table B.1: PGNs and prompt templates used for policy continuity evaluation across representative positions (early opening, late opening, middlegame, and endgame). (Continued)

Stage	PGN	Prompt Template
Endgame	1. e4 d5 2. exd5 Qxd5 3. Nc3 Qa5 4. d4 c6 5. Nf3 Bg4 6. Be2 Bxf3 7. Bxf3 Nf6 8. O-O e6 9. Bf4 Bb4 10. Ne4 Nxe4 11. Bxe4 O-O 12. c3 Be7 13. b4 Qd8 14. Qh5 g6 15. Qh3 Nd7 16. Bh6 Re8 17. a3 Nf6 18. Bf3 Bf8 19. Bxf8 Rxf8 20. a4 Rc8 21. Rad1 Nd7 22. c4 Nb6 23. d5 cxd5 24. cxd5 Nxd5 25. Bxd5 exd5 26. Qb3 Qc7 27. Rxd5 Rfd8 28. Rxd8+ Rxd8 29. h3 Rd2 30. Qe3 Qd8 31. Qxa7 Rd1 32. Rxd1 Qxd1+ 33. Kh2 Qd6+ 34. g3 Qf6 35. Qe3 Qd6 36. Qc5 Qd2 37. a5 Kg7 38. Kg2 Kf6 39. Qb6+ Kg7 40. Qxb7 h5 41. a6	`The game in question was a rated blitz chess match hosted on the online chess platform Lichess.org on January 1, 2023. The match featured a rapid time control of five minutes with no additional time per move, denoted as "300+0." Playing with the white pieces was a player using the username "Talca" who held a rating of {elo}. The black pieces were commanded by player "pierobaldo" who held an identical rating of {elo} at the time of the game. The opening played in this game was the Scandinavian Defense, specifically the Main Line variation, which is introduced with the moves 1. e4 d5 2. exd5 Qxd5 3. Nc3 Qa5. The game featured an average of {elo} Elo with a 0-point difference`

B.1.2 Policy Continuity Under Opening Conditioning

We evaluate policy continuity from the initial position under different opening-conditioned prompts.

1. **King's Pawn Opening**: `You are an anonymous white player rated {elo}, playing against an anonymous black player rated {elo}, using the King's Pawn Opening`
2. **Queen's Pawn Opening**: `You are an anonymous white player rated {elo}, playing against an anonymous black player rated {elo}, using the Queen's Pawn Opening`
3. **English Opening**: `You are an anonymous white player rated {elo}, playing against an anonymous black player rated {elo}, using the English Opening`
4. **Zukertort Opening**: `You are an anonymous white player rated {elo}, playing against an anonymous black player rated {elo}, using the Zukertort Opening`

B.2 Quantitative Diagnostics of Policy Continuity

These plots provide quantitative measures of policy smoothness across Elo values, complementing the qualitative trends observed in the main text.

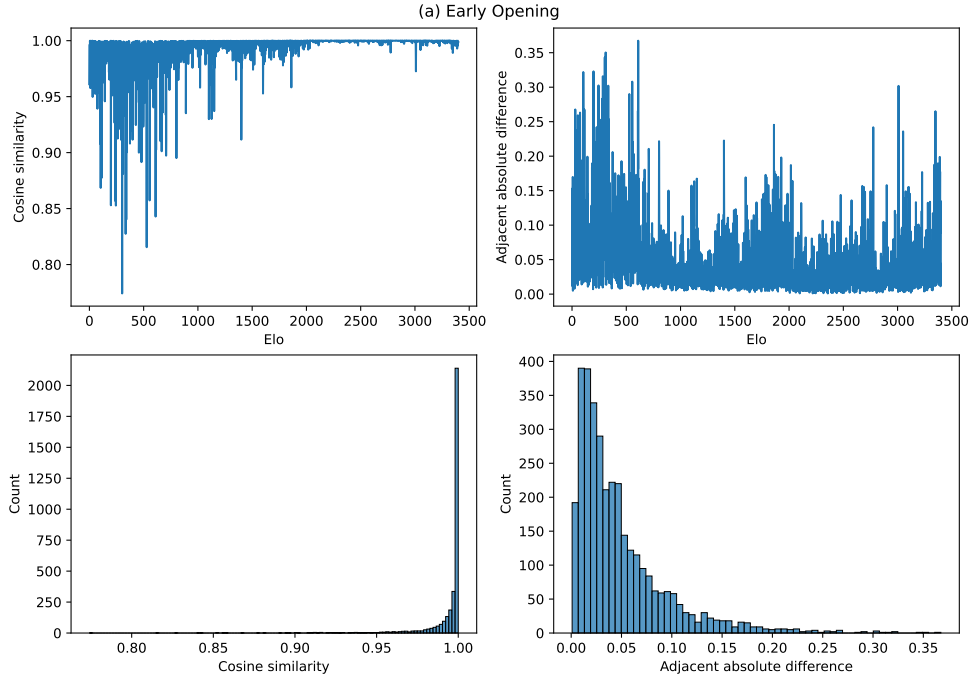

Figure B.1: Quantitative policy continuity diagnostics across Elo values. Top-left: cosine similarity between policies at adjacent Elo values. Top-right: L2 norm of the difference between adjacent policies. Bottom-left: distribution of adjacent cosine similarities. Bottom-right: distribution of adjacent policy differences.

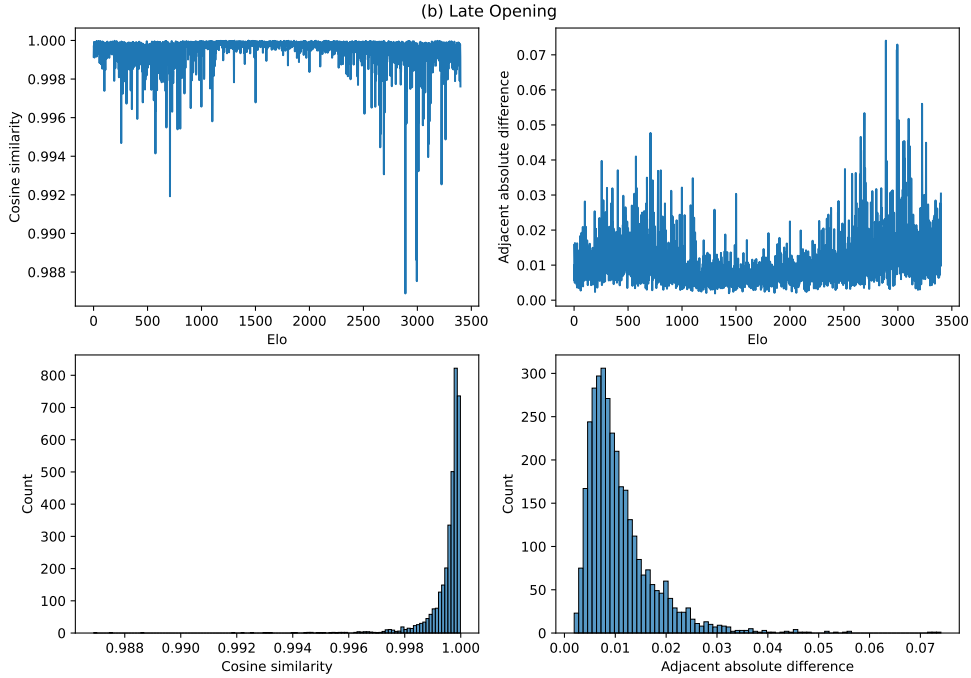

Figure B.2: Quantitative policy continuity diagnostics across Elo values. Top-left: cosine similarity between policies at adjacent Elo values. Top-right: L2 norm of the difference between adjacent policies. Bottom-left: distribution of adjacent cosine similarities. Bottom-right: distribution of adjacent policy differences.

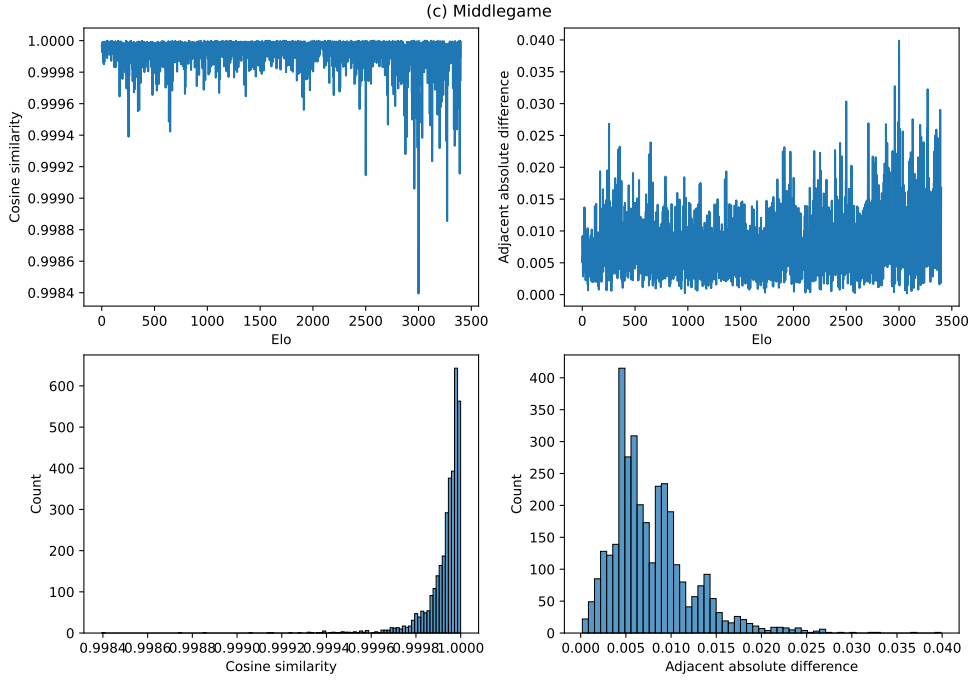

Figure B.3: Quantitative policy continuity diagnostics across Elo values. Top-left: cosine similarity between policies at adjacent Elo values. Top-right: L2 norm of the difference between adjacent policies. Bottom-left: distribution of adjacent cosine similarities. Bottom-right: distribution of adjacent policy differences.

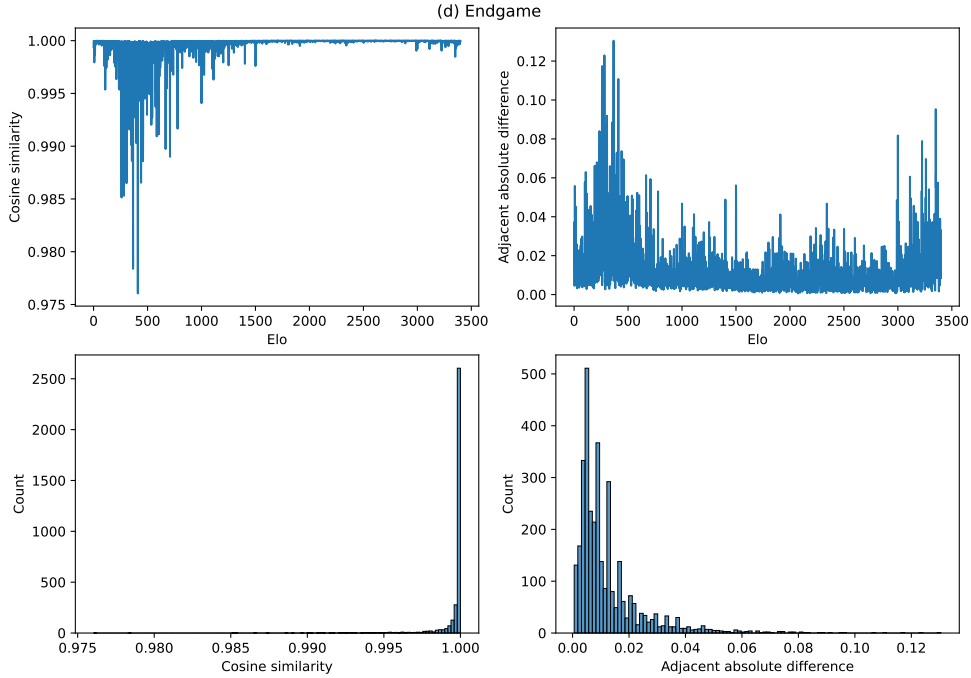

Figure B.4: Quantitative policy continuity diagnostics across Elo values. Top-left: cosine similarity between policies at adjacent Elo values. Top-right: L2 norm of the difference between adjacent policies. Bottom-left: distribution of adjacent cosine similarities. Bottom-right: distribution of adjacent policy differences.

Appendix C

Prompt Templates

C.1 Example LichessTemplates Prompt Templates

The following examples illustrate the linguistic variability and metadata coverage used during training and evaluation.

From LichessTemplates-Pretrain:

- `This {time_control_lower} chess game was played online at Lichess.org on {date}, with each player given {td_all_seconds} and {text_time_increment} per move. The game featured {white_title_full_quot_name} {white_alias_alias_paren} as {W_lit}, with an Elo rating of {white_elo}, against {black_title_full_quot_name} {black_alias_alias_paren} as {B_lit}, who had a rating of {black_elo}. The opening used was the {opening_commas}, delineated by the moves: {opening_moves}.`
- `In a {ratedness_lower} {time_control_lower} chess game hosted on Lichess.org on {date}, {white_title_full_paren_title} {white_alias_known_as_quot_name}, rated {white_elo}, played as {W_lit} against {black_title_full_paren_title} {black_alias_known_as_quot_name} and rated {black_elo}. The game opened with the {opening}, starting with moves {opening_moves}. The time control for the match was {td_all_seconds} {with_text_time_increment} per move.`

- `On {date}, at Lichess.org, {white_title_full} {white_alias} (white) with an Elo rating of {white_elo} played against {black_title_full} {black_alias} (black) who had an Elo rating of {black_elo}. This rated {time_control_lower} game had a time control of {text_td} per player {time_increment_words_without_any_i The game featured the {opening}, which began with the moves {opening_moves}.`

From LichessTemplates-Instruct:

- `Join {black_title} {black_alias}, ELO {black_elo}, as he plays {B_lit} utilizing the {opening} with the opening moves {opening_moves}. Challenge this {black_rank_p to a game on Lichess and test your skills against his opening strategy.`
- `Play as {white_title_full} {white_alias}, taking the {w_lit_lower_pieces} and opening the game with {opening_moves}, known as the {opening_variant_name_of_th`
- `In this chess game on Lichess, the player, controlling the {b_lit_lower_pieces}, opens with the {opening} by playing {opening_moves}.`

C.2 Metadata-Conditioned Benchmark Prompts

All prompt-conditioned models were provided with the following prompt template (formatted with game metadata) when evaluated on the metadata-conditioned benchmarks:

"`An anonymous white player with an ELO of {white_elo} plays chess against another anonymous black player, with an ELO of {black_elo} in a {type.lower()} game on Lichess played with {time_info}.`"

Here, "`time_control`" is rendered as "`a time control of <duration as human-friendly text> with (no increment|an increment of <increment as human-friendly text>)`"

C.3 GM PGN Header Prompt

The GM PGN header prompt used for evaluation is based on the open-sourced implementations accompanying Carlini [12] and Allie [127]. This version differs slightly from the prompt described in the Carlini blog and reproduced in the Allie appendix.

In particular, the blog version includes a fixed game result and lists Kasparov as White and Carlsen as Black, whereas the implementation version omits the result field and lists Carlsen as White and Kasparov as Black. We use the implementation version for all experiments.

```
[White "Magnus Carlsen"]
[Black "Garry Kasparov"]
[WhiteElo "2900"]
[BlackElo "2800"]
```

C.4 Elo and Time Control PGN Header

Aside from the opening-specific benchmarks, ChessGPT-base and GPT-3.5 Turbo Instruct were prompted using a PGN header prepended to the movelist:

```
"""[White "???"]
[Black "???"]
[WhiteElo "\{white\_elo\}"]
[BlackElo "\{black\_elo\}"]
[TimeControl "\{time\_control\}"]

"""
```

This format mirrors the structured metadata representation used during training of these models, in contrast to the natural-language prompts used for UniMaia.

C.5 Auxiliary Targets Prompt Suffix

When predicting the auxiliary targets, we append the following suffix to the prompt if the time control is available:

`The time control for this {time_control.lower()} chess game was {text_td} {time_increment_with_no_additional_increment}.`

where `text_td` is the time delay in hours, minutes, and seconds, and `time_increment_with_no_additio` is either `with no additional increment` or `with {time_increment} <second or seconds> of increment`, with the correct pluralization.

If the game has `%clk` annotations for move delay prediction, we append the following:

`Time taken last turn by active player: {active_player_last_move_time}. Time taken last turn by opponent: {opponent_last_move_time}. Time remaining for the active player: {active_player_time_remaining}. Time remaining for the opponent: {opponent_time_remaining}.`

Appendix D

Detailed Evaluation Results

This appendix provides the full set of evaluation results referenced in Section 7.3. These include detailed comparisons across prompt-conditioned and metadata-conditioned benchmarks, as well as additional analyses for language-based models and performance across Elo ranges.

Unless otherwise specified, all results are reported using the unified evaluation harness described in Section 7.3.

D.1 Top-Move Accuracy

We report top-move accuracy (Acc@1) results for metadata-conditioned benchmarks without excluding early-game moves. These results complement those in the main text, which omit the first 10 plies to reduce the influence of opening memorization.

D.2 Expected Accuracy

This section reports expected accuracy ($\mathbb{E}$[Acc]) across both prompt-conditioned and metadata-conditioned benchmarks. Expected accuracy reflects the probability of selecting the correct move when sampling from the predicted distribution, and provides a complementary view to top-move accuracy.

Table D.1: **Metadata-conditioned** results, measured by **Acc@1**. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models. †Evaluated with the Elo and time control PGN header. ‡Evaluated with the general policy prompt.

Model	ABB	M1-S	M1	M2R
Lc0-CF (t3)	0.4091	0.4234	0.4249	0.4040
Lc0-CF (t82)	0.4036	0.4195	0.4199	0.3981
Lc0-CF (BT3)	0.4073	0.4231	0.4240	0.4032
Lc0-CF (BT4)	0.4053	0.4181	0.4210	0.3991
Lc0-CF (BT4-spsa)	0.4032	0.4116	0.4177	0.3972
Maia-1*	0.5043	0.5072	0.5119	0.5094
Maia-1**	0.5057	0.5122	<u>0.5158</u>	0.5096
Maia-2 (Blitz)	0.4961	0.4998	0.5048	0.4964
Maia-2 (Rapid)	0.4956	0.5038	0.5065	0.5031
Allie-Policy	**0.5568**	**0.5654**	**0.5650**	**0.5497**
ChessCLIP	0.0799	0.0803	0.0797	0.0821
ChessGPT-base†	0.4408	0.4452	–	0.4380
ChessGPT-chat‡	0.1671	0.1661	–	0.1631
GPT-3.5†	0.5398	<u>0.5554</u>	–	0.5370
UniMaia	<u>0.5447</u>	0.5534	–	<u>0.5413</u>
UniMaia-Aux	0.5361	0.5495	–	0.5356

D.2.1 Prompt-Conditioned

Tables D.2 and D.3 report expected accuracy on prompt-conditioned benchmarks, with and without the first 10 plies of each game.

Including opening moves highlights performance in prompt-rich, frequency-dominated settings, while excluding them isolates performance in more general gameplay scenarios.

D.2.2 Metadata-Conditioned

Tables D.4 and D.5 report expected accuracy on metadata-conditioned benchmarks. As in the main text, we report results both with and without the first 10 plies to assess the influence of opening memorization.

Table D.2: **Prompt-conditioned** results, measured by $\mathbb{E}$[**Acc**]. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	LOB-P	LOB-C	LIF-D	LIF	LIF-T10	LGB
Lc0-CF (t3)	0.3628	0.3627	0.2310	0.2742	0.2298	0.3390
Lc0-CF (t82)	0.3633	0.3630	0.2311	0.2744	0.2293	0.3398
Lc0-CF (BT3)	0.3644	0.3642	0.2322	0.2756	0.2308	0.3405
Lc0-CF (BT4)	0.3657	0.3655	0.2328	0.2772	0.2311	0.3436
Lc0-CF (BT4-spsa)	0.3815	0.3813	0.2462	0.2948	0.2448	0.3616
Maia-1*	0.4130	0.4138	0.3321	0.3653	0.3195	0.3651
Maia-1**	0.4299	0.4307	0.3321	0.3678	0.3204	0.3827
Maia-2 (Blitz)	0.2990	0.2971	0.3227	0.3670	0.3216	0.3423
Maia-2 (Rapid)	0.3056	0.3040	0.3224	0.3650	0.3149	0.3401
Allie-Policy	0.4905	0.4906	0.3510	0.4104	0.3525	0.4480
ChessCLIP	0.0402	0.0403	0.0658	0.0616	0.0496	0.0405
UniMaia	<u>0.5899</u>	**0.7140**	<u>0.3522</u>	<u>0.4297</u>	**0.3678**	<u>0.4834</u>
UniMaia-Aux	**0.5973**	<u>0.6823</u>	**0.3606**	**0.4331**	<u>0.3600</u>	**0.4891**

D.2.3 Inference Strategy Ablations

This section provides detailed results for different prompting and inference strategies used with large language models, including ChessGPT and GPT-3.5 Turbo Instruct. These experiments illustrate the sensitivity of prompt-conditioned models to input formatting and prompt design.

D.3 Performance by Elo Range

We first report dataset size heatmaps for the LIF and ABB benchmarks, followed by accuracy and size heatmaps for all benchmarks.

These visualizations provide additional insight into how performance varies across player skill levels and dataset distributions.

Next, we include the accuracy by Elo and size heatmaps for the remaining benchmarks.

Table D.3: **Prompt-conditioned** results, measured by $\mathbb{E}$[**Acc**]. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	LIF-D	LIF	LIF-T10	LGB
Lc0-CF (t3)	0.2360	0.2763	0.2336	0.3624
Lc0-CF (t82)	0.2364	0.2766	0.2336	0.3651
Lc0-CF (BT3)	0.2372	0.2781	0.2349	0.3657
Lc0-CF (BT4)	0.2379	0.2800	0.2352	0.3712
Lc0-CF (BT4-spsa)	0.2530	0.2994	0.2508	0.3937
Maia-1*	0.3438	0.3713	0.3248	0.3434
Maia-1**	0.3438	0.3733	0.3254	0.3570
Maia-2 (Blitz)	0.3463	0.3913	0.3446	0.3842
Maia-2 (Rapid)	0.3463	0.3886	0.3375	0.3792
Allie-Policy	**0.3652**	<u>0.4215</u>	**0.3607**	<u>0.4393</u>
ChessCLIP	0.0705	0.0662	0.0518	0.0422
UniMaia	0.3444	0.4129	<u>0.3507</u>	0.4252
UniMaia-Aux	<u>0.3592</u>	**0.4218**	0.3507	**0.4521**

Table D.4: **Metadata-conditioned** results, measured by $\mathbb{E}$[**Acc**]. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	ABB	M1-S	M1	M2R
Lc0-CF (t3)	0.2985	0.3093	0.3104	0.2955
Lc0-CF (t82)	0.2977	0.3088	0.3098	0.2948
Lc0-CF (BT3)	0.2997	0.3118	0.3120	0.2970
Lc0-CF (BT4)	0.3009	0.3123	0.3133	0.2979
Lc0-CF (BT4-spsa)	0.3176	0.3295	0.3306	0.3140
Maia-1*	0.3816	0.3840	0.3862	0.3827
Maia-1**	0.3840	0.3877	0.3897	0.3829
Maia-2 (Blitz)	0.3860	0.3921	<u>0.3938</u>	0.3861
Maia-2 (Rapid)	0.3819	0.3890	0.3907	0.3858
Allie-Policy	**0.4310**	**0.4388**	**0.4386**	<u>0.4225</u>
ChessCLIP	0.0652	0.0627	0.0640	0.0685
UniMaia	0.4084	0.4181	–	0.4025
UniMaia-Aux	<u>0.4222</u>	<u>0.4330</u>	–	**0.4234**

Table D.5: **Metadata-conditioned** results, measured by $\mathbb{E}$[**Acc**]. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined. *Evaluated only with the official models. **Evaluated with both the official and unofficial models.

Model	ABB	M1-S	M1	M2R
Lc0-CF (t3)	0.3035	0.3145	0.3165	0.3009
Lc0-CF (t82)	0.3028	0.3137	0.3159	0.3002
Lc0-CF (BT3)	0.3049	0.3170	0.3183	0.3027
Lc0-CF (BT4)	0.3063	0.3175	0.3198	0.3036
Lc0-CF (BT4-spsa)	0.3244	0.3360	0.3386	0.3212
Maia-1*	0.3868	0.3883	0.3902	0.3898
Maia-1**	0.3888	0.3916	0.3932	0.3900
Maia-2 (Blitz)	0.4102	0.4144	<u>0.4166</u>	0.4137
Maia-2 (Rapid)	0.4056	0.4107	0.4131	0.4130
Allie-Policy	**0.4409**	<u>0.4476</u>	**0.4475**	<u>0.4354</u>
ChessCLIP	0.0700	0.0670	0.0686	0.0742
UniMaia	0.4224	0.4317	–	0.4182
UniMaia-Aux	<u>0.4382</u>	**0.4488**	–	**0.4427**

Table D.6: ChessGPT **prompt-conditioned** results, measured by **Acc@1**. Best values are in bold and second-best values are underlined.

Model	LOB-P	LOB-C	LIF-D	LIF	LIF-T10	LGB
ChessGPT-base (prompt)	**0.7998**	**0.8955**	<u>0.3801</u>	**0.4503**	<u>0.4060</u>	**0.5436**
ChessGPT-base (Elo + time control PGN)	<u>0.6100</u>	<u>0.6186</u>	**0.3808**	<u>0.4464</u>	**0.4153**	<u>0.4788</u>
ChessGPT-chat (simple, duplicate)	0.3417	0.2272	0.0981	0.1097	0.0964	0.1532
ChessGPT-chat (simple, original)	0.2601	0.1730	0.0892	0.0992	0.0831	0.1243
ChessGPT-chat (general policy)	0.1630	0.1503	0.1494	0.1637	0.1480	0.1501

Table D.7: ChessGPT **prompt-conditioned** results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined.

Model	LIF-D	LIF	LIF-T10	LGB
ChessGPT-base (prompt)	<u>0.3672</u>	<u>0.4303</u>	<u>0.3888</u>	<u>0.4336</u>
ChessGPT-base (Elo + time control PGN)	**0.3813**	**0.4440**	**0.4158**	**0.4339**
ChessGPT-chat (simple, duplicate)	0.0933	0.1020	0.0874	0.1129
ChessGPT-chat (simple, original)	0.0873	0.0939	0.0727	0.0918
ChessGPT-chat (general policy)	0.1490	0.1557	0.1491	0.1564

Table D.8: ChessGPT **metadata-conditioned** results, measured by **Acc@1**. Best values are in bold and second-best values are underlined.

Model	ABB	M1-S	M2R
ChessGPT-base (prompt)	<u>0.4330</u>	<u>0.4330</u>	<u>0.4310</u>
ChessGPT-base (Elo + time control PGN)	**0.4408**	**0.4452**	**0.4380**
ChessGPT-chat (simple, duplicate)	0.1128	0.1006	0.1146
ChessGPT-chat (simple, original)	0.0933	0.0889	0.0998
ChessGPT-chat (general policy)	0.1671	0.1661	0.1631

Table D.9: ChessGPT **metadata-conditioned** results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined.

Model	ABB	M1-S	M2R
ChessGPT-base (prompt)	<u>0.4260</u>	<u>0.4257</u>	<u>0.4248</u>
ChessGPT-base (Elo + time control PGN)	**0.4336**	**0.4375**	**0.4305**
ChessGPT-chat (simple, duplicate)	0.0980	0.0898	0.1071
ChessGPT-chat (simple, original)	0.0874	0.0821	0.0932
ChessGPT-chat (general policy)	0.1697	0.1719	0.1647

Table D.10: GPT-3.5 **prompt-conditioned results**, measured by **Acc@1**. Best values are in bold and second-best values are underlined.

Model	LOB-P	LOB-C	LIF-D	LIF	LIF-T10	LGB
GPT-3.5 (GM PGN header)	**0.5748**	**0.6431**	0.4344	0.5218	0.4684	**0.5525**
GPT-3.5 (GM PGN header + Prompt)	0.5118	0.5186	0.4341	0.5166	0.4607	0.5138
GPT-3.5 (Prompt + Original PGN header)	0.5196	0.5183	0.4338	0.5166	0.4593	0.5147
GPT-3.5 (Elo + Time Control PGN header)	–	–	**0.4457**	**0.5312**	**0.4778**	0.5514

Table D.11: GPT-3.5 **prompt-conditioned** results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined.

Model	LIF-D	LIF	LIF-T10	LGB
GPT-3.5 (GM PGN header)	0.4531	0.5352	0.4841	0.5562
GPT-3.5 (GM PGN header + Prompt)	0.4584	0.5411	0.4849	0.5555
GPT-3.5 (Prompt + Original PGN header)	0.4579	0.5410	0.4849	0.5543
GPT-3.5 (Elo + Time Control PGN header)	**0.4608**	**0.5444**	**0.4914**	**0.5569**

Table D.12: GPT-3.5 **metadata-conditioned** results, measured by **Acc@1**. Best values are in bold and second-best values are underlined.

Model	ABB	M1-S	M2R
GPT-3.5 (GM PGN header)	0.5298	0.5451	0.5248
GPT-3.5 (GM PGN header + Prompt)	0.5271	0.5409	0.5212
GPT-3.5 (Elo + Time Control PGN header)	**0.5398**	**0.5554**	**0.5370**

Table D.13: GPT-3.5 **metadata-conditioned** results, measured by **Acc@1**. We **omit** the first 10 plies from each game in the benchmark. Best values are in bold and second-best values are underlined.

Model	ABB	M1-S	M2R
GPT-3.5 (GM PGN header)	0.5410	0.5565	0.5390
GPT-3.5 (GM PGN header + Prompt)	0.5493	0.5640	0.5461
GPT-3.5 (Elo + Time Control PGN header)	**0.5498**	**0.5667**	**0.5489**

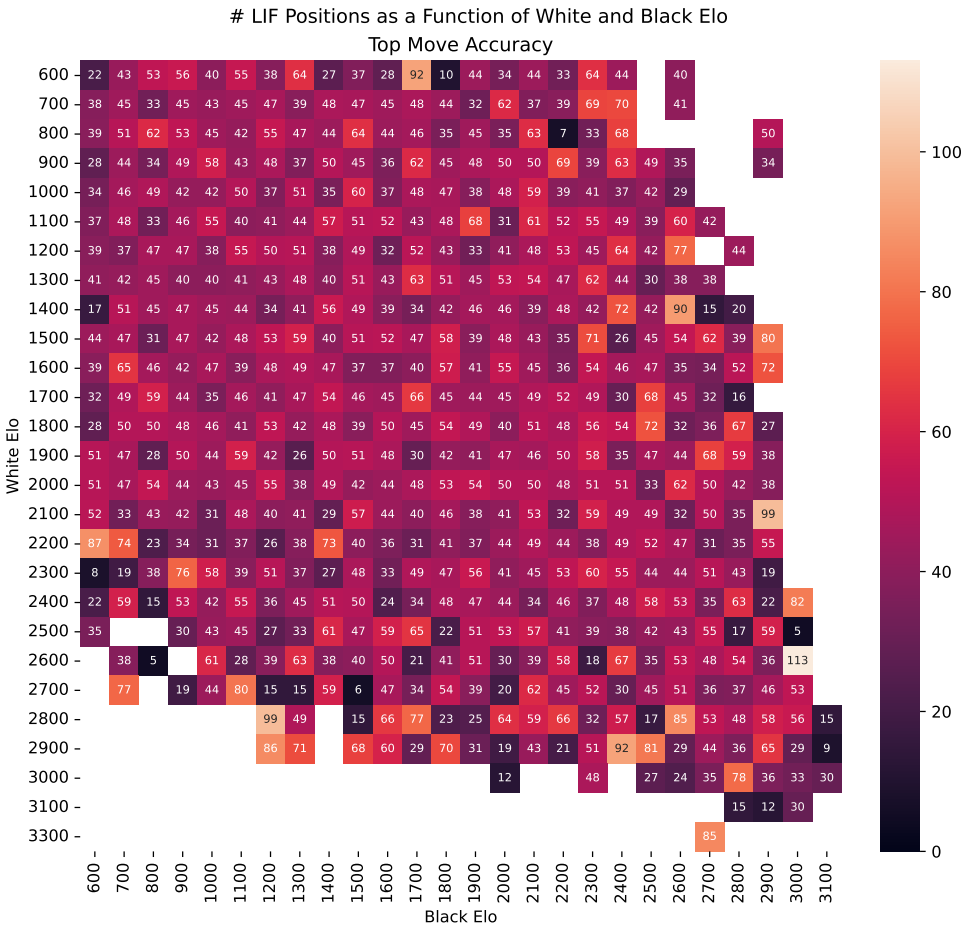

Figure D.1: Number of LIF entries in each rounded White and Black Elo bucket.

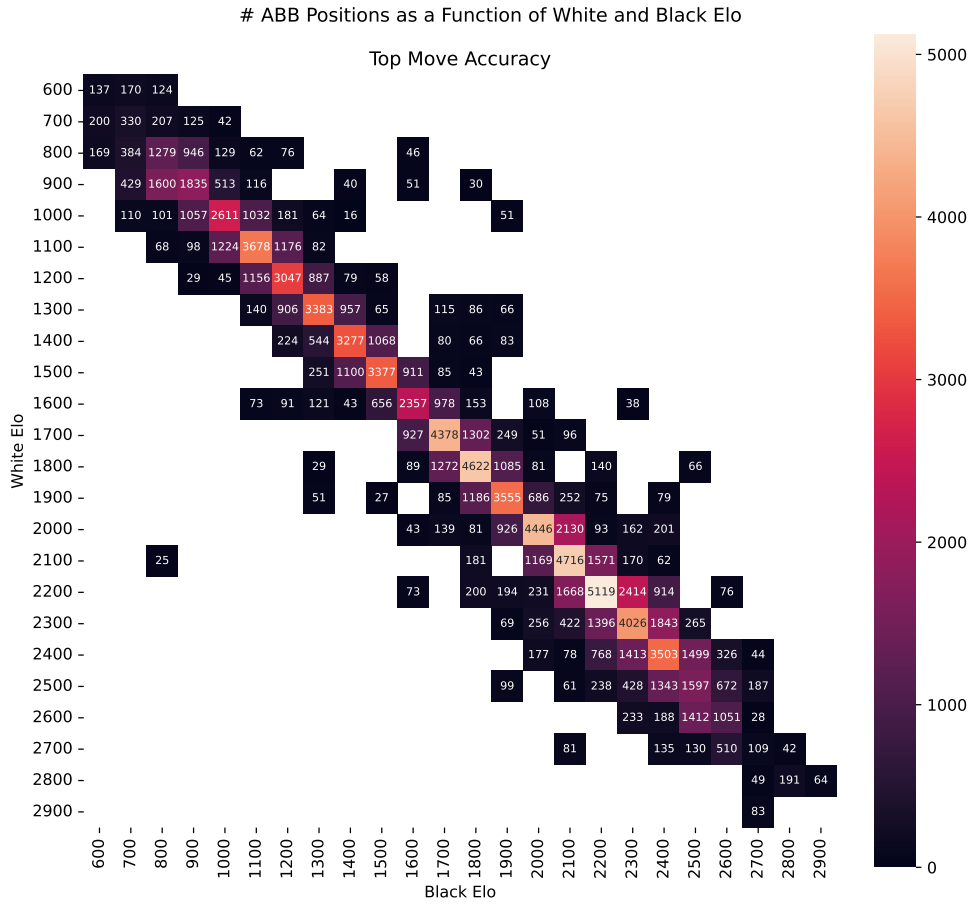

Figure D.2: Number of ABB entries in each rounded White and Black Elo bucket.

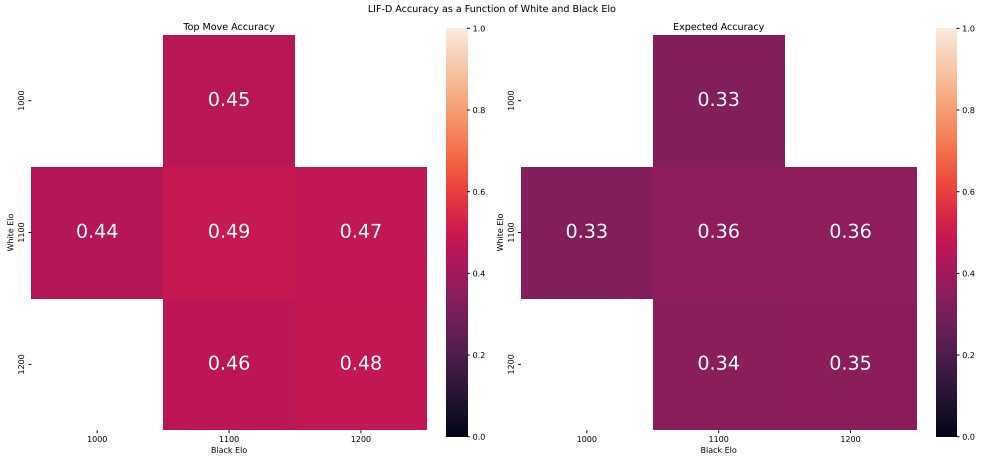

Figure D.3: LIF-D accuracy as a function of White and Black Elo. Left: top-move accuracy. Right: expected accuracy.

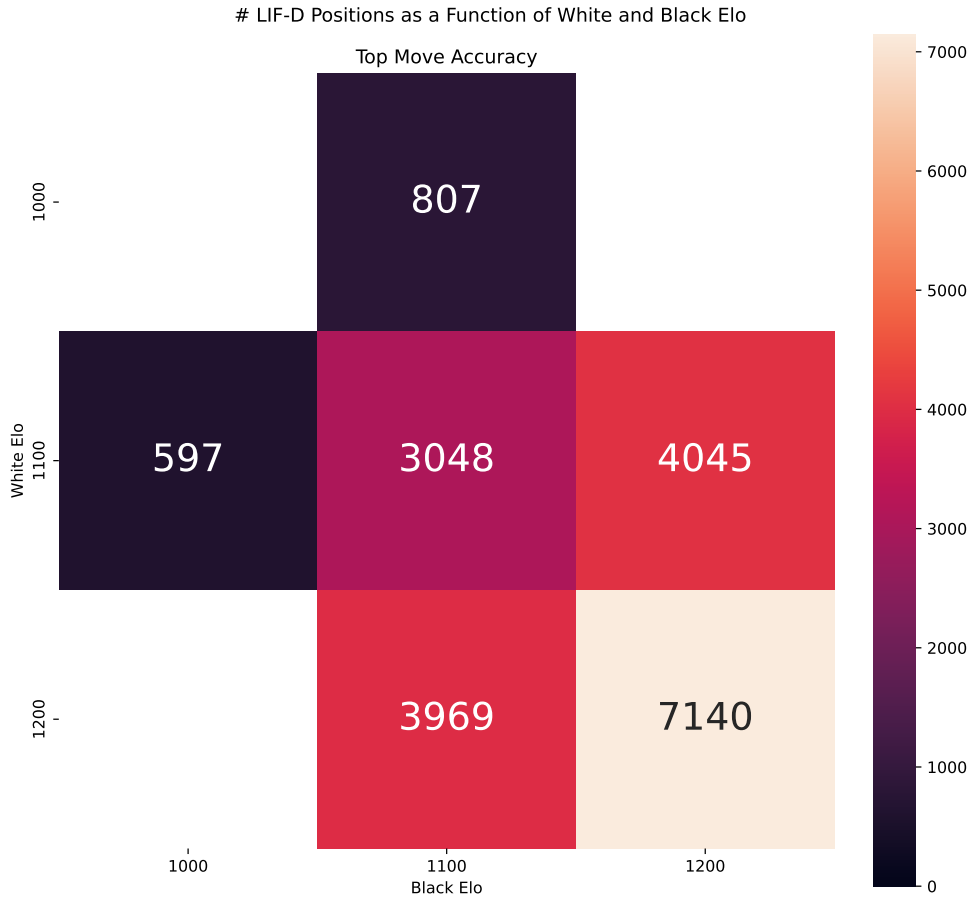

Figure D.4: Number of LIF-D entries in each rounded White and Black Elo bucket.

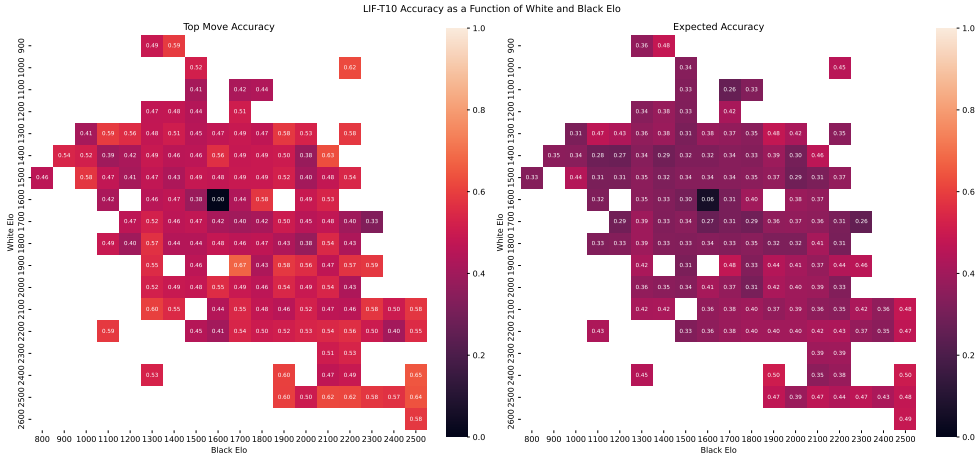

Figure D.5: LIF-T10 accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy

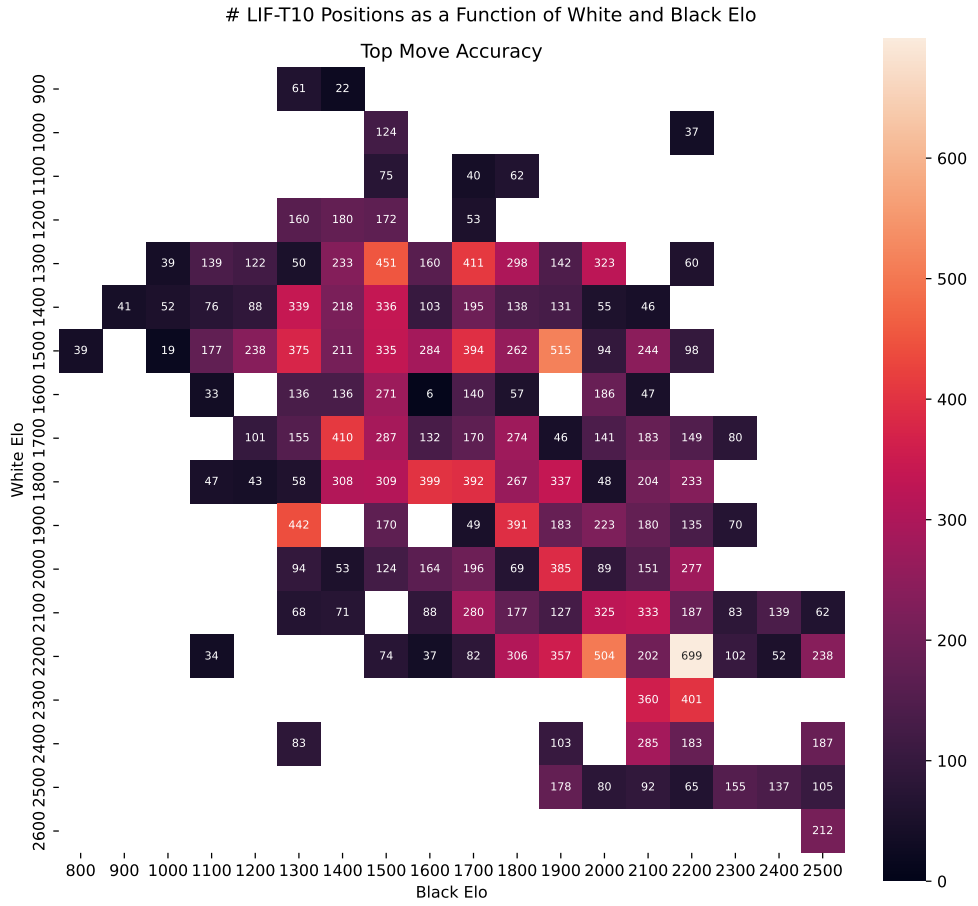

Figure D.6: Number of LIF-T10 entries in each rounded White and Black Elo bucket.

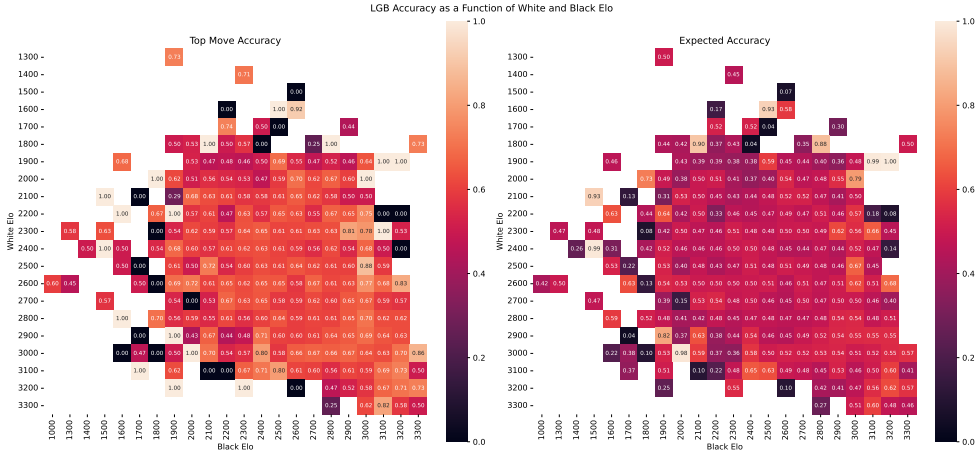

Figure D.7: LGB accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy

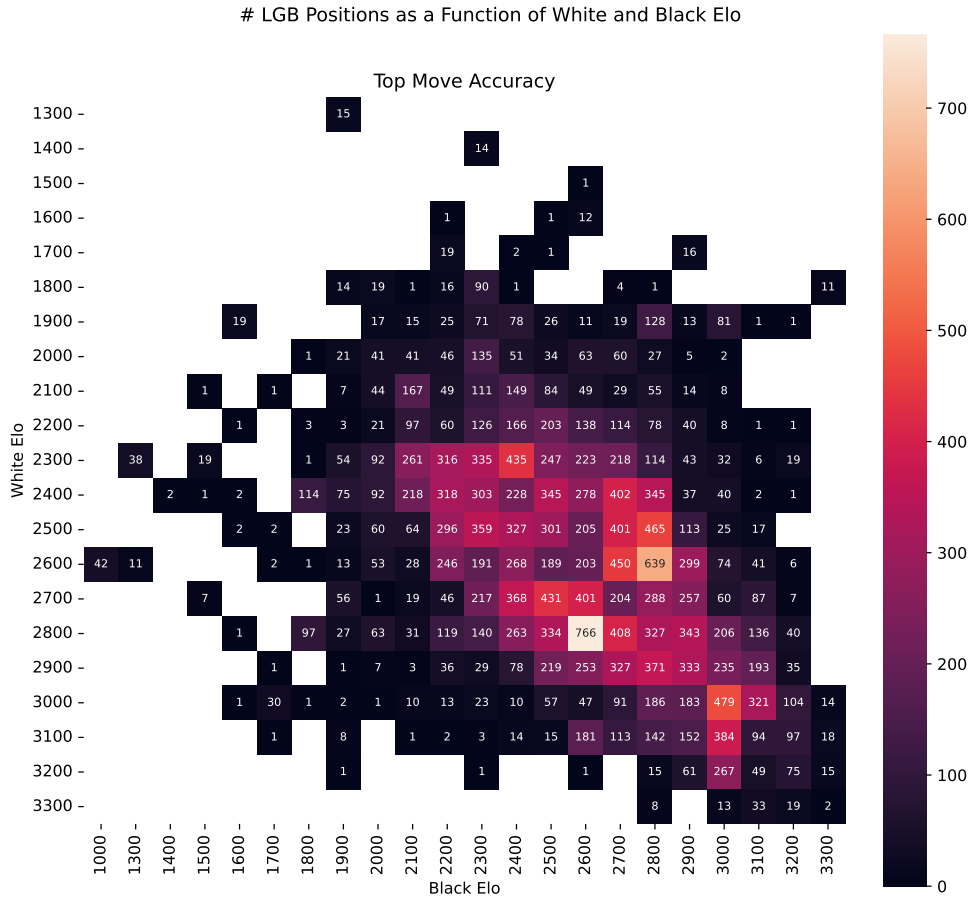

Figure D.8: Number of LGB entries in each rounded White and Black Elo bucket.

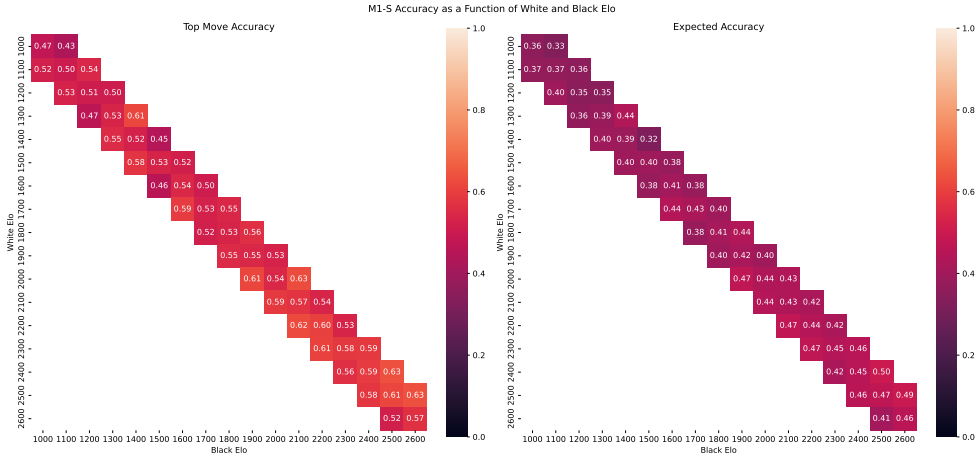

Figure D.9: M1-S accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy

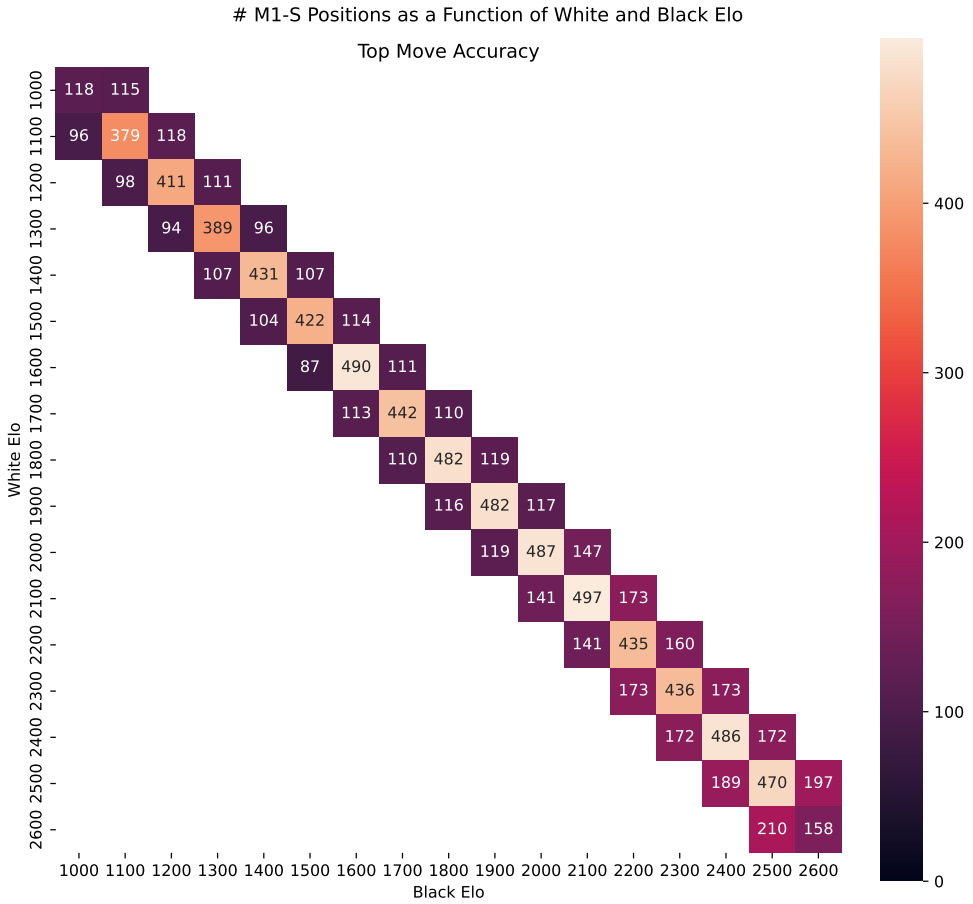

Figure D.10: Number of M1-S entries in each rounded White and Black Elo bucket.

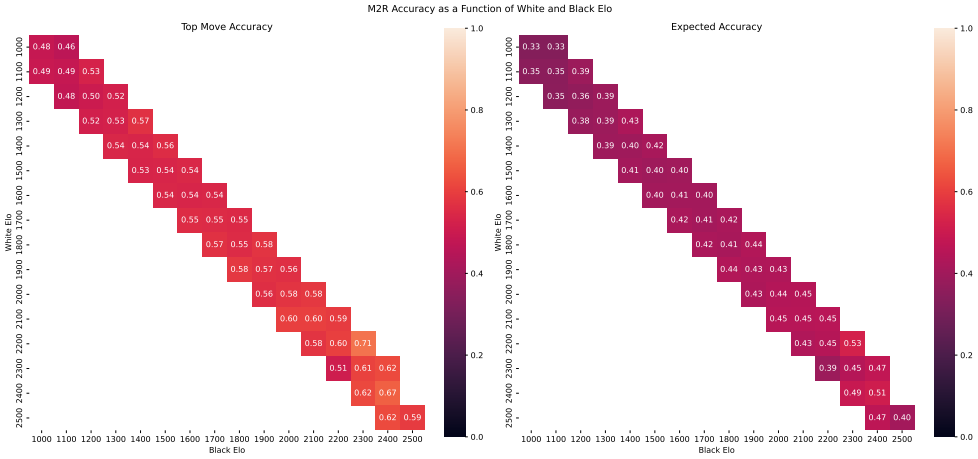

Figure D.11: M2R accuracy as a function of White and Black Elo. Left: top move accuracy. Right: expected accuracy

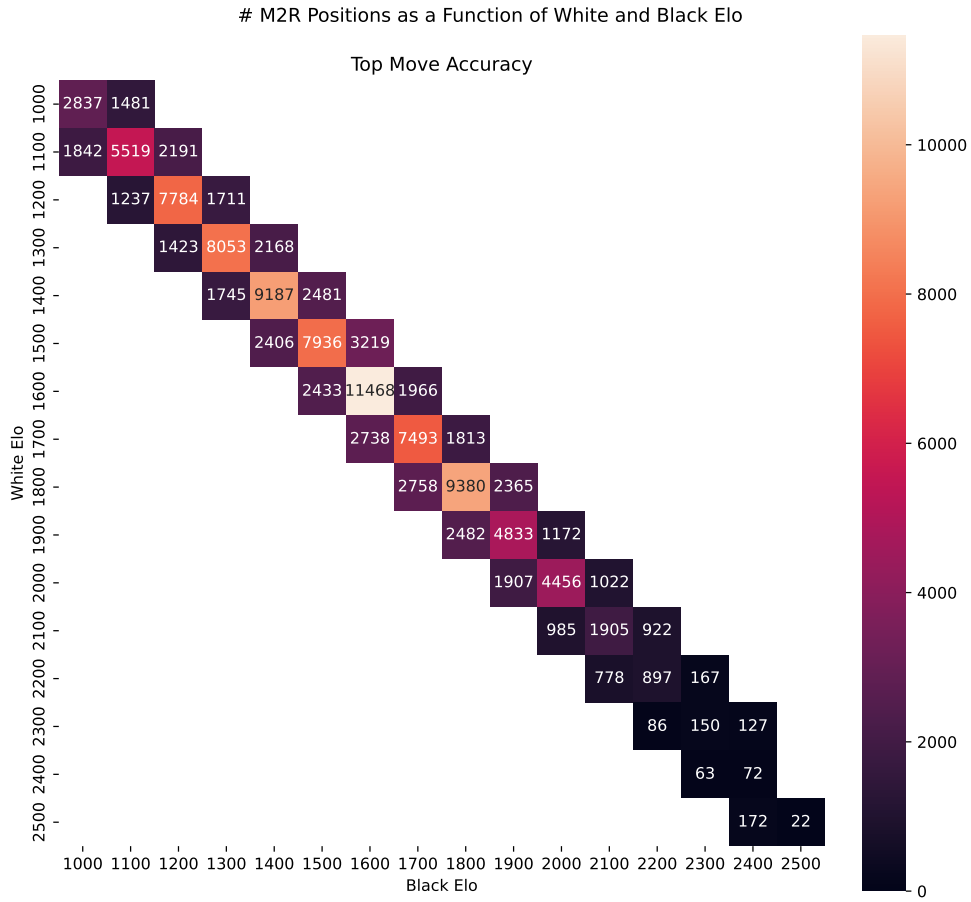

Figure D.12: Number of M2R entries in each rounded White and Black Elo bucket.

Glossary

Elo A skill rating system for players in two-player zero-sum games. Ratings are calibrated such that the expected probability of player A defeating player B is given by

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}.$$

xi, xii, 5, 6, 11, 17–20, 22, 27, 28, 39, 67, 69–74, 79, 80, 84–89, 92, 93, 97, 99, 101, 102, 104, 126–130, 135–137, 140–154

Leela Chess Zero An open-source GPU-based chess engine inspired by AlphaZero. It uses a neural network combined with Monte Carlo Tree Search to evaluate positions and select moves. Performance is typically measured in thousands of nodes per second (kN/s). xiii, xvi, 4, 6, 10, 14

Monte Carlo Tree Search A search algorithm commonly used in reinforcement learning that builds a decision tree through repeated simulations (rollouts), using sampled outcomes to estimate action values and guide exploration. xvi, 6

Stockfish An open-source CPU-based chess engine that uses a minimax search with alpha–beta pruning to evaluate game states and select moves. Since Stockfish 14, it incorporates a NNUE network to evaluate leaf positions. Performance is typically measured in millions of nodes per second (MN/s). 9, 10, 14, 19