Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering
CAI Centre for Artificial Intelligence

Projektarbeit (Informatik)

Language Models Explore the Linguistics of Chess

Autoren	Jerome Maag Lars Schmid
Hauptbetreuung	Mark Cieliebak

Datum	22.12.2023

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

Erklärung betreffend das selbständige Verfassen einer Projektarbeit an der School of Engineering

Mit der Abgabe dieser Projektarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Projektarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum:

Winterthur, 22. Dezember 2023

Unterschriften:

Jerome Maag

Lars Schmid

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Projektarbeiten zu Beginn der Dokumentation nach dem Titelblatt mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Language Models Explore the Linguistics of Chess

Authors: Jerome Maag, Lars Schmid

Supervisor: Mark Cieliebak

Institution: ZHAW

Date: 22nd December 2023

Abstract

This research presents an innovative exploration into the application of Large Language Models (LLMs) in the realm of chess, extending beyond their traditional use in natural language processing. This study departs from conventional AI approaches and explores the feasibility of employing LLMs in a structured, rule-based domain, examining their potential to generalize learning in complex systems. Utilizing a two-phased approach, the research initially employed the simpler game Connect 4 as a preliminary testbed, then extended the methodology to the more complex domain of chess. Central to these experiments was the development of a custom tokenizer for chess notation and the assembly of an extensive dataset. The study utilized the GPT-2 architecture, trained on datasets of varying sizes with chess game notations. Various evaluation metrics, including "Average Number of Correct Plies", "Hard Position Accuracy", and "Legal Piece Moves Accuracy", assessed the model's performance. These evaluations revealed a significant correlation between dataset size and model effectiveness, particularly emphasizing the importance of integrating the Beginning of Sequence (BOS) token to enhance syntactical correctness and reduce errors. While the models adeptly handled numerous chess scenarios, they faced challenges in tackling certain complex situations, providing insights into limitations and areas for future improvement. The results of the best model demonstrated an average accuracy rate of generating sequences of up to 50 correct moves, successfully illustrating the adaptability of LLMs to rule-based systems like chess, opening new avenues for AI applications in complex structured domains.

Table of Contents

1 Introduction

1.1 Glossary and Abbreviations

This section presents a concise glossary and collection of abbreviations, crucial for understanding the specialised terminology employed in this research. The aim is to clarify fundamental principles and abbreviations associated with artificial intelligence, machine learning, chess, and computational linguistics.

Table 1: Glossary and Abbreviations

Term	Definition
AI (Artificial Intelligence)	The broader field of creating intelligent machines, particularly intelligent computer programs.
AlphaZero	An AI program developed by DeepMind that uses self-play to learn chess and other games from scratch.
Beam Search	An algorithm used in machine learning to generate sequences, such as text or chess moves, by exploring the most promising options at each step.
BERT (Bidirectional Encoder Representations from Transformers)	A transformer-based machine learning technique for NLP pre-training developed by Google.
BOS (Beginning of Sequence)	A special token or marker that signifies the start of a sequence of text or data.
C4 (Connect 4)	A two-player connection game used in this research as a preliminary test before chess.
Deep Blue	A chess-playing computer developed by IBM, known for defeating world champion Garry Kasparov.
Elo Rating	The Elo rating system is commonly used in chess to calculate the relative skill levels of players.
EOS (End of Sequence)	A special token or marker that signifies the end of a sequence of text or data.
Epochs	Iterations over the entire dataset used during the training of machine learning models.
FEN (Forsyth-Edwards Notation)	A method of describing the entire board in a single line of text, capturing the precise state of the game at a given moment.
GPT / GPT-2 / GPT-3 / GPT-4	Generative Pre-trained Transformers, a series of language models developed by OpenAI[1], increasing in complexity and capability.

[1] https://openai.com/

Term	Definition
Hugging Face[2]	A company providing tools and resources for machine learning, particularly focused on natural language processing.
LAN (Long Algebraic Notation)	A method of recording chess moves in a detailed format, specifying both the starting and ending squares of a move, e.g., "e2e4".
Learning Rate	In machine learning, the learning rate is the speed at which a model learns. A higher learning rate means faster learning, but it risks missing important details, while a lower rate means slower but possibly more thorough learning.
LLM (Large Language Model)	Refers to advanced AI models capable of processing and generating human-like text, such as GPT-2 (primarily for text generation) or BERT (used for understanding the context of words in search queries).
MuZero	An AI system developed by DeepMind, capable of learning the dynamics of various games without prior knowledge of their rules.
NLP (Natural Language Processing)	A field of AI focused on the interaction between computers and human (natural) languages.
PGN (Portable Game Notation)	A standard plain text format for recording chess games, incorporating moves and other game-related data.
Ply	A term used in chess to refer to one turn taken by one of the players, with a full move consisting of a turn by each player.
PyTorch[3]	An open-source machine learning library known for its flexibility and GPU acceleration.
Reinforcement Learning (RL)	A type of machine learning where an agent learns to make decisions by performing actions and receiving feedback.
SAN (Standard Algebraic Notation)	A method of recording chess moves, using a standardized format that abbreviates each piece by a single letter and denotes the squares using a grid reference. For instance, "e4" denotes a pawn moving to the e4 square.
Temperature	In the context of AI, temperature refers to a parameter that influences the randomness of predictions by a model. Higher temperatures generally result in more varied and less predictable outputs.
Tokenization	The process of breaking down text into smaller units called tokens, which are used in language processing and model training.

[2] https://huggingface.co/
[3] https://pytorch.org/

Term	Definition
Transformer Architecture	A deep learning model architecture used extensively in modern NLP tasks. It is the basic technology behind many modern AI advancements, including the GPT series or the BERT model.
UCI (Universal Chess Interface)	A protocol for chess engines to communicate with user interfaces, allowing different chess software to interact, making it a standard in chess software development.
Weights & Biases[4]	A machine learning tool for tracking and logging model training progress and performance.
xLAN (Extended LAN)	A modified version of Long Algebraic Notation (LAN), specific to this project's methodology. In contrast to LAN, it consistently includes the piece that is moved, e.g., "Pe2e4".

1.2 Motivation

The primary goal of this research project is to examine the capacities of Large Language Models (LLMs) in a unique setting by utilising them in the field of chess. This signifies a notable deviation from their conventional uses and ventures into a domain typically reserved for specialised AI systems.

Historically, chess has been an important area for evaluating artificial intelligence. Its well-defined rules, clear outcomes, and the complexity of move sequences make it an ideal testing ground for computational models. As John von Neumann aptly stated, chess represents a form of computation where theoretically correct procedures can be determined for any given position [1]. This concept, articulated before the emergence of contemporary AI, continues to resonate in discussions surrounding machine intelligence, as exemplified in the study "Chess AI: Competing Paradigms for Machine Intelligence" [2]. The computational nature of chess provides a structured yet richly complex environment for AI research.

The transformer architecture, which is central to LLMs, has demonstrated remarkable versatility, being applied in diverse domains beyond Natural Language Processing (NLP). This adaptability is evident in its applications ranging from image generation, as seen in "Image Transformer" [3], to music production with "Jukebox" [4], and even extending to breakthroughs in protein folding predictions [5]. Investigating the application of LLMs to chess provides a unique opportunity to examine their adaptability and learning capabilities, particularly in understanding and navigating complex, rule-based systems.

This research project specifically aims to determine whether an LLM, trained exclusively on a game notation for chess like Portable Game Notation (PGN) data, can generate valid chess that adhere to the standard rules of the game. Such an investigation could provide valuable insights into the comprehension abilities of LLMs, especially in contexts governed by strict rule sets. Employing LLMs in chess represents a novel approach, diverging from traditional AI chess algorithms.

Should LLMs demonstrate proficiency in learning and applying chess rules solely from game data, it would highlight their potential to comprehend complex systems without the need for explicit rule-based programming. This could pave the way for broader applications of AI in understanding and interacting with other structured yet complex domains.

[4] https://wandb.ai/

The subsequent section will define the specific task and scope of this research in detail, addressing the key questions and objectives set for this investigation.

1.3 Task

The primary aim of this research project is to explore whether Large Language Models (LLMs) can generate valid chess moves solely based on exposure to game sequences. This challenge involves assessing the model's ability to apply chess rules without direct instruction or prior knowledge of these rules. The focus is not on the strategic quality of the moves, but rather on their validity and adherence to standard chess rules.

LLMs, particularly modern ones like GPT-4[5] or Claude[6], have gained significant recognition for creating text that is almost impossible to distinguish from human-produced content. Unlike conventional methods, LLMs approach chess as a linguistic structure, perceiving moves as elements of a language. This perspective is based on the observation that chess moves and games can be encoded in algebraic notation, a system akin to a linguistic script. By treating each move as a "word" and each game as a "sentence", LLMs can be trained to understand and predict chess moves through the same mechanisms used for language processing. This involves processing vast databases of chess games, encoded in PGN, which serves as the "corpus" for the LLMs. Through exposure to diverse game scenarios, LLMs might learn to discern patterns, strategies, and even styles, in a manner similar to how they comprehend grammar and syntax in language. At the core of this research is the question of whether LLMs, trained exclusively on game data, can learn to successfully produce valid chess moves.

Additional queries guiding this research include the model's ability to recognize both common and rare moves, such as pawn promotions or en passant captures. An important aspect of this investigation is to evaluate the model's ability to accurately model the state of the chessboard, especially as the game progresses and the number of moves increases. This ability is crucial as it directly impacts the model's effectiveness in generating valid moves.

By focusing on the GPT architecture, this study examines the effectiveness of the transformer architecture prevalent in modern LLMs in learning chess rules through pre-training with a dataset consisting of chess games. The focus is on whether the inherent capabilities of this architecture are conducive to understanding and applying the rules of a structured, rule-based game like chess.

The scope of this research is primarily focused on the model's ability to generate accurate chess moves. It is not aimed at evaluating the model's proficiency in strategic gameplay or its capacity to generate superior moves. The emphasis is on the fundamental capability of the model to understand and replicate the basic mechanics of chess based on its training with game data.

The next section will provide a detailed overview of related work in the field, offering context and background for this research project.

1.4 Related Work

In the field of artificial intelligence, chess has long been used as a testing ground for new theories and algorithms. This section presents significant milestones and current research activities related to the use of AI, in particular Large Language Models (LLMs) and transformer architectures, to understand and play chess.

[5] https://openai.com/research/gpt-4
[6] https://claude.ai/

1.4.1 Chess in AI

Historically, computer scientists have devoted significant effort to teaching machines the skill of playing chess. Known for its tremendous complexity, chess represents a major challenge for artificial intelligence, with a vast number of possible positions, estimated at around 10^{50} in the 1996 study "An Upper Bound for the Number of Reachable Positions" [6]. Consequently, chess serves as a suitable model for applying AI and machine learning techniques, especially in addressing the combinatorial explosion problem.

The intersection of chess and artificial intelligence (AI) gained significant attention with the introduction of IBM's Deep Blue. In their 2002 paper [7], Campbell, Hoane and Hsu explained the technological advances and strategic planning that resulted in Deep Blue's historic victory over world chess champion Garry Kasparov in 1997. Deep Blue, a revolutionary chess computer, utilized brute force computing techniques, evaluating up to 200 million chess positions per second. This approach, based on alpha-beta pruning and parallel processing, marked a significant milestone in AI, demonstrating a machine's capability to outperform human expertise in a complex intellectual domain. The success of Deep Blue symbolized the potential of AI in tackling problems characterized by high strategic complexity and extensive possibilities.

Following the era of Deep Blue, recent advances in AI and machine learning have introduced new paradigms in the field of chess. A key development is summarized in the 2017 paper by Silver et al. titled "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm" [8]. This study presented AlphaZero, an AI system developed by DeepMind that differs fundamentally from the Deep Blue methodology. AlphaZero used a tabula rasa approach to reinforcement learning, learning to play chess at a superhuman level through self-play alone, without human intervention, in contrast to Deep Blue's reliance on hand-crafted heuristics and extensive opening libraries. AlphaZero's methodology incorporated the use of Deep Neural Networks and Monte Carlo Tree Search (MCTS)[7], combining learning and planning in a unified framework. This system demonstrated remarkable strategic depth and creativity, often diverging from traditional human chess principles. AlphaZero's success in mastering not only chess but also shogi and go, demonstrates the versatility of its learning algorithm, exemplifying the potential of AI to generalize and adapt to different complex environments.

Building on the foundation of AlphaZero, MuZero, presented in the 2020 paper "Mastering Atari, Go, Chess, and Shogi by Planning with a Learned Model" [9], represents a significant advancement in applying AI to games such as chess. Unlike its predecessors, MuZero learns the underlying dynamics of different games without prior knowledge of their rules, combining a deep neural network with a planning algorithm to predict outcomes and develop strategies. This approach, simulating potential future game states, represents a shift away from traditional rule-based AI methods. MuZero's ability to understand and adapt to different game environments without explicit rule instructions reflects human-like cognition, illustrating the evolution of AI towards systems capable of learning and applying knowledge in different contexts. This breakthrough has implications beyond games, extending to broader applications involving complex, rule-based systems.

In summary, the evolution of AI in chess, from the strategic computing of Deep Blue to the self-learning capabilities of AlphaZero and MuZero, illustrates a trajectory of increasingly sophisticated AI methodologies. These advancements have shifted the focus from brute-force computation and rule-based programming to AI systems capable of learning, adapting, and making decisions in

[7] A heuristic search algorithm used in decision-making processes, notably in used in AI.

complex environments. This historical progression sets a foundation for exploring the application of LLMs in this domain.

1.4.2 Teaching LLMs to Play Chess

In recent years, research into the application of LLMs to chess has led to various innovative approaches and experiments. A notable study titled "Learning Chess with Language Models and Transformers" (2022) [10] utilized BERT, a notable transformer-based model, for generating chess moves, building on the foundational study "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" (2018) [11]. Instead of relying on PGN, the researchers employed Forsyth-Edwards Notation (FEN) to represent chess positions. This method of encoding diverges significantly from PGN, which focuses on the sequence of moves rather than the state of the board after each move. Including FEN into the training data means that the model developed in this project does not have to learn to derive the state of the board from the sequence of moves, as is the case for the research presented in this report.

"The Chess Transformer" (2021) [12] and "Teaching A Language Model To Play Chess" (2021) [13] explored the potential of fine-tuning GPT-2 [14] with Portable Game Notation (PGN) data. These studies were based on the hypothesis that GPT-2, a generative language model known for its human-like text generation capabilities, could be adapted to understand and replicate the structured language of chess moves. In these experiments, the model was trained with large datasets of recorded chess games in PGN format, enabling it to learn the patterns and rules inherent in chess move sequences.

Similarly, "Watching a Language Model Learning Chess" (2021) [15] experimented with various configurations of GPT-2. This project involved fine-tuning different versions of GPT-2 with varying volumes of game data to observe the evolution of the learning curves. It concluded that using more games for training significantly improves results within the same training time.

Evaluation methodologies across these studies have varied but generally encompass measuring model accuracy and game-play endurance. For instance, in the study "Learning Chess with Language Models and Transformers" [10], the trained model was used against different opponents including a rule-based "Guru Player", a "Random Player", and a "Q-Learner", additionally measuring how long the model could sustain play against a grandmaster level Stockfish engine. The study "Watching a Language Model Learning Chess" [15] evaluated the correctness of generated moves from typical opening positions, positions from actual games, and randomly generated positions.

This innovative research on instructing LLMs in the game of chess has had a significant impact on the methodology adopted in this project. By examining their methodologies and results, valuable insights were gained into the potential and challenges of applying LLMs to chess. This project's study is based on these existing foundations and seeks to expand and improve the use of LLMs in the field of chess.

1.5 Further Outline

Chapter 2 "Theoretical Foundations" provides a general overview of Large Language Models (LLMs) and their underlying transformer architecture. The mechanisms by which LLMs learn to generate text are explained, emphasizing the importance of pre-training on large datasets and the process of fine-tuning for specific applications. This chapter aims to establish a basic understanding for the subsequent experimental sections.

Chapter 3 "Connect 4 Model: Pre-Experiment" presents the preliminary experiments conducted with the game Connect 4. It includes the methodology used to generate data, as well as the results and findings from this initial experimental phase.

In Chapter 4, "Chess Model: Development and Training," the focus shifts to the more complex area of chess. This section describes the procedures for data collection and processing, including the conversion of standard chess notation into a format suitable for training an LLM. The chapter further discusses the details of training and validating the chess model.

Chapter 5, "Results", critically examines the results of the experiments using several metrics. It provides a comparative analysis of the models' performance, highlighting their capabilities in generating valid chess moves and identifying areas for improvement.

Chapter 6, "Conclusions and Prospects for Further Research" reflects on the findings, discusses their implications, and explores ideas for possible future research directions.

Chapter 7, "Directories", includes bibliographic references and a list of figures and tables. Chapter 8, "Appendix", provides technical documentation and details about accessing the project's codebase.

2 Theoretical Foundations

This chapter provides a comprehensive introduction to Large Language Models (LLMs) and its underlying transformer architecture, aiming to establish a basic understanding for the subsequent experimental parts.

2.1 Large Language Models

Large Language Models (LLMs), a prominent class of neural networks, have revolutionized the field of natural language processing (NLP). These models, known for their versatility, are designed to understand, interpret, and generate human language by training on extensive text datasets. Notable LLMs include OpenAI's[8] GPT series (e.g., GPT-2 [14], GPT-3.5 [16], GPT-4 [17]), Google's PaLM [18] and Meta's LLaMa [19]. While traditionally used in applications like chatbots, code generation, creative writing, or sentiment analysis, this project explores their potential in a novel domain: the structured, rule-based language of chess moves. The challenge lies in adapting these models, which excel in processing human language, to accurately interpret and generate sequences based on chess move language.

The development of LLMs typically involves two critical stages: pre-training and fine-tuning. During pre-training, models learn from a broad range of language data, establishing a robust foundational knowledge base. Subsequently, fine-tuning tailors these models to specific tasks or datasets, enhancing their performance in niche applications. Uniquely, this project diverges from the common approach of leveraging pre-trained models; instead, the focus is on pre-training an LLM from the ground up, aiming for a deep, specialized understanding of the language of chess without the influence of prior linguistic training.

2.1.1 Text Generation with LLMs

Text generation with LLMs, often known as inference, is an iterative process. Beginning with an input prompt, the model predicts subsequent tokens, thus constructing a coherent output one token at a time. This autoregressive method allows the LLM to produce contextually relevant text. In this project, this process is analogous to "playing out" a chess game, where the model predicts moves based on the current board state. For example, when provided with a game sequence in PGN format, the LLM will produce subsequent chess moves, thus simulating a game.

Training for this task involves exposing the model to a substantial corpus of chess game data. The model learns the inherent structure and rules of the chess game, similar to how it would learn the grammar and syntax of a natural language. This approach allows the LLM to "dream up" chess moves, reflecting the distribution of moves and strategies found in its dataset.

2.1.2 Transformer Models

Transformer models represent a significant departure from traditional neural network architectures like convolutional neural networks (CNNs) [20] and recurrent neural networks (RNNs) [21]. Introduced in 2017 [22], they have since become a cornerstone in NLP. Unlike CNNs, which excel in pattern recognition, and RNNs, which process sequential data, transformers utilize "self-attention" to weigh the importance of different parts of the input data. This feature is crucial for this project as it allows the model to understand the contextual relevance of each move in a game of chess, a task that involves complex relationships between pieces and positions.

[8] https://openai.com/

Transformer models consist of two main components: the encoder and the decoder. The encoder analyses the input data and converts it into an internal representation, while the decoder uses this representation to generate an output. These models are adept at tasks that require a deep understanding and generation of language, such as machine translation. Figure 2 illustrates the general architecture of a transformer model.

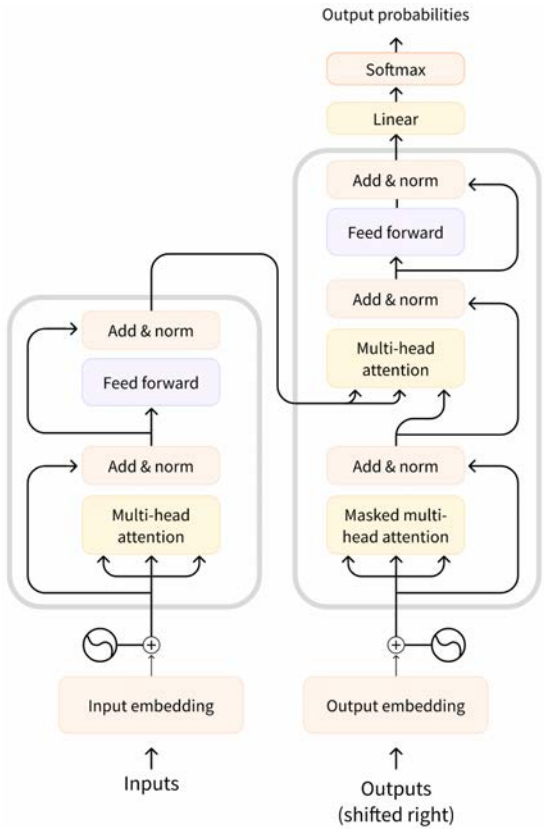

Figure 2: Transformer Model Architecture[23]

Transformers additionally feature efficient parallel processing, making them significantly faster than RNNs, which must process data sequentially. This efficiency makes them more suitable for training with large datasets.

There are three categories of transformer models: Encoder-only, Decoder-only, and Encoder-Decoder models. Encoder-only models, like BERT (Bidirectional Encoder Representations from Transformers) [11], consist solely of an encoder and are designed for tasks requiring an understanding of the input, such as sentence classification. Decoder-only models, like GPT (Generative Pre-trained Transformer) [24] [22], are optimized for generative tasks like text generation. Encoder-Decoder models, also known as Sequence-to-sequence models, are suitable for tasks involving generating output based on given input, such as machine translation. Further details about transformers can be found in the following sources: [25], [26], [27].

For the chess application developed in this project, a decoder-only model was employed, optimized for generative tasks. This choice is motivated by the aim of this project to teach a model to sequentially generate chess moves, akin to constructing sentences in a language.

2.1.3 Tokenizer

A tokenizer is a vital tool in the realm of LLMs, breaking down text into manageable units called tokens. These tokens, whether words, punctuation, or in this case, elements of a chess game, form the building blocks for model input. Advanced tokenization methods like Byte-Pair Encoding (BPE)

WordPiece, or SentencePiece, which identify frequently occurring word combinations or subwords and treating them as individual tokens, are particularly effective for complex or specialized texts. [27]

Most known models come with pre-trained tokenizers optimized for general purposes. Figure 3 shows an example of how GPT3.5 would tokenize an input string, while Figure 4 shows the resulting IDs, using OpenAI's tokenizer[9].

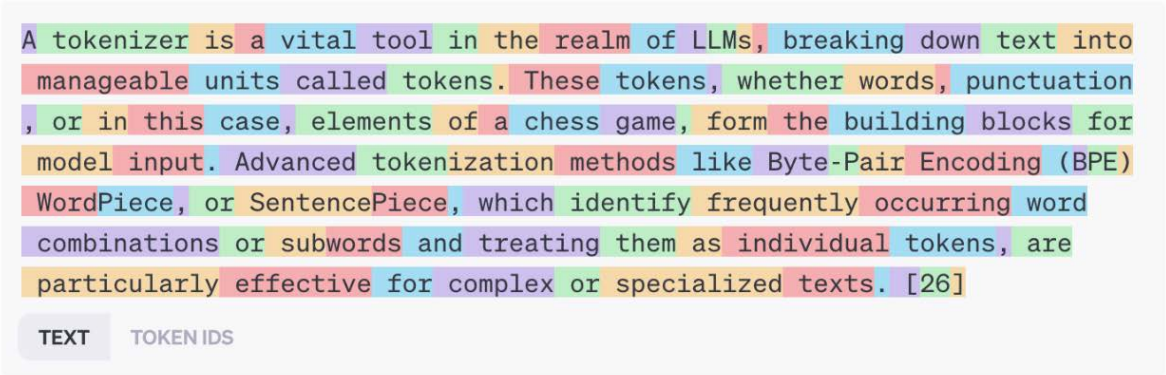

Figure 3: Example of How GPT3.5 Performs Tokenization (Text)

Figure 4: Example of How GPT3.5 Performs Tokenization (Token IDs)

Contrasting this approach, this project employs a custom tokenizer, crafted specifically for chess notation. This tokenizer recognizes and converts elements of a chess game, such as specific moves or game results, into tokens. Each token thus represents a unique aspect of the game, enabling precise and contextual analysis. For instance, the move "e4" or the game result "1-0" would each be represented by a single, distinct token. The use of a specialized tokenizer reduces the model's vocabulary size, thereby streamlining the training process and improving efficiency. Additionally, this approach ensures consistent tokenization of moves, with each ply in our xLAN notation requiring exactly three tokens, enhancing the model's ability to accurately process and generate moves. The detailed structure and implementation of this specialized tokenizer, tailored for chess notation, will be further elaborated in Chapter 4.3.1.

[9] https://platform.openai.com/tokenizer

3 Connect 4 Model: Pre-Experiment

The initial phase of this research involved utilizing the game Connect 4 as a foundational testing ground. This methodology, resembling the one described in "Learning Chess with Language Models and Transformers" (2022) [10], where the game Nim served as an initial step. Utilizing Connect 4 was essential before delving into the complexities of chess. This chapter provides a high-level overview and intentionally omits exhaustive implementation details. Many aspects of the implementation, including the development of a custom tokenizer, are similar to those detailed in Chapter 4 discussing the main experiment.

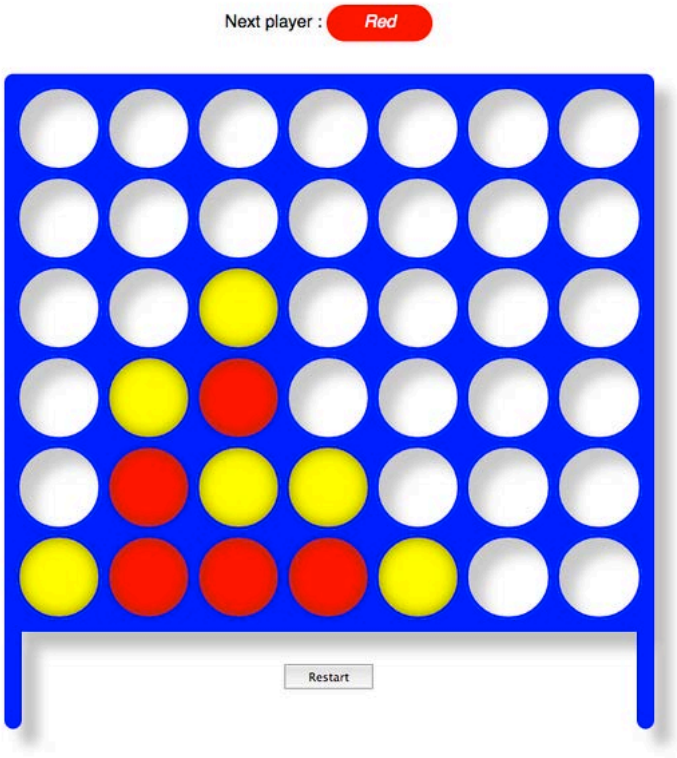

Figure 5: Connect 4 (Game Board)[28]

3.1 Connect 4

Connect 4, also known as "Four-in-a-Row,"[10] is a two-player game where the objective is to form a line of four discs of the same colour in a vertically suspended grid. The game's simplicity, characterized by its straightforward rules and limited game states, contrasts with chess's multifaceted nature. This contrast makes Connect 4 a suitable model for initial experimentation in AI, providing a more controlled environment for developing and testing algorithms.

As a computationally solved game, a term denoting the ability to predict the game's outcome from any position, Connect 4 allows for a focused exploration of game strategies within AI. This aspect was highlighted by the achievements of James D. Allen [29] and Victor Allis [30] in 1988, and further refined by Kissmann and Edelkamp in 2008 [31], who demonstrated the game's finite possibilities. The solved status of Connect 4, with approximately 4.53 x 10^{12} possible positions, offers

[10] German translation: "Vier Gewinnt"

a clear framework for testing AI models, unlike the exponentially greater complexities encountered in chess.

The utilization of Connect 4 in AI and machine learning research is well-documented [32], [33]. Its role in developing AI models, particularly those based on reinforcement learning, has been significant, often serving as an entry point for exploring more sophisticated strategic games. The evaluation of AI performance in Connect 4 typically revolves around the model's win rate, providing a measurable and straightforward metric for assessing AI capabilities.

In this project, Connect 4's reduced complexity compared to chess offers several advantages. It facilitates a high volume of experimental runs within a limited timeframe, an essential factor given the resource-intensive nature of developing and training advanced AI models. Additionally, the insights gained from the Connect 4 experiments are expected to provide valuable information regarding the feasibility and potential challenges of applying similar methods to chess. The simple nature of Connect 4 creates an ideal environment for hypothesis testing and technique refinement.

Another crucial advantage is the opportunity to develop and refine a data generation, model training, and performance validation pipeline. This established process in Connect 4 is intended to be both portable and scalable, adaptable for later application to the more intricate domain of chess.

3.2 Game Notation for Connect 4

The choice of game notation styles plays a significant role in the efficiency of the training process for language models. For Connect 4, two distinct notations were employed, each chosen for their potential application in subsequent chess experiments. The exploration of these notation styles aims to assess the impact of different levels of game notation detail on the training process of the models.

3.2.1 Long Connect 4 Notation (LC4N)

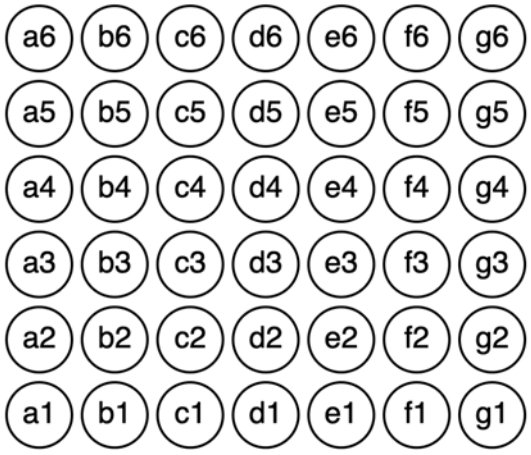

Figure 6: Connect 4 (Schematic Illustration)

The "Long Connect 4 Notation" (LC4N) is characterized by its detailed representation of each move, specifying the move number, column, and row, as shown in Figure 6. An example of a game in LC4N notation is:

```
1. f1 b1 2. a1 f2 3. g1 c1 4. b2 e1 5. b3 d1 0-1
```

This notation style results in a larger vocabulary for the model, consisting of 46 tokens including the EOS token. The comprehensive nature of LC4N could potentially increase the complexity of the model's learning process. However, it offers the advantage of being able to record each move explicitly, which is crucial for understanding the dynamics of the game.

3.2.2 Short Connect 4 Notation (SC4N)

In contrast, the "Short Connect 4 Notation" (SC4N) adopts a more concise format. It focuses mainly on the move number and the column, deriving row information from the game's rules. For instance, the above LC4N sequence in SC4N would be:

```
1. f b 2. a f 3. g c 4. b e 5. b d 0-1
```

This notation still encompasses all necessary information to define the game sequence but reduces the complexity of the notation. The vocabulary size is notably smaller, consisting of only 10 tokens, including the EOS token. SC4N's streamlined approach may offer enhanced efficiency in the model training process, making it a potentially more effective option for larger and more complex games like chess.

3.3 Data Generation for Connect 4

For training the Connect 4 model, sequences of Connect 4 games were generated using randomly selected valid moves. This approach, focusing on move validity rather than strategic depth, aligns with the objective to understand how LLMs process and generate game-related data. The generated data was categorized into datasets of different sizes (10k, 100k, 1M) to evaluate the impact of data volume on the model's performance during validation.

An additional dataset was created that encompasses all possible subsequent moves for every played move. This expansive approach intends to provide a more robust understanding of the game's mechanics, a principle that is essential when considering the application of similar methods to chess. The results and implications of employing these notations and datasets in the training process are further discussed in Chapter 3.6.

3.4 Connect 4 Model Training

In the task of training models to learn the game of Connect 4, the GPT-2 architecture was selected for its decoder-only structure. This model architecture, designed to generate sequences based on preceding context, is well-suited for Connect 4 gameplay, where each move depends on the existing state of the game. This accessibility of GPT-2[11] through Hugging Face[12] further facilitated its selection.

Several key parameters played a pivotal role in the training process. The batch size determines the number of game scenarios processed concurrently, balancing between computational demands and learning efficiency. The learning rate controls the speed at which the model adapted to new information. Finally, the number of epochs determines how often the model iterates through the entire dataset. The GPT-2 model underwent training on a dataset comprising generated sequences of Connect 4 games, as outlined in Chapter 3.3.

3.5 Connect 4 Model Validation

For validation, the dataset was divided into two subsets. The first subset was designed to assess the model's proficiency in predicting individual game moves, while the second subset concentrated on evaluating the accuracy of predicting game outcomes.

In the move prediction analysis, each game sequence in the dataset was truncated by removing the last four moves. The model's task was to predict the subsequent move based on this truncated sequence. This prediction was then verified to ensure that it adhered to the rules of the game. For

[11] https://huggingface.co/gpt2
[12] https://huggingface.co/

the winner prediction analysis, the outcome of each game was concealed. The model was prompted to predict this hidden outcome based on the remaining sequence of the game.

This method yielded two metrics, "Move Prediction Accuracy" and "Winner Prediction Accuracy", which were utilized to compare different versions of the model. Both metrics provide a value ranging from 0 to 1, with a higher value indicating superior performance. The results of these comparisons are shown in the next section.

3.6 Results

This chapter presents the results of the GPT-2 models pre-trained on Connect 4 data, assessed through the metrics "Move Prediction Accuracy" and "Winner Prediction Accuracy". The models' performances are compared according to the notation used for training (SC4N and LC4N) and the sizes of the datasets. Additionally, models trained with an expanded dataset are evaluated.

3.6.1 Evaluation of SC4N-Based Models

This section presents the validation outcomes of models trained on the game of Connect 4 using Short Connect 4 Notation (SC4N). Figure 7 showcases the model's initial learning curve and adaptation to Connect 4 using a dataset of 10,000 games across five episodes.

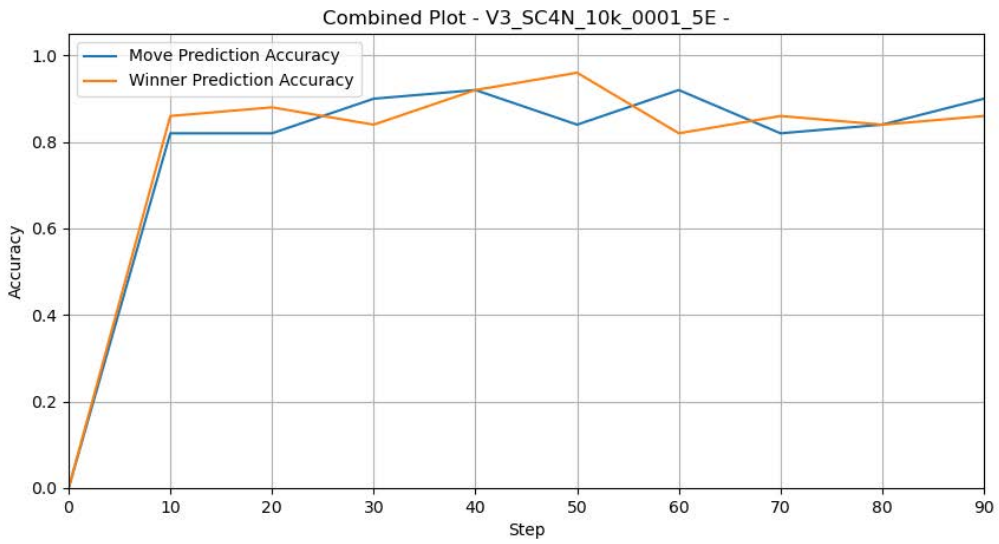

Figure 7: Results for Connect 4 Model (10,000 Games, 5 Episodes, SC4N)

Figure 8 illustrates the impact of extended training, showing the model's validation results after training on 10,000 games for fifty episodes. This demonstrates the benefits of longer training durations. Despite using the same data volume as in Figure 7, the model achieves significantly better results, comparable to those of models trained with ten times the data but fewer episodes.

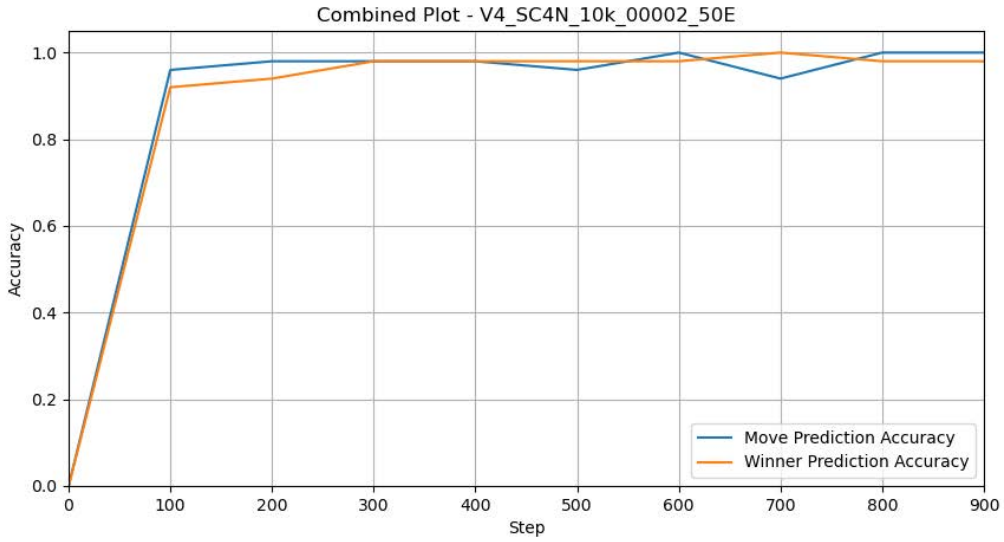

Figure 8: Results for Connect 4 Model (10,000 Games, 50 Episodes, SC4N).

Figure 9 and Figure 10 explore the effects of larger datasets, displaying the model's performance when trained on 100,000 games for twenty episodes and 1,000,000 games for five episodes, respectively. These figures indicate that increased training data volume clearly improves the model's accuracy and learning efficiency.

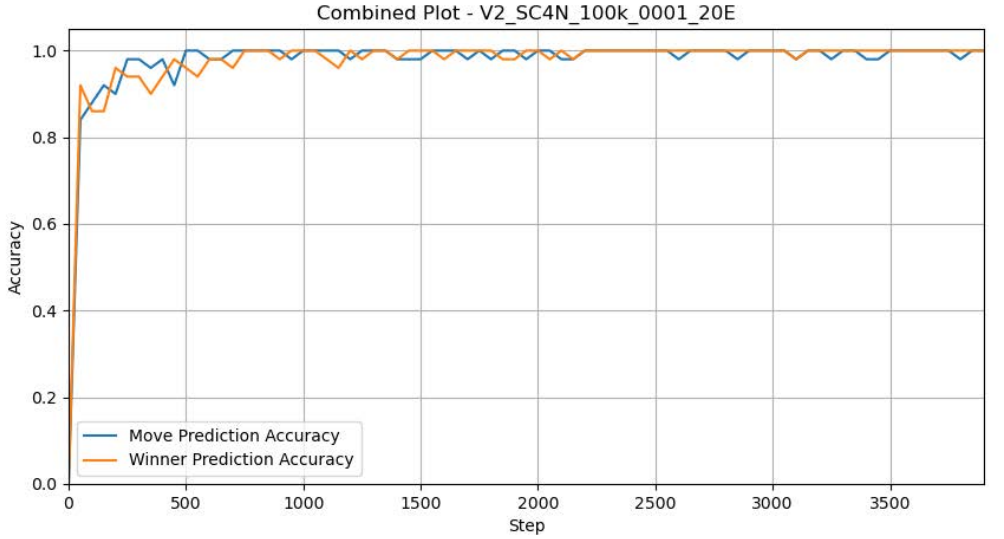

Figure 9: Results for Connect 4 Model (100,000 Games, 20 Episodes, SC4N)

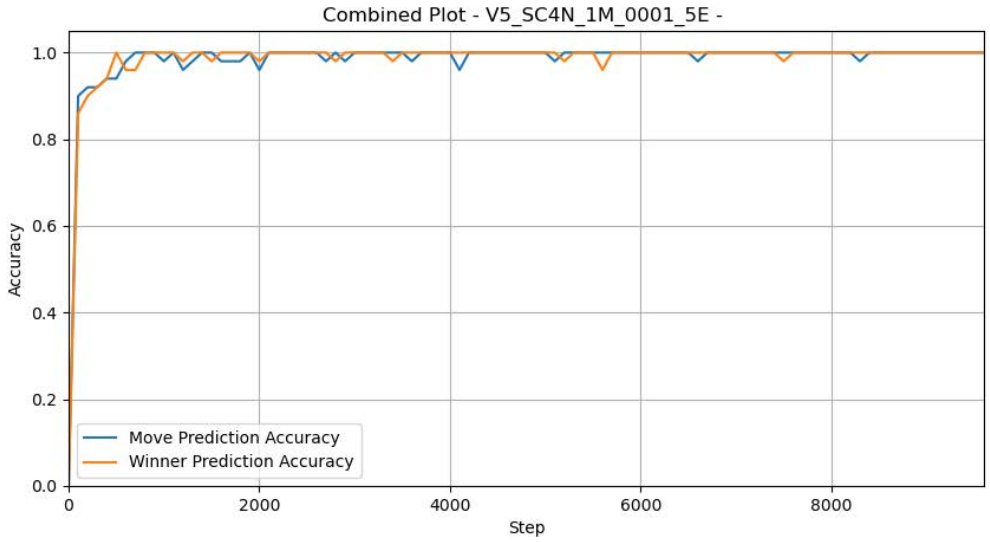

Figure 10: Results for Connect 4 Model (1,000,000 Games, 5 Episodes, SC4N)

3.6.2 Evaluation of LC4N-Based Models

Figure 11 illustrates the model's performance trained on 10,000 games over five episodes using Long Connect 4 Notation (LC4N), offering insights into its effectiveness in learning Connect 4 strategies from a limited dataset.

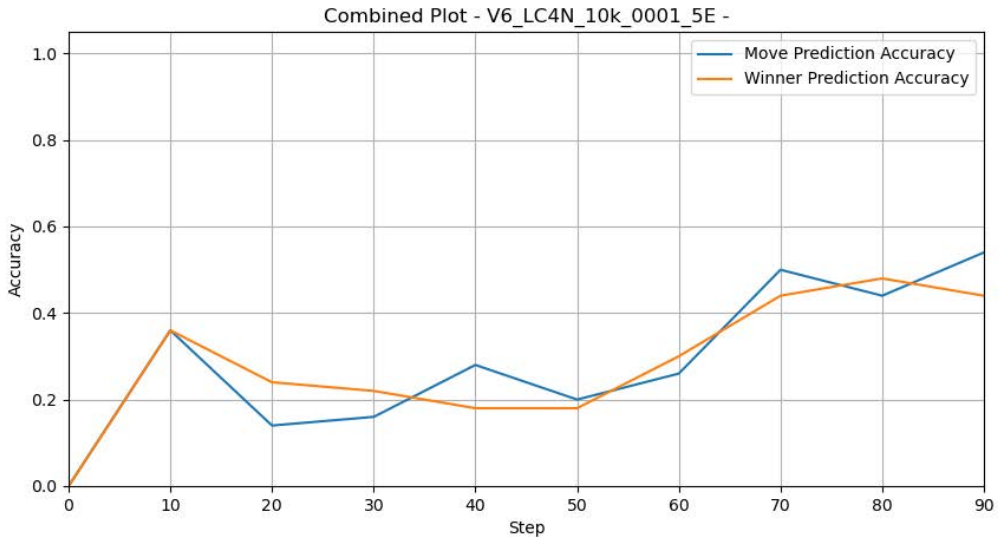

Figure 11: Results for Connect 4 Model (10,000 Games, 5 Episodes, LC4N)

Figure 12 focuses on the effect of training duration, depicting the model's validation results after training on 10,000 games for twenty episodes. Similar to the SC4N notation, the number of training episodes significantly influences the results.

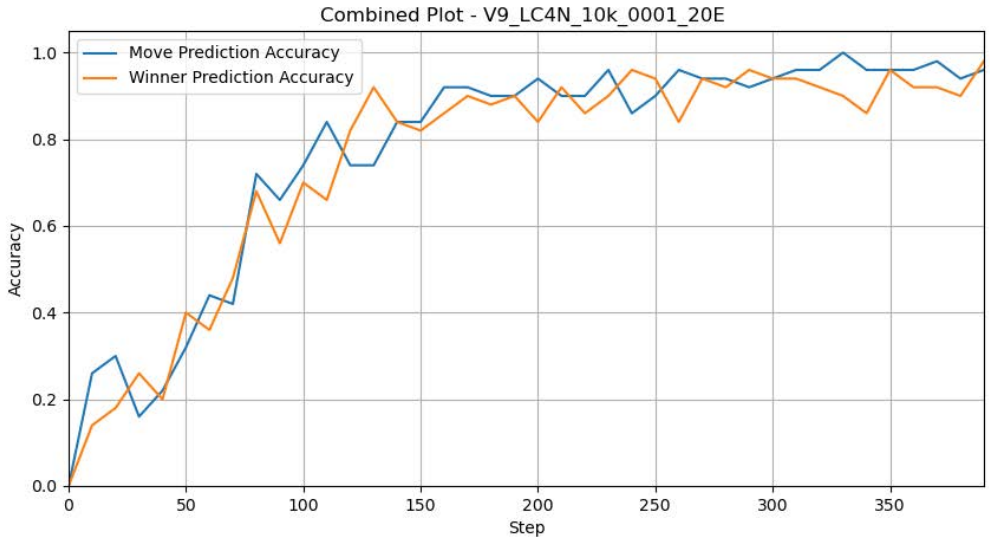

Figure 12: Results for Connect 4 Model (10,000 Games, 20 Episodes, LC4N)

Figure 13 examines the impact of a tenfold increase in training data, showing the model's validation results after training on 100,000 games for five episodes.

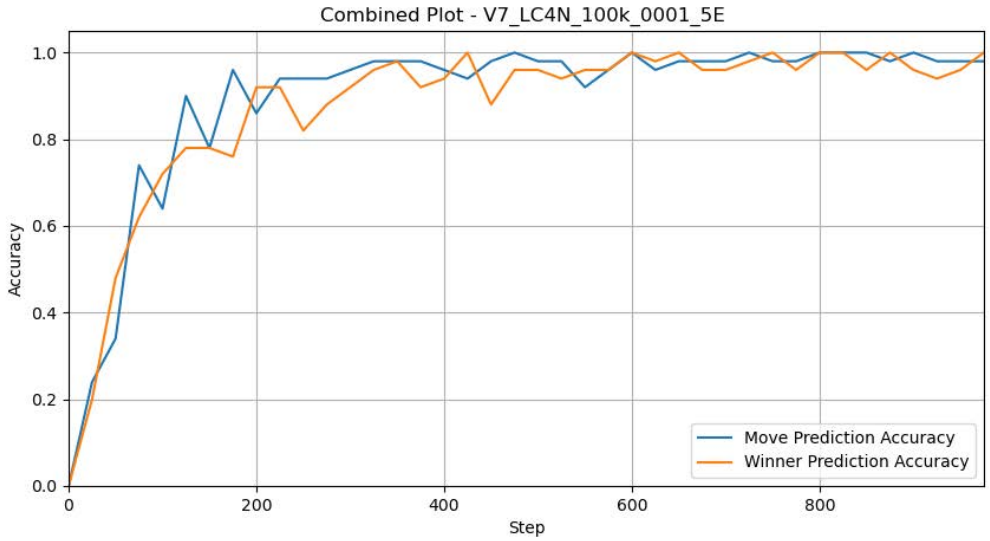

Figure 13: Results for Connect 4 Model (100,000 Games, 5 Episodes, LC4N)

Figure 14 presents the outcomes of the model trained on a substantially larger dataset of 1,000,000 games for five episodes.

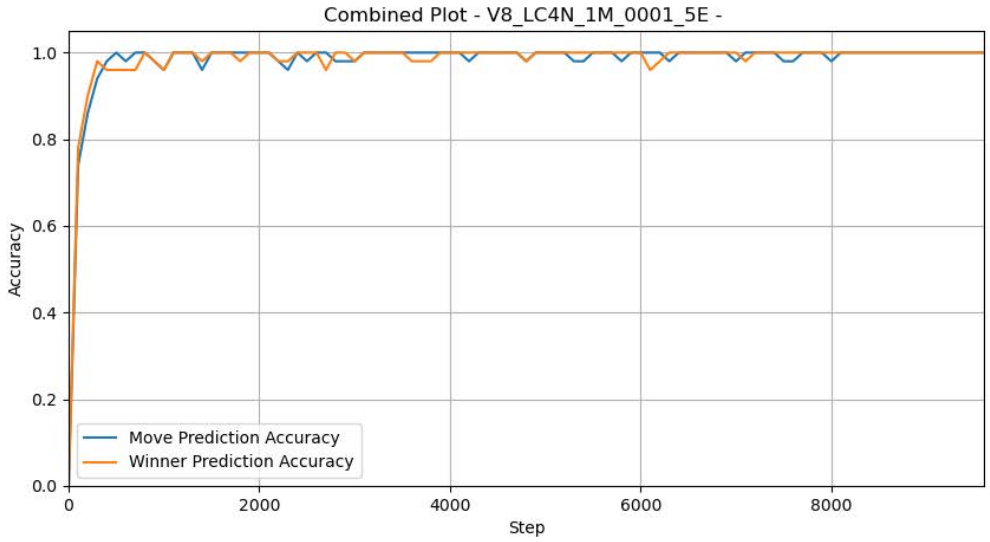

Figure 14: Results for Connect 4 Model (1,000,000 Games, 5 Episodes, LC4N)

3.6.3 Evaluation of Expanded Dataset Models

Figure 15 presents the metrics "Move Prediction Accuracy" and "Winner Prediction Accuracy" for the model trained with 100,000 games using the dataset expanded to include all possible moves, as described in Section 3.3.

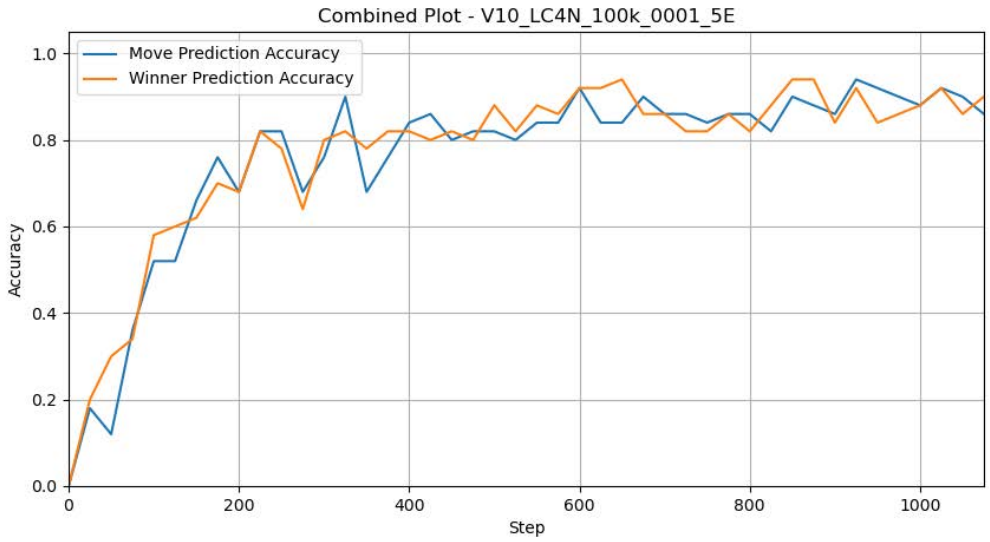

Figure 15: Results for Connect 4 Model (100,000 Games, 5 Episodes, LC4N, Dataset Including Games Expanded to Encompass all Possible Moves)

Figure 16 illustrates the results for the model trained on an expanded dataset consisting of 1 million game sequences.

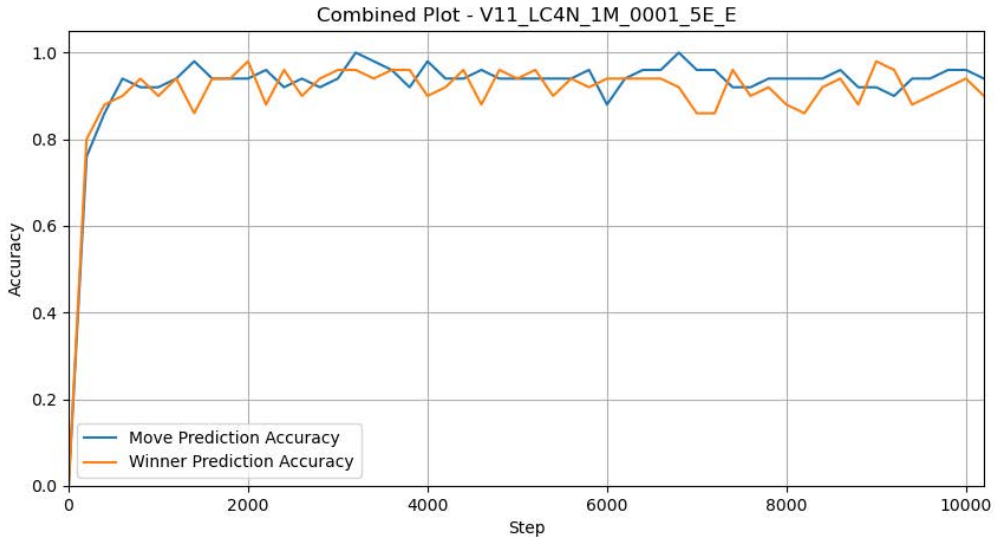

Figure 16: Results for Connect 4 Model (1,000,000 Games, 5 Episodes, LC4N, Dataset Including Games Expanded to Encompass all Possible Moves)

For both models, the data indicates, as observed in previous models, that more data typically leads to better results. However, when compared to datasets using "normally generated" games (i.e., not expanded to include all possible moves), the performance of these models is less effective. This underperformance may be attributed to the expansion process, which leads to several repeated sequences in the dataset, potentially hindering the model's ability to learn significant generalisations.

3.6.4 Model Evaluation Comparison

Table 2 provides a comparative overview of the Connect 4 models as described in the previous sections, trained using different notations, datasets, and training configurations. Each model is evaluated based on two key performance metrics: "Move Prediction Accuracy" and "Winner Prediction Accuracy".

Table 2: Overview of the evaluation of the trained Connect 4 Models

Notation	Expanded to include all moves	Number of Games in Training Data	Number of Episodes	Move Prediction Accuracy	Winner Prediction Accuracy
SC4N	No	100k	20	1	1
SC4N	No	10k	5	0.9	0.86
SC4N	No	10k	50	1	0.98
SC4N	No	1M	5	1	1
LC4N	No	10k	5	0.54	0.44
LC4N	No	100k	5	0.98	1
LC4N	No	1M	5	1	1
LC4N	No	10k	20	0.96	0.98
LC4N	Yes	100k	5	0.86	0.9

LC4N	Yes	1M	5	0.94	0.9

3.6.5 Interpretation of Results

This section analyses the results obtained from training models on Connect 4 using both SC4N and LC4N notations, focusing on the interplay between notation complexity, vocabulary size, and training data volume.

The model trained with SC4N notation showed better performance on smaller datasets (10,000 games) compared to the LC4N model. This observation is based on the results in Figure 7 for SC4N, and Figure 11 for LC4N. The SC4N model demonstrated higher accuracy in both "Move Prediction Accuracy" and "Winner Prediction Accuracy" with limited data, indicating a more efficient learning capability. This finding is significant, especially considering that SC4N, with its smaller vocabulary, seems to quickly adapt to the game's structure.

In contrast, the larger vocabulary of LC4N (46 tokens) compared to SC4N (10 tokens) increases complexity. Each additional token in LC4N increases the learning challenge, requiring more training data to obtain the same accuracy. This higher complexity is reflected in the LC4N model's lower initial performance with 10,000 games, as it struggles to capture the game dynamics as effectively as the SC4N model. The impact of training duration on performance is evident in Figure 8 for SC4N and Figure 12 for LC4N. Given Connect 4's relative simplicity, the effectiveness of SC4N in capturing game dynamics with less data might not directly translate to more complex games like chess. This correlation between notational complexity and learning efficiency in games with higher complexity merits further investigation.

In conclusion, the models trained on 1 million games successfully learned to produce moves adherent to the rules of Connect 4, while the models trained with 100,000 games also achieved a high level of accuracy. In summary, the lower complexity and smaller vocabulary of SC4N appear to offer advantages in Connect 4. However, applying these findings to chess requires careful consideration due to the greater complexity of the game.

3.7 Challenges and Limitations

The initial evaluation of the results for the Connect 4 models indicated that models trained with LC4N performed significantly better than those trained with SC4N. This observation prompted the decision to conduct chess experiments exclusively using xLAN, an equivalent to LC4N. However, subsequent re-evaluations of the models, as presented in the previous sections, did not replicate these initial results.

4 Chess Model: Development and Training

Building on the foundational experiments with Connect 4, the second part of this project focuses on the development and training of a model for the more complex game of chess. This experiment applies lessons learned from Connect 4 to explore the capabilities of Large Language Models (LLMs) with this more complex game.

Chess, unlike Connect 4, involves a larger number of pieces, each with unique movement rules, and a larger game board. This leads to a significantly greater number of possible positions and outcomes, challenging an LLM's ability to generate valid moves, understand the underlying principles of the game, and memorize board positions.

Maintaining consistency with the Connect 4 methodologies, the approach for the chess experiments revolves around pre-training an LLM exclusively on chess game notation. This training is conducted without direct instructions or embedded knowledge of the game's rules, focusing on a notation format specifically tailored for this purpose.

4.1 Chess Notation

Recording chess moves in a standardized manner is crucial for analysing, digitally processing, and sharing games across diverse platforms. This chapter discusses two primary forms of chess notation: Portable Game Notation (PGN) and Long Algebraic Notation (LAN), each serving different purposes. Further, it introduces the novel Extended Long Algebraic Notation (xLAN), a notation developed specifically for this project. Later in this chapter, the importance of tokenizing the xLAN data will be detailed, a crucial step in preparing the dataset for training the model.

4.1.1 Portable Game Notation (PGN)

PGN serves as the universal format for representing chess games in textual form, allowing easy sharing of games and positions. It employs Standard Algebraic Notation (SAN) for moves, with additional tags to provide game-related metadata, such as player names, locations, and event dates.

For example, a game in PGN format may appear as follows:

```
[Event "World Championship"]
[Site "Moscow URS"]
[Date "1985.10.15"]
[Round "16"]
[White "Karpov"]
[Black "Kasparov"]
[Result "0-1"]

1. e4 c5 2. Nf3 e6 3. d4 cxd4 4. Nxd4 Nc6 5. Nb5 d6 ...
```

The sequence of moves describes the actions taken in the game, starting with the piece moved, indicated by a single-letter abbreviation (except for pawns), and the square it moved to. Special moves are marked with symbols (e.g., "+" for check, or "#" for checkmate). PGN is the most widespread format for archiving and distributing notable chess games. A formal syntax for PGN notation can be found in the appendix (Chapter 0).

4.1.2 Long Algebraic Notation (LAN)

LAN offers a more explicit representation of chess moves compared to PGN. Each move in LAN details both the starting and target squares, providing a clear and direct interpretation, particularly beneficial in digital chess formats. For instance, the move "Pawn from e2 to e4" is notated as "e2e4" or "e2-e4" in LAN. Captures are indicated with an "x" between squares, e.g., "e4xd5" means a piece on "e4" captures the piece on "d5".

A game in LAN format could look as follows:

```
1. e2e4 c7c5 2. Ng1f3 e7e6 3. d2d4 c5xd4 4. Nf3xd4 Nb8c6 5. Nd4b5 d7d6 ...
```

Despite its detailed and more explicit nature, LAN is less frequently employed in conventional chess recordings due to its verbosity. However, in digital chess contexts, LAN ensures a clear, precise, and direct interpretation of moves by various software applications and provides an efficient format for programming purposes.

In the next section, a derivative notation system, termed Extended LAN (xLAN), is introduced, developed specifically for the purpose of training the LLM for chess.

4.1.3 Extended Long Algebraic Notation (xLAN)

The introduction of xLAN, an adaptation of LAN, stems from the need for a uniform and fixed-length format that simplifies the prompting process for the LLM. Unlike standard LAN, xLAN consistently specifies the piece type for each move, resulting in a standard three-token structure per move: {piece}{start_square}{end_square}.

For example, the conversion from PGN to LAN and subsequently to xLAN can be illustrated as follows:

PGN:	1. g3 e6	2. Bg2 Qf6	3. f4 Bc5	4. e4 Qd4	5. e5 Qf2# 0-1
LAN:	1. g2-g3 e7-e6	2. Bf1-g2 Qd8-f6	3. f2-f4 Bf8-c5	4. e2-e4 Qf6-d4	5. e4-e5 Qd4-f2# 0-1
xLAN:	1. Pg2-g3 Pe7-e6	2. Bf1-g2 Qd8-f6	3. Pf2-f4 Bf8-c5	4. Pe2-e4 Qf6-d4	5. Pe4-e5 Qd4-f2# 0-1

Castling moves are converted to a format that explicitly indicates the king's movement in the format {king}{start_square}{end_square}. Pawn promotions are represented in the format {goal_piece}{start_square}{end_square}. It is important to note that the information about captures, checks, and checkmates is not contained in this notation. The model will have to learn that a piece got captured by keeping track of the board state, which will be evaluated specifically, as described in Section 4.5. An alternative suggestion for extending the xLAN notation to include this type of information is described in Section 6.2.2.

4.2 Data Collection

The dataset for training the chess models was sourced from online games, played on the "Lichess"[13] platform, specifically from games played in September 2023. Accessible through the Lichess database[14], this dataset represents the most current collection of game data, comprising 93,218,629 games played by users during September 2023 on the platform. The PGN[15] format of this dataset includes detailed information such as move sequences, links to online games, dates, times, player

[13] https://lichess.org/
[14] https://database.lichess.org/
[15] See Section 4.1.1 for more information about PGN.

names, Elo ratings, openings played, time controls, and game termination modes (e.g., time forfeit or conventional chess game endings like checkmate).

4.2.1 Data Filtering and Conversion

This dataset was refined to include only games concluding through standard chess endings, crucial for training the model in recognizing legitimate game conclusions. Games concluding due to time expiration were excluded to prevent the model from learning incorrect game terminations. Post-filtering, the dataset, which initially had a 65% rate of conventional endings, was reduced to 24,237,424 games. Notably, every 28,000th game ended due to a player being banned from the platform.

Using the Python chess library[16], the dataset was converted into xLAN[17] format, though some viable games were lost due to conversion limitations. The resulting dataset, after eliminating duplicates, comprised 23,521,034 games.

4.2.2 Preparation of the Tokenized Dataset

The dataset was tokenized for model training (details in Chapter 4.3.1), including games up to 601 plies. Given the model's maximum sequence length of 512 tokens[18], games exceeding 500 (166 plies considering that one ply consists of three tokens[19]) were excluded. This reduction led to a final dataset of 23,340,296 games.

Various subsets were created for training, based on opening sequences, to introduce more diversity. Four datasets were created with opening sequences of varying lengths, retaining only one game from each sequence. By keeping a single game after the first 6 moves, a dataset of 1,028,170 games, forming the 1M dataset. Retaining a game after 5 plies the 350k dataset, and with 4 plies, the 71k dataset was formed. For the 19k dataset, division was made after 10 tokens, corresponding to 3 plies and one piece.

4.2.3 Dataset with Expanded Moves

Mirroring the Connect 4 approach[20], a unique dataset was generated by expanding games to include all legal subsequent moves from each game position, significantly increasing the amount of data derived from a single game. On average, each position in a game offers approximately 30 legal moves, which results in a 50-move game (100 plies) generating up to 3,000 variations. Unlike the other datasets, most games in this expanded dataset do not have a termination.

To create this dataset, a diverse and comprehensive range of 200 games was manually selected. Every legal move following each position within these games was added to the dataset. This process resulted in a dataset of 270,000 games.

4.2.4 Comparison of Datasets

Table 3 and Table 4 provide comprehensive overviews of the datasets. Table 3 describes dataset sizes and formats, while Table 4 focuses on dataset characteristics like game lengths, win rates, and draw percentages. The conversion from PGN to xLAN format notably influenced draw rates, attributed to the exclusion of games with agreed draws. The length of the games in the dataset with

[16] https://python-chess.readthedocs.io/
[17] See Section 4.1.3 for more information about xLAN.
[18] See Section 4.4.1 for more information about model parameters.
[19] See Section 4.3.1 for more information about tokenization.
[20] See Chapter 3.3.

expanded moves is nearly half as long as in the other datasets. This is attributed to most games not being played to completion; after each move, all possible positions following legal moves are included, leading to a large number of shorter games.

Table 3: Overview of the Datasets Used to Train the Chess Models

Dataset Description	Dataset Name	Format	Size (Number of Games)	Size (MB)
Raw Lichess dataset September 2023	Lichess Dataset	PGN	93,218,629	ca. 201,040
Conversion to xLAN, removing games without clear ending	xLAN Dataset (with duplication)	xLAN	24,237,424	14,420
Removing duplicated lines	xLAN dataset (without duplication)	xLAN	23,521,034	14,390
Tokenize dataset	Tokenized Dataset (including long games)	Tokens	23,521,034	12,420
Removing all games with more than 500 Tokens	Tokenized Dataset (excluding long games)	Tokens	23,340,296	12,180
Keeping only one game for the first 18 Tokens	1M Dataset	Tokens	1,028,170	524
Keeping only one game for the first 15 Tokens	350k Dataset	Tokens	345,351	175
Keeping only one game for the first 12 Tokens	71k Dataset	Tokens	71,641	36
Keeping only one game for the first 10 Tokens	71k Dataset	Tokens	19,383	10

Table 4: Dataset Characteristics

Dataset	Size (Number of Games)	Median Length (Number of Plies)	Longest Game (Number of Plies)	White wins (%)	Black wins (%)	Draw (%)
Lichess	93,218,629	62	601	49,8	46,6	3,6
xLAN	23,521,034	63	601	50,0	44,7	5.3
Tokenized	23,340,296	63	166	50,1	44,8	5,1
1M	1,028,170	62	166	48,5	46,6	4,9
350k	345,351	62	166	50,7	43,9	5,4
71k	71,641	63	166	47,0	46,6	5,4
19k	19,383	63	166	49,2	44,0	6,7

Expanded moves	270,877	33	170	not included in dataset	not included in dataset	not included in dataset

Table 5 reveals a significant shift in the diversity of openings when transitioning from larger to smaller datasets. Common openings like "Pe2e4" are predominant in larger datasets such as Lichess and Tokenized, but their frequency diminishes in smaller datasets. For instance, the sequence "Pe2e4 Pe7e5 Ng1f3" appears in every sixth game of the tokenized dataset and is prevalent in larger datasets. However, in the 1M dataset, where only one game is retained after the first 6 plies, it occurs in less than one in every 100 games. In the smallest dataset (19k), this sequence appears in only 0.03% of the games, equating to only 5 games in the entire dataset. This diversity is essential for a chess model designed to understand and predict a wide range of game strategies and scenarios, as it prevents overfitting to a few common openings and encourages learning from a broader spectrum of game situations.

In the dataset with expanded moves, the variety of moves is significantly smaller compared to other datasets. This is due to the fact that many additional sequences are generated from the initial game sequence, which then end up being very similar to each other.

Table 5: Occurrence of Opening Sequences in Percentage (%)

Opening Sequence (xLAN moves)	**Lichess Dataset**	**Tokenized Dataset**	**1M Dataset**	**350k Dataset**	**71k Dataset**	**19k Dataset**	**Expanded Moves Dataset**
Pe2e4	58.5	60.3	26.5	19.6	12.4	9.2	53.3
Pe2e4 Pe7e5	23.2	26.6	5.2	2.9	1.1	0.7	22.6
Pe2e4 Pe7e5 Ng1f3	14.0	15.8	0.7	0.2	0.04	0.03	10.8
Pd2d4	25.3	24.0	18.5	14.5	10.1	7.9	32.2
Pd2d4 Pd7d5	10.3	10.4	3.7	2.1	0.9	0.6	18.9
Pd2d4 Pd7d5 Pc2c4	3.5	3.4	0.3	0.1	0.04	0.03	7.4
Pe2e4 Pc7c5 Ng1f3 Pd7d6 Pd2d4 Pc5d4	0.8	0.6	0.01	0.01	0.01	0.01	2.3

4.3 Data Processing

This section delves into the specifics of data processing, a critical phase that prepares the dataset for effective model training. The process involves transforming the raw game data into a structured format that the model can efficiently learn and interpret.

4.3.1 Chess Data Tokenization

The process of tokenization plays a significant role in the pre-training of LLMs. Tokenization, as outlined theoretically in Section 2.1.3, involves breaking down text into smaller constituent units

called tokens. These tokens form the vocabulary that the model learns and utilizes to generate outputs.

In this project, custom tokenization was employed, tailored specifically for the domain of chess move generation. This customized approach, in contrast to standard tokenizers like byte-level Byte-Pair-Encoding used in GPT-2, provides precise control over the model's outputs, which will prove particularly useful for the validation phase. It is also specific to the area of chess move generation and differs significantly from typical LLM training in the domain of natural language texts.

The vocabulary was defined manually based on the Extended Long Algebraic Notation (xLAN) described in Section 4.1.3. It encompasses the following elements:

- **Pieces:** Each type of chess piece (King, Queen, Rook, Bishop, Knight, Pawn) is represented by a distinct token.
- **Squares:** Every square on the chessboard has a unique token, covering all 64 squares.
- **Game Results:** Specific tokens denote game outcomes, such as "1-0" for a win by White, "0-1" for a win by Black, and "1/2-1/2" for draws.
- **Special Tokens**: These include tokens indicating the start and end of a game and a padding token for alignment.

The tokenization structure, depicted in JSON format, is as follows:

```
{
    "paddingToken": {"PADDING": 0, "BOS": 75},
    "pieces": {"K": 1, "Q": 2, "R": 3, "B": 4, "N": 5, "P": 6},
    "squares": {"a1": 7, "a2": 8, ..., "h7": 69, "h8": 70},
    "results": {"1-0": 71, "0-1": 72, "1/2-1/2": 73},
    "gameSeparator": {"EOS": 74}
}
```

In practice, each chess move in the dataset is deconstructed into a sequence of tokens according to this defined vocabulary. For instance, the move "Pe2e4" (a pawn moving from the square "e2" to "e4") is segmented into three tokens: "P" for Pawn, "e2," and "e4". This method is uniformly applied across the dataset to ensure that the model internalizes a standardized representation of chess moves.

This custom tokenization approach offers several benefits. It ensures the model's outputs adhere strictly to the defined chess vocabulary, crucial for accurately assessing its performance[21]. Additionally, the relatively compact size of the vocabulary can potentially speed up the model's learning process, making it more efficient in understanding and replicating chess moves.

4.4 Chess Model Training

In this project, the GPT-2[22] architecture (Generative Pre-trained Transformer 2) was selected to train the LLM specifically for chess game notation. Known for its effectiveness in text generation and comprehension, GPT-2 offers a robust framework for decoding complex language patterns.

The decoder-only structure of GPT-2 makes it particularly well-suited for tasks involving sequential prediction and generation. This aligns perfectly with the objectives of the chess LLM, which aims to generate the next move based on the game's progression. GPT-2 excels in generating sequential data

[21] See Chapter 4.5.

[22] https://github.com/openai/gpt-2

and predicting subsequent tokens based on prior context. While models like BERT or BART are powerful in their respective domains, their bidirectional or encoder-focused architectures are not primarily designed for sequential data generation, making GPT-2 a more suitable choice. The popularity and widespread use of GPT-2, coupled with its easy integration and customization through the Hugging Face platform, further reinforces this choice.

For the model training, the "transformers" library from Hugging Face[23] was used. This library offers a straightforward interface for training a Language Model (LLM), direct access to the GPT-2 model, and a simple class for model validation during training. PyTorch[24], a widely used open-source machine learning library known for its flexibility and GPU acceleration, was utilized in conjunction with the "transformers" library. The training process was monitored and logged using Weights & Biases[25], enabling effective tracking of the model's performance and facilitates comparison of different models.

4.4.1 GPT-2 Model Configuration

Key aspects of the model configuration included:

- **Vocabulary Size:** Adjusted to encompass 75 or 76 tokens, including the Beginning of Sequence (BOS) token, corresponding to the tokens defined in the custom tokenizer for chess games (see Section 4.3.1).
- **Max Sequence Length:** Set at 512, this length ensures a balance between capturing sufficient game depth with computational efficiency, enabling the model to process and predict longer sequences of chess moves while managing memory requirements.
- **Special Tokens:** Incorporated to denote the start and end of a game, guiding the model on when to stop predicting or start a new game.
- **Model Size and Layers:** Utilized the standard GPT-2 structure, comprising 768 embeddings, 12 hidden layers, and 12 attention heads.
- **Activation function:** Employed the "gelu_new" activation function, a variant of the Gaussian Error Linear Unit (GELU), was used.

4.4.2 Hardware and Training Duration

The training hardware included a Windows 10 system equipped with an NVIDIA RTX 3090 graphics card (24 GB memory), an Intel Core i7-13700k processor, and 64 GB of RAM. The setup provided 3 TB of storage for the models and training data.

Training durations varied across datasets. Approximate training times for 4 epochs: the 19K dataset took around 30 minutes per session; the 71K dataset, about 1 hour and 40 minutes: the 350K dataset, roughly 8 hours and 30 minutes; and the 1 million data set, approximately 24 hours per session.

4.5 Chess Model Validation

This section details the methodology used to evaluate the trained models' ability to generate valid chess moves. It encompasses both quantitative and qualitative analyses, assessing the model's adherence to chess rules and their comprehension of complex positions.

[23] https://huggingface.co/docs/transformers/index
[24] https://pytorch.org/
[25] https://wandb.ai/

4.5.1 Model Performance Metrics

The validation phase employed various metrics to evaluate the performance of the models, assessing aspects like move accuracy, rule adherence, and predictive consistency. These metrics fall into three main categories:

- **Syntactical Analysis**: Evaluates if the generated moves adhere to the Extended Long Algebraic Notation (xLAN) format, ensuring syntactical accuracy within chess notation.
- **Semantical Evaluation**: Determines whether the moves are valid according to chess rules, including piece movement legality, game-specific rules adherence (e.g., castling, en passant), and board state accuracy.
- **Statistical Analysis**: Focuses on the distribution and frequency of different error types, such as syntax, piece logic, and path obstruction errors, in various scenarios.

Notably, the standard language model metric of perplexity was deemed less relevant here due to the rule-based nature of the output required in chess.

4.5.2 Average Number of Correct Plies

A central aspect of the evaluation involves having the model simulate games by generating moves for both players. This evaluation metric involved generating 100 game sequences of up to 80 plies each, starting from the initial board setup. The sequence was halted as soon as an error occurred, with the number of correctly generated plies recorded for subsequent computation of the average across all games. The model was prompted with a temperature value of 0.7, which influences the randomness of move selection, and the Beginning of Sequence (BOS) token, which signals the start of a new sequence generation. The assessment involved calculating the average number of correctly generated plies and categorizing errors[26] into specific types as follows:

- **Syntax Error**: Incorrect move format (i.e., not corresponding to the format `{piece}{start_square}{end_square}`), indicating a misunderstanding of chess notation.
- **Piece Logic Error**: Illogical or impossible moves for a given piece.
- **Path Obstruction Error**: Moves that fail to account for blocking pieces on the board.
- **Pseudolegal Errors**: Correct piece movements that violate other chess rules (e.g., castling through check or moving a pinned piece).
- **No Error:** The model generated a game of 80 plies correctly or the game terminated with a clear outcome in less than 80 plies.

4.5.3 Hard Position Accuracy

This metric evaluated the models' handling of 67 manually selected challenging positions, including unusual castling scenarios, pawn promotions, or positions requiring specific moves to avoid checkmate. Categories included:

- **Never Seen Openings:** This category includes starting sequences not present in the training data. Evaluating the model's move generation starting from these sequences assesses its ability to generalize, rather than solely relying on memorization from its training.
- **Long Games:** This category consists of starting sequences longer than 40 moves (80 plies), challenging the model to maintain an understanding of the board state over an extensive number of moves.

[26] These error categories have been slightly adapted from the categories used in the study "Chess as a Testbed for Language Model State Tracking" (2022) [34].

- **Checks:** These positions test the model's understanding of rules involving checking the opponent. Situations include preventing valid castling moves because they would place the player in check, forcing king moves, the necessity to block an opponent's checking piece, or handling discovered checks.
- **Castle:** Similar to the "Checks" category, these positions evaluate the model's comprehension of specific scenarios involving castling. Some positions may restrict castling due to an enemy piece's influence, or castling might be invalid if the player has already moved either the king or the rook.
- **Pawns:** This category involves challenging positions related to pawns, such as a pawn being blocked by another pawn of the same colour or by an enemy piece, scenarios involving en passant moves, or even forced en passant situations.

To evaluate a trained model on such a position, a starting sequence is given to the model, from which it is prompted to generate the next ply. If the model successfully generates any one of the legal moves in this position, it counted as "passed". The resulting value for this metric is a number ranging from 0 to 1, representing the percentage of positions the model could solve. A selection of these positions that proved to be very challenging for the trained models are presented in Section 5.2.1.

4.5.4 Legal Piece Moves Accuracy

The "Legal Piece Moves Accuracy" metric evaluates the model's ability to correctly generate valid moves for a given piece, based on its position on the board. This metric takes advantage of the fact that the model is trained using a predefined vocabulary, as described in Chapter 4.3.1. Every ply consists of three tokens: {piece}, {start_square}, and {end_square}. Typically, when interacting with the model, it is prompted it to generate the next three tokens from a given starting sequence. However, to assess the model's capability in modelling the state of the game board, the {piece} can be provided, and the model prompted to generate possible tokens for the next token {start_square}. Alternatively, {piece} and {start_square} could be provided, and the model prompted to generate possible tokens for the next token {end_square}.

This process is best illustrated with an example: After the starting sequence "Pg2g4 Pf7f6 Pe2e3 Nb8a6 B", the only valid next token would be "f1" because the bishop on "c1" is blocked and cannot move in this position, as can be seen in Figure 17.

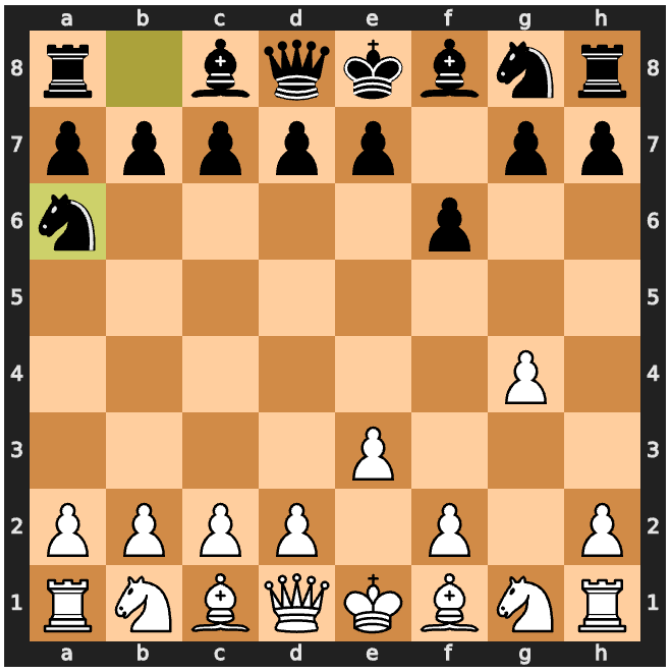

Figure 17: Illustration of Chess Board Position Assessing the Metric "Legal Piece Moves Accuracy"

The manually selected positions for this evaluation cover all pieces and include very specific and challenging situations. Examples of such scenarios include a queen that must block a check, a pinned piece with no legal moves, a king with only one legal move available, and situations involving moving a piece after a pawn has been successfully promoted, among others.

This approach extends the validation beyond merely generating valid chess moves to demonstrating an understanding of all possible legal moves in each situation. To succeed in this metric, the model must demonstrate generating all the n possible tokens in the first n elements of its output when prompted using beam search[27]. For future research, it may be beneficial to modify this metric from a binary (true/false) result to a metric representing the percentage of valid tokens in the model's output, thereby providing a more nuanced evaluation of the model's performance.

[27] Beam search is a heuristic algorithm used in AI to efficiently search through large solution spaces. It works by exploring a limited number of the most promising options at each step, known as the "beam width," and discarding less promising paths. In this project, beam search aids the model in evaluating the most strategically sound moves from a given board position, crucial for assessing metrics like "Legal Piece Moves Accuracy."

5 Results

This chapter presents an analysis of the experimental results obtained from training LLMs on chess move prediction. It delves into the empirical findings from various metrics applied to assess the model's proficiency in generating valid chess moves. The evaluations include the metrics "Average Number of Correct Plies", "Hard Position Accuracy", and "Legal Piece Moves Accuracy" as described in Chapter 4.5. This analysis not only illustrates the models' effectiveness in generating chess moves but also highlights potential areas for refinement and future exploration.

5.1 Results for Metric "Average Number of Correct Plies"

This section presents the results of the "Average Number of Correct Plies" metric[28]. This metric calculates the average number of valid moves generated by the model when playing against itself. Notably, for efficiency, the models were evaluated up to a maximum of 80 plies.

Figure 18 illustrates the average number of correct plies resulting from game sequences generated by different models, expressed as a percentage. Four distinct models are represented, each characterized by a unique combination of dataset size, number of epochs, and learning rate. The training progress is plotted on the x-axis, ranging from 0% to 100%, and the number of plies is shown on the y-axis, ranging from 0 to 40. The progressively deteriorating performance of the red line, representing a model trained on 1 million games, is further analysed in Figure 22.

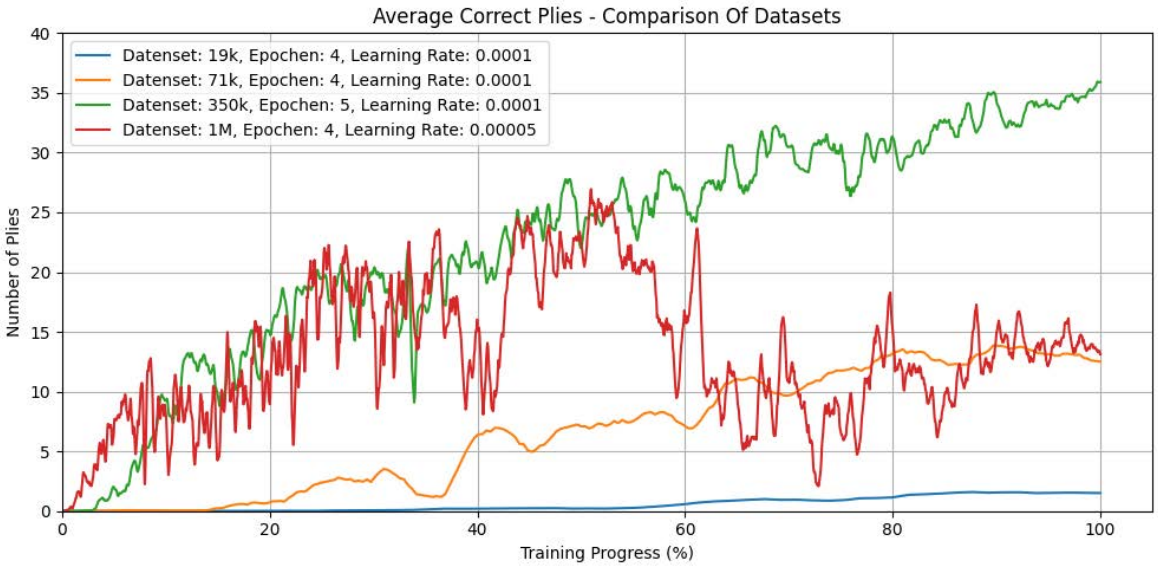

Figure 18: "Average Number of Correct Plies" over Training Progress, Comparison of Dataset Size

Figure 19 compares the average number of correct plies produced by various configurations of training sets and epochs. The bars are categorized by dataset size and epochs on the x-axis, while the y-axis measures the average number of correct plies.

[28] See Section 4.5.2.

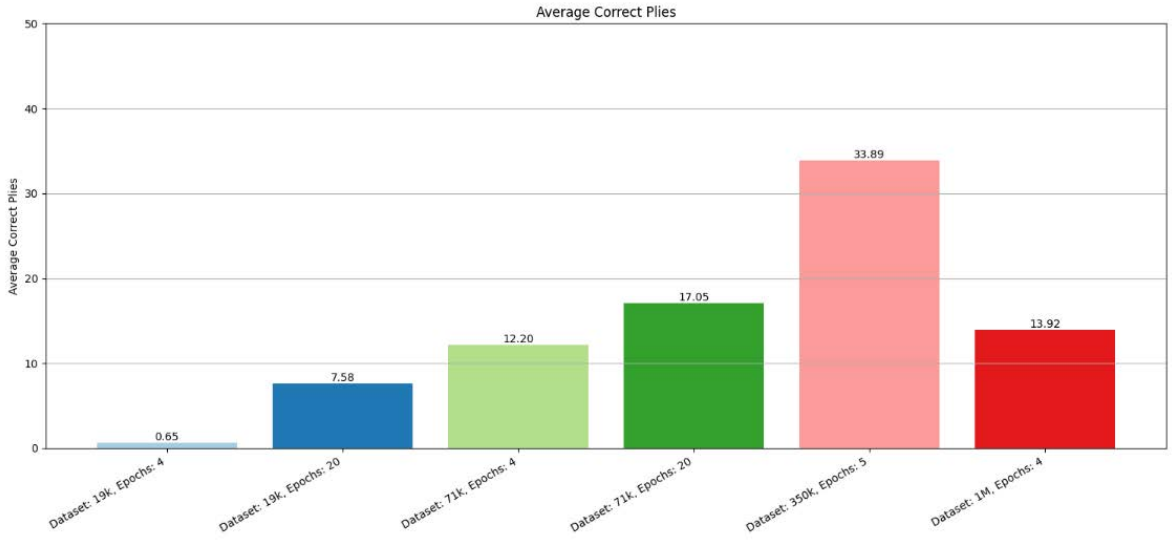

Figure 19: Comparison of "Average Number of Correct Plies" Across Different Model Configurations

Figure 20 and Figure 21 show that increasing the number of epochs from 4 to 20 improves the model's ability to generate valid chess moves. This improvement is reflected in the higher average of correct plies achieved with more epochs, confirming the findings from the Connect 4 experiments[29]. The variability in model's performance at different stages of training is indicated by fluctuations in the orange line, possibly due to the relatively small size of 100 generated games that were used for the validation at each checkpoint. Increasing this number could possibly lead to a smoother curve but would also take longer to compute.

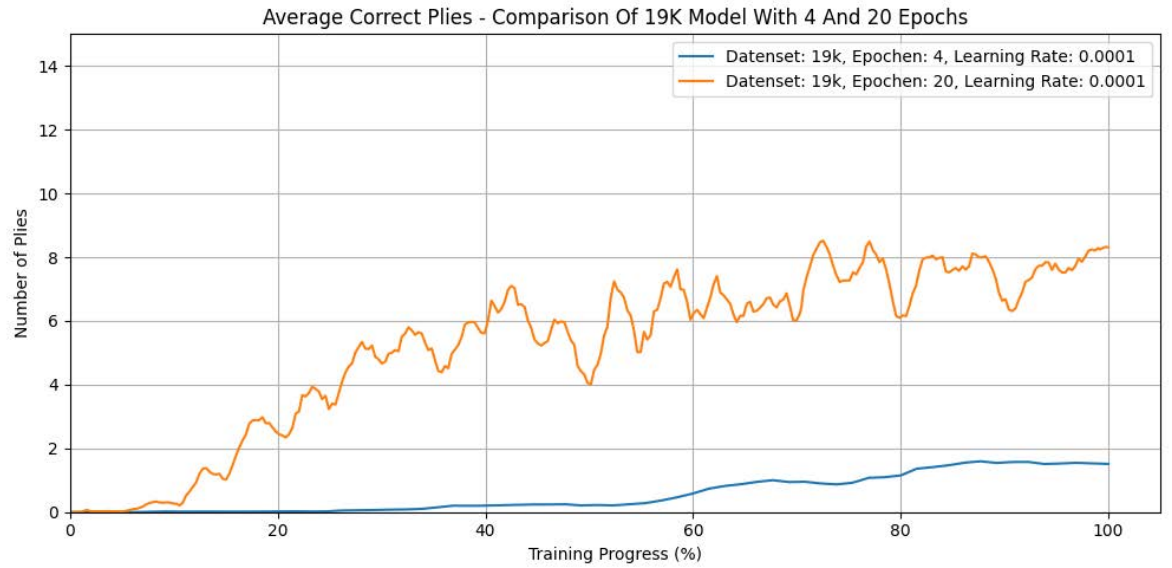

Figure 20: Metric "Average Number of Correct Plies" with Increased Number of Epochs on 19k Dataset

Upon closer examination of Figure 21, it becomes evident that the model trained with 20 epochs begins to underperform compared to the model trained with 10 epochs, particularly after reaching the 60% mark in training progress. This decline in performance is most likely attributable to the omission of the Beginning of Sequence (BOS) token, a topic described in more detail in Chapter 5.4. To summarize, excluding the BOS token for training increases the probability of the model

[29] See Figure 8 and Figure 12 respectively.

generating syntax errors, especially at the start of the generated sequence. Notably, increasing the number of training epochs appears to exacerbate this effect.

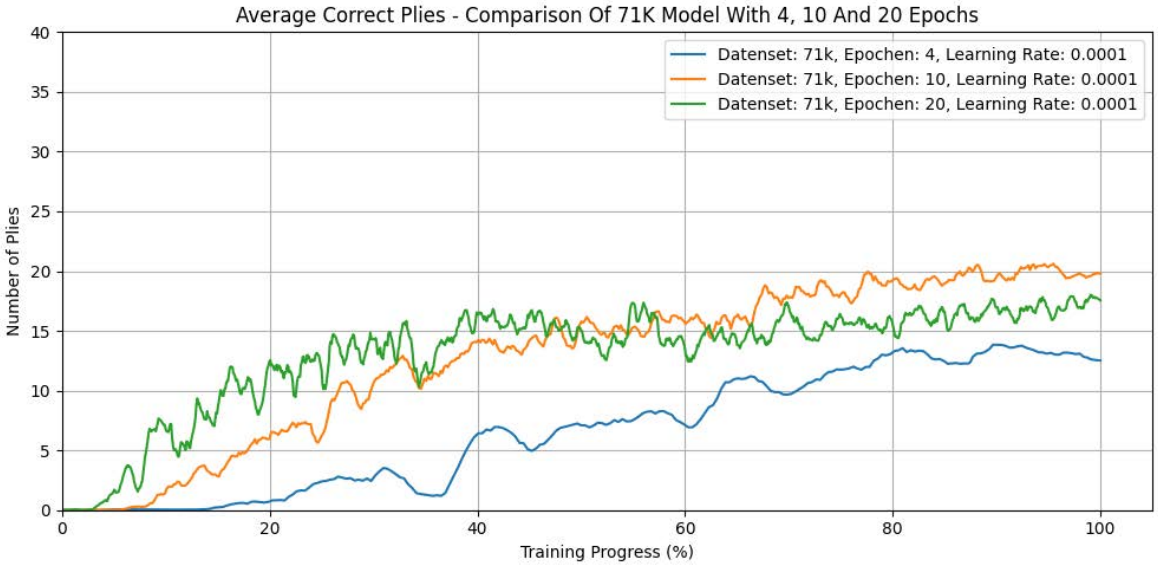

Figure 21: Metric "Average Number of Correct Plies" with Increased Number of Epochs on 71k dataset

In Figure 22, the blue line represents the "Average Number of Correct Plies" result of a model trained with 1 million games. Many of the generated game sequences began with a syntax error[30], leading to a notable reduction in "Average Number of Correct Plies". The orange line reflects the adjusted results, including only sequences without a syntax error in the first move. This issue was later rectified by retraining the model with the inclusion of the BOS token[31].

Figure 22: "Average number of Plies" over Training Progress for the 1M Model, With and Without including Syntax Errors on Move One

[30] See error categories detailed in 4.5.2.

[31] See Chapter 5.4 for more details.

Figure 23 displays the distribution of error categories of game sequences generated by the chess LLM. The interquartile range (IQR) for the error type "Syntax" is very close to zero, indicating that the majority of errors within this category occur at the onset of the prediction sequence with only a handful of outliers. A similar pattern is observed in the "Piece Logic" category, where the median is approximately at 5 plies. This can be interpreted as follows: If the model does not generate a syntax error or a piece logic error respectively in the first moves of the game sequence, it is very unlikely that these types of errors will occur in the subsequent generated sequence. The categories "Path Obstruction" and "Pseudolegal" show a uniform distribution across the range from 0 to 80 plies without a particular trend. For the "No Error" category, the median is at 80 plies, which represents the maximum sequence length achievable. There are also several instances below 80 plies, which occur if the end of the game is reached before the 80th ply.

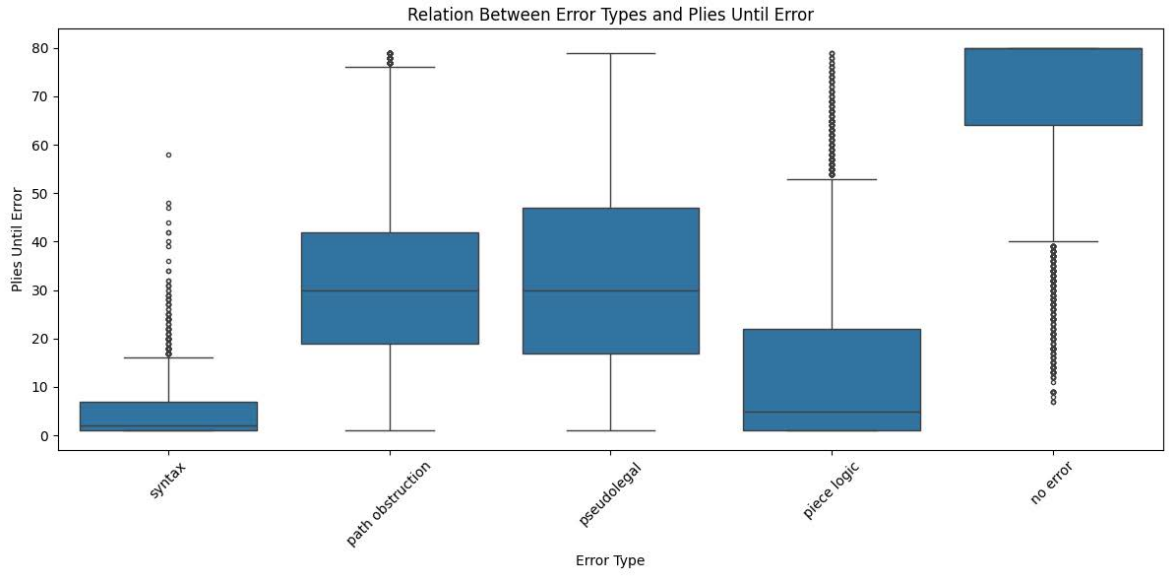

Figure 23: Boxplot Analysis of First Incorrect Move Categories

5.2 Results for Metric "Hard Position Accuracy"

This section presents the results of the "Hard Position Accuracy" metric, as detailed in Section 4.5.3.

Figure 24 shows the progressive results for the "Hard Position Accuracy" metric when training the model on different datasets. The x-axis shows the progress of the training as a percentage, while the y-axis indicates the accuracy for this metric, also as a percentage.

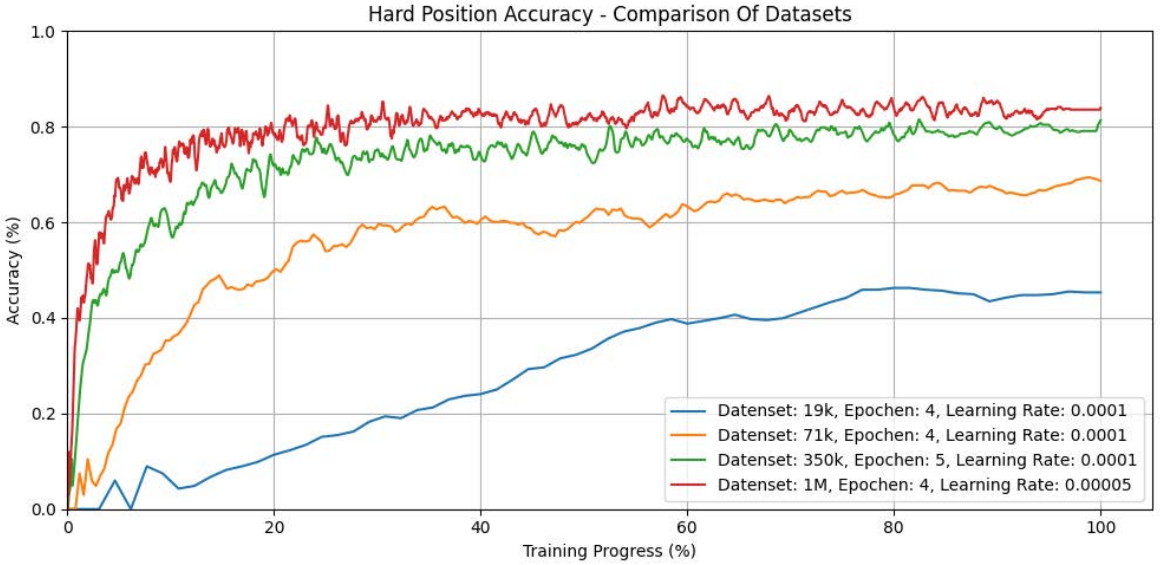

Figure 24: "Hard Position Accuracy" over Training Progress, Comparison of Dataset Size

Figure 25 compares the results for the "Hard Position Accuracy" metric of models trained using different datasets and configurations. Each bar in the chart represents a specific model configuration, with its height indicating the accuracy for this metric.

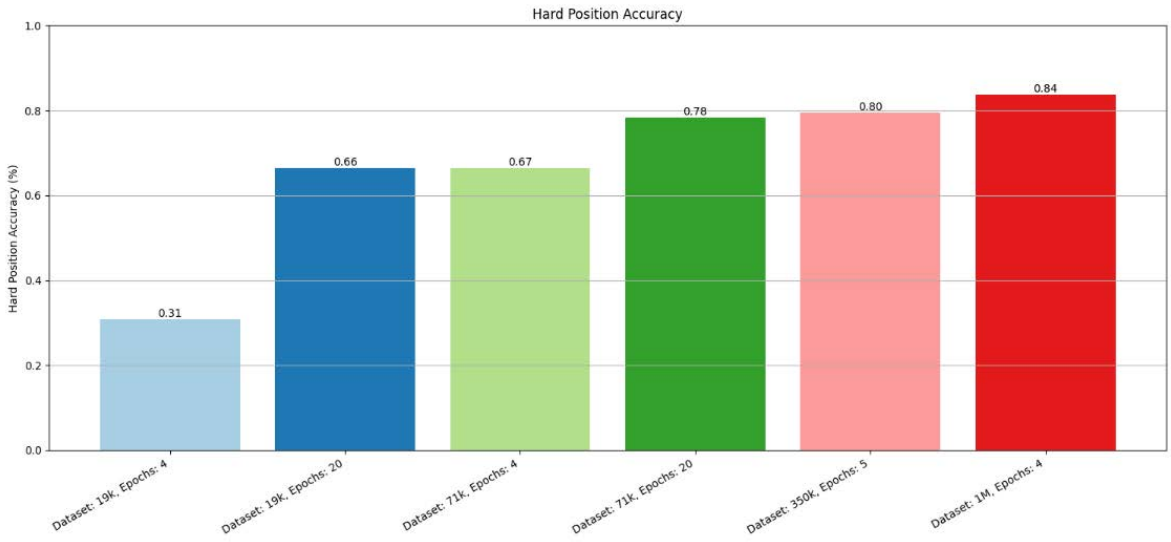

Figure 25: Comparison of "Hard Position Accuracy" Across Different Model Configurations

Figure 24 and Figure 25 reveal a clear trend: more data generally leads to better results in terms of "Hard Position Accuracy".

The pattern observed in Figure 26 and Figure 27 mirrors the one observed in Figure 20 and Figure 21 respectively, showing that the model's ability to solve hard positions significantly improves when trained for a higher number of epochs on the same dataset.

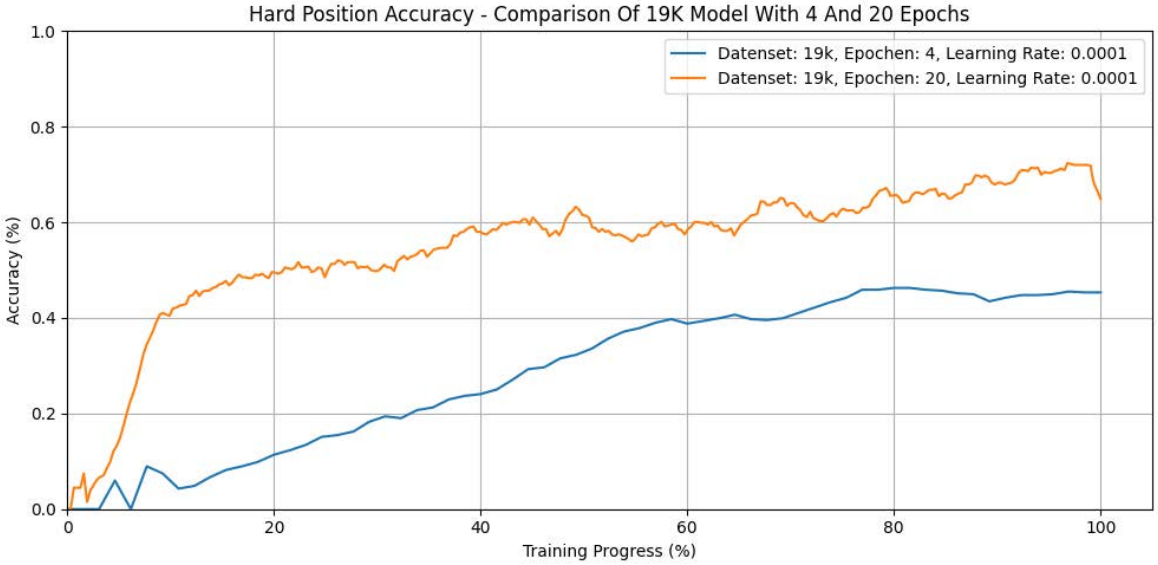

Figure 26: Comparing "Hard Position Accuracy" Across Different Epochs on 19k Dataset

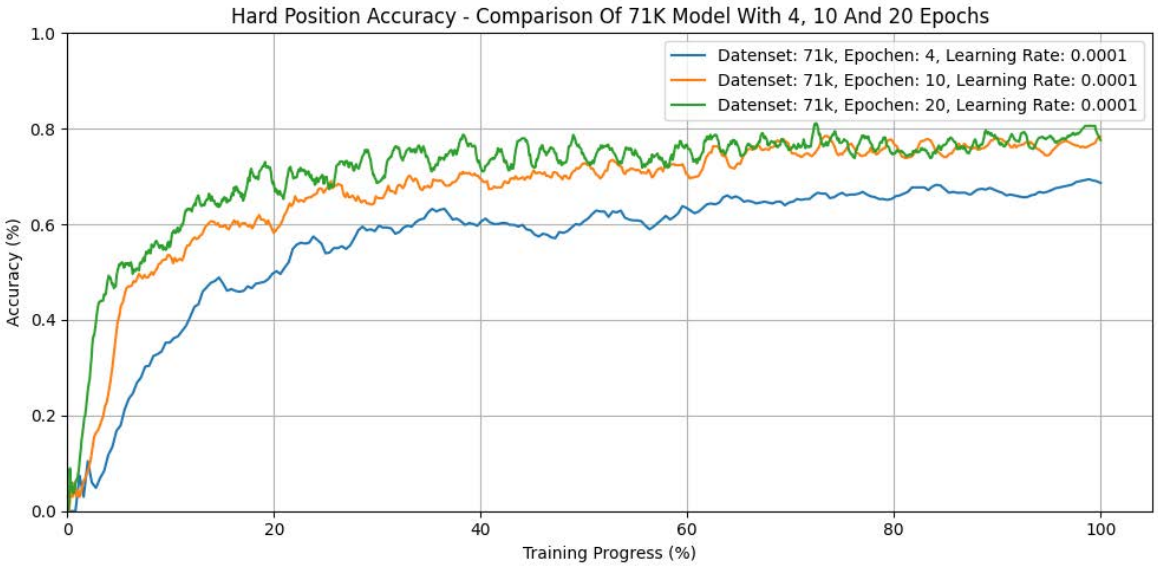

Figure 27: Comparing "Hard Position Accuracy" Across Different Epochs on 71k Dataset

Figure 28 examines the model's performance on the "Hard Position Accuracy" metric in more detail. The x-axis displays the IDs of the positions, while the height of each bar indicates the percentage of correct predictions for each specific position. We can see that the model mastered 11 out of the 67 positions, which translates to a success rate of 84%. An analysis of a selection of positions where the model still struggles to generate valid moves is presented in the next section.

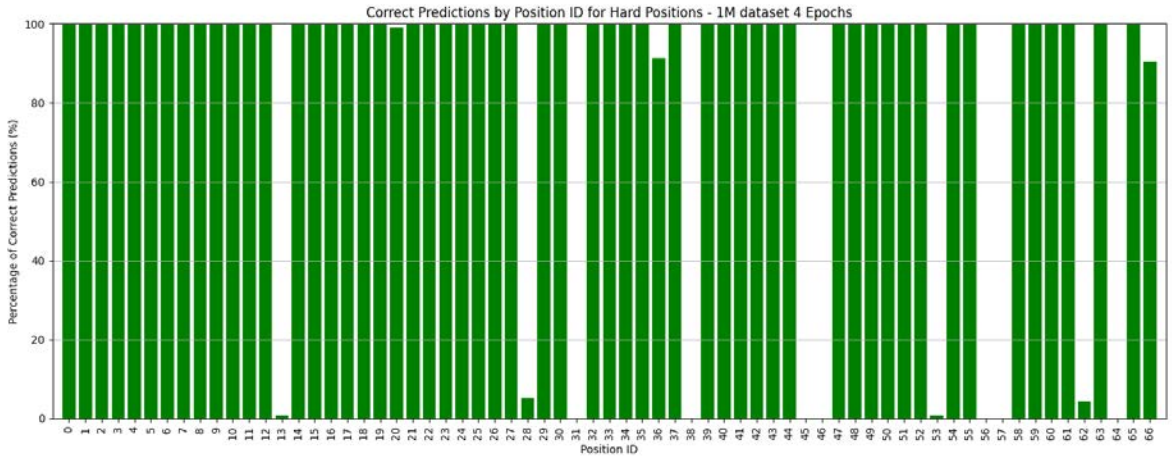

Figure 28: Detailed Analysis of Metric "Hard Position Accuracy" by Position

5.2.1 Challenging Positions for "Hard Position Accuracy"

This section showcases some of the challenging positions where the model, trained on 1 million games, still encounters errors, often failing to generate valid moves.

In the position in Figure 29, Black is forced to move its king to either "d8" or "e7" to evade the attack of White's queen. The move suggested most often by the model is bishop from "f5" to "d7", which is invalid because the bishop's path is obstructed by the pawn on "e6".

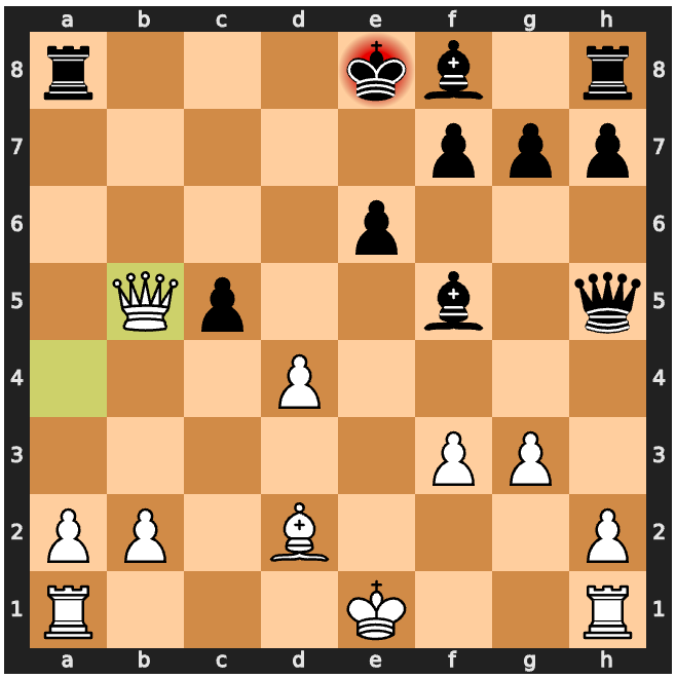

Figure 29: Hard Position Accuracy #28 – King is Forced to Move

Figure 30 shows a position, where White's king is put into check by Black's rook. In this situation, White must block the attack with one of its pieces, as its king cannot move. The pieces available for blocking are either one of its bishops or the queen. However, the model's generated output frequently attempts to move White's king to either "b1" or "d1", erroneously ignoring its own pieces blocking those squares. This scenario is solved correctly in 0% of the cases.

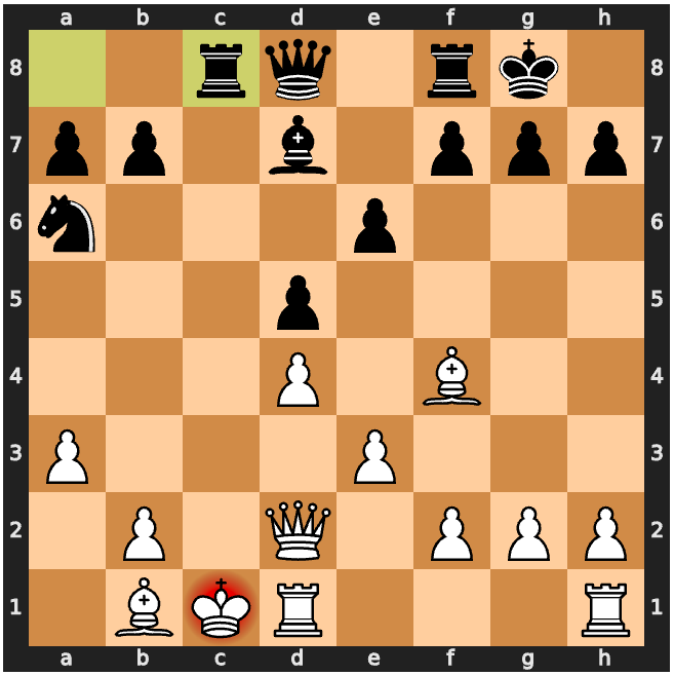

Figure 30: Hard Position Accuracy #31 – Piece Needs to Block an Attack

Figure 31 showcases a position where White just captured Black's bishop on "d5", resulting in a discovered check by White's queen. Black is forced to either move its king to a safe square on "g8" or "h8", or to block the attack by moving its knight to "e4" or its pawn from "e5" to "e4". The incorrectly generated moves by the model include moving the rook on the a-file to "b8" or "c8" or capturing the pawn on "d5" with the knight.

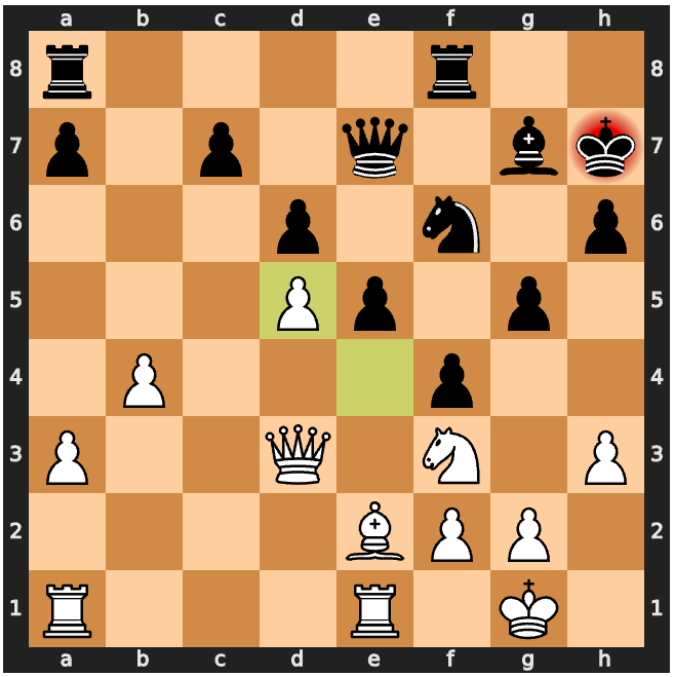

Figure 31: Hard Position Accuracy #38 – Discovered Check

Figure 32 shows an especially tricky situation: White's only legal move is to capture Black's f-pawn en passant with its e-pawn. In this scenario, the model consistently fails to identify this valid move. The model's suggested moves in this position are to move the White's king to one of the surrounding squares or to move the pawn on "f2" to "e3".

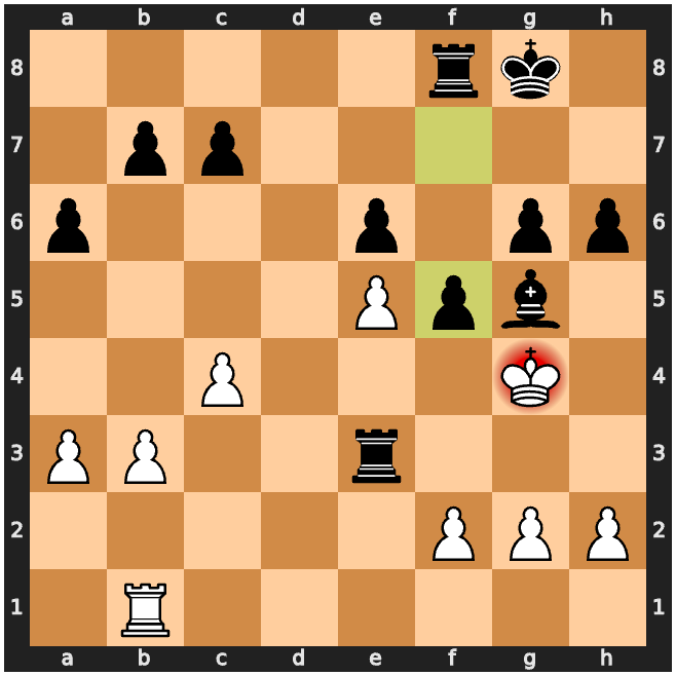

Figure 32: Hard Position Accuracy #56 – Forced En Passant

In the position shown in Figure 33, Black can legally move only its bishops. However, the model, in 98% of cases, incorrectly attempts to move the king to "h7", "g7", or "g8". These moves are illegal as they would result in Black's king being placed in check. The error likely stems from the two bishops not being moved for an extended series of moves, potentially leading the model to "forget" their presence on the board.

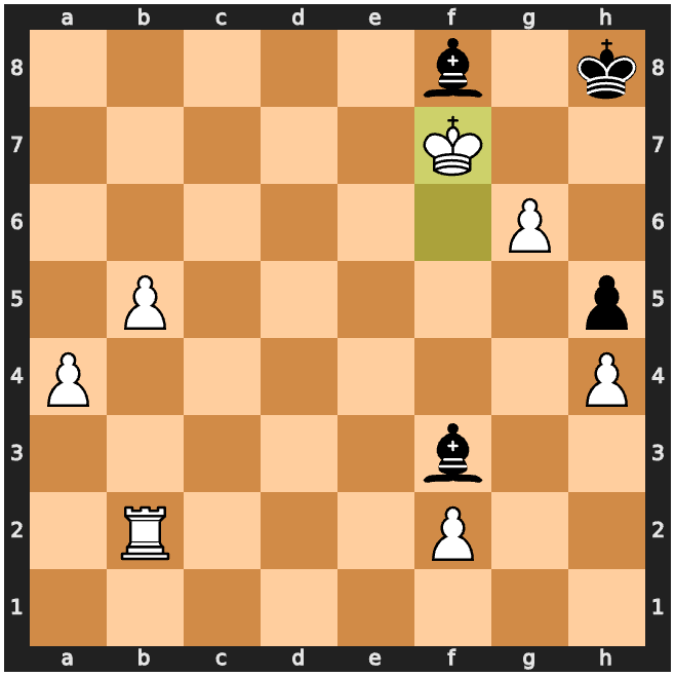

Figure 33: Hard Position Accuracy #64 – Pieces Haven't Been Moved for a Long Time

5.3 Results for Metric "Legal Piece Moves Accuracy"

This section presents the results of the "Legal Piece Moves Accuracy" metric, as detailed in Section 4.5.4.

Figure 34 shows the progressive results for the "Legal Piece Moves Accuracy" metric for different versions of the model. The x-axis shows the training progress as a percentage, while the y-axis indicates the accuracy of this metric, also as a percentage.

Figure 34: "Legal Piece Moves Accuracy" over Training Progress, Comparison of Dataset Size

Figure 35 compares the results for the "Legal Piece Moves Accuracy" metric across different model configurations. Each bar in the chart represents a model trained using a specific dataset, with its height indicating the achieved accuracy.

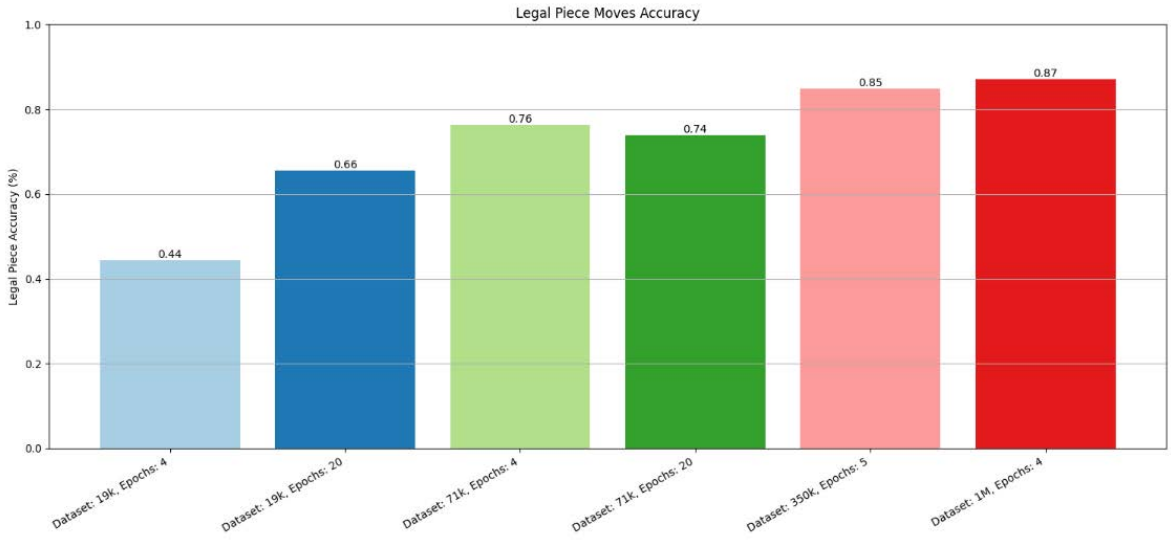

Figure 35: Impact of Training Configurations on Metric "Legal Piece Moves Accuracy"

Figure 36 and Figure 37 confirm the pattern observed for previously discussed metrics: The model's performance in the "Legal Piece Moves Accuracy" metric improves significantly when trained for a higher number of epochs while using the same dataset.

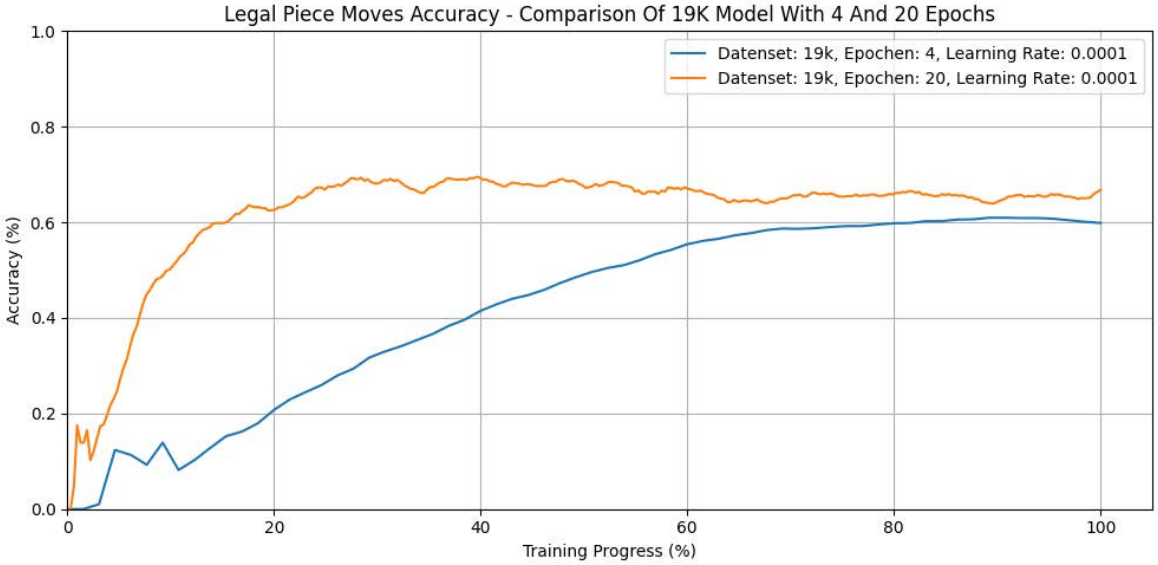

Figure 36: Comparing "Legal Piece Moves Accuracy" Across Different Epochs on 19k Dataset

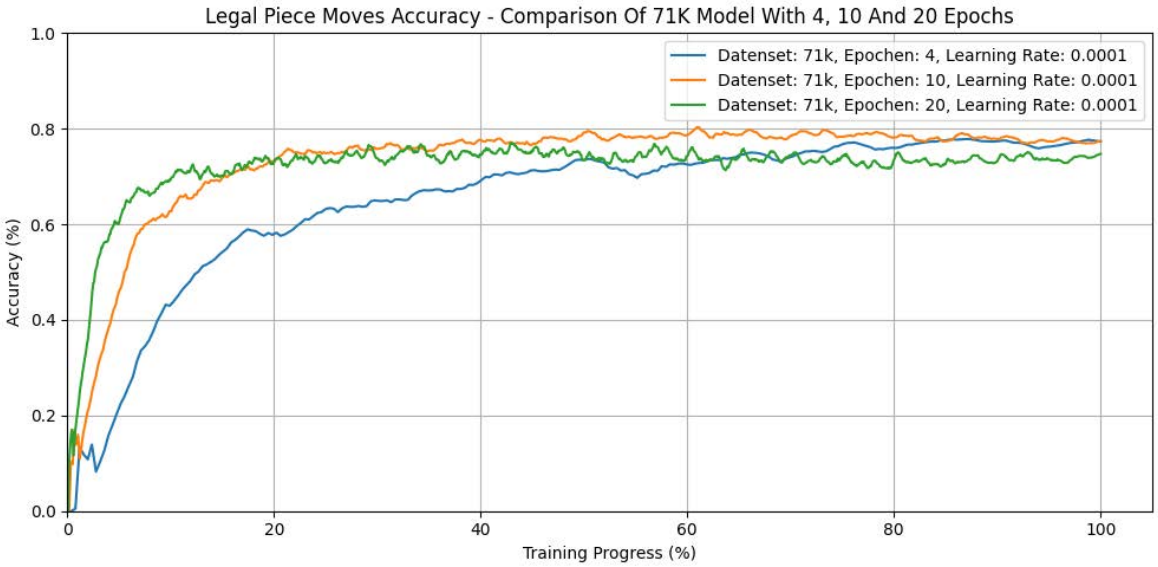

Figure 37: Comparing "Legal Piece Moves Accuracy" Across Different Epochs on 71k Dataset

The following figures examine the model's performance on the "Legal Piece Moves Accuracy" metric in more detail.

Figure 38 shows the results of the model trained on 1M games for positions where it is given a sequence leading to a specific start piece and is prompted to generate the next token indicating possible positions for this piece. This model performs well in these positions, closely related to its ability to generate valid moves.

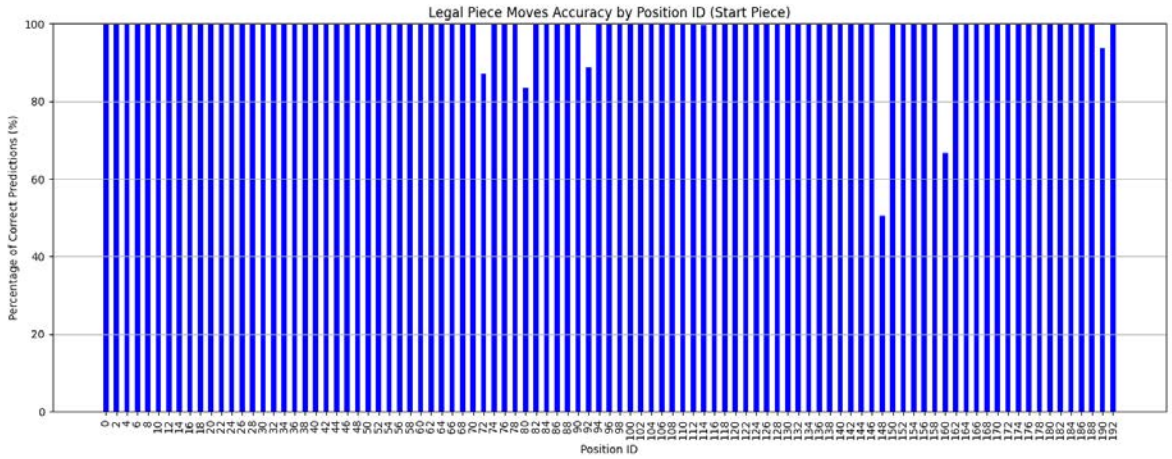

Figure 38: Detailed Analysis of "Legal Piece Moves Accuracy" with Given Start Piece

Figure 39 shows positions where the model is given a sequence that includes a piece and its start square and is prompted to generate the next token indicating possible target squares for the piece's move. The x-axis displays the IDs of the positions, while the height of each bar indicates the percentage of correct predictions for each position. This percentage is calculated based on the proportion of valid tokens within the model's most likely predictions.

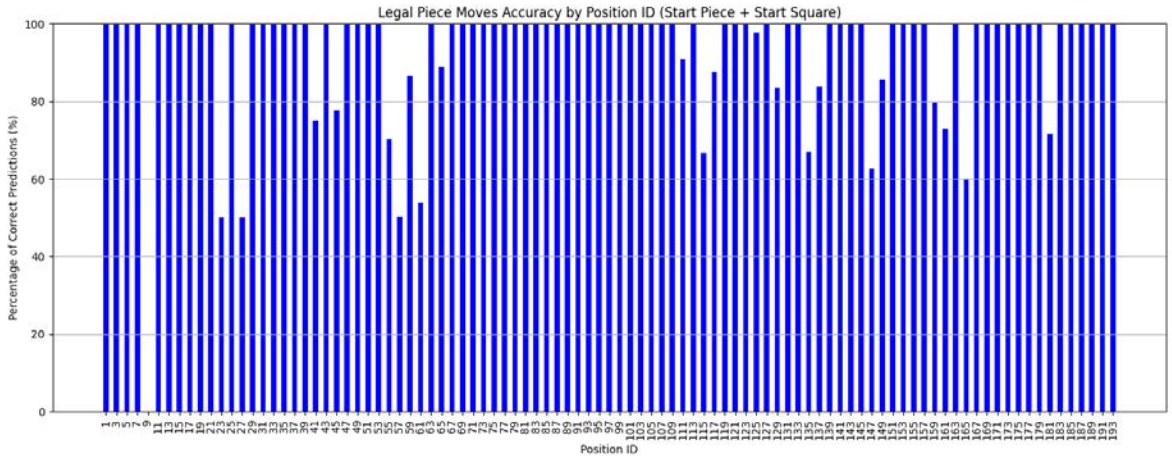

Figure 39: Detailed Analysis of "Legal Piece Moves Accuracy" with Given Start Piece and Start Square

5.3.1 Challenging Positions for "Legal Piece Moves Accuracy"

This section presents a selection of positions from the "Legal Piece Moves Accuracy" dataset where the trained models achieved the accuracy scores lower than 60%, as depicted in Figure 38 and Figure 39.

In the position shown in Figure 40, the model is prompted to generate valid target squares for Black's king on "e8". The correct answer is "d8", as moving to "e7" or "f8" would put the king into check. The model typically suggests "g8" or "c8", incorrectly attempting to castle. "d8" is usually the third most likely move to be generated in this position, indicating challenges the model faces with the rules of castling.

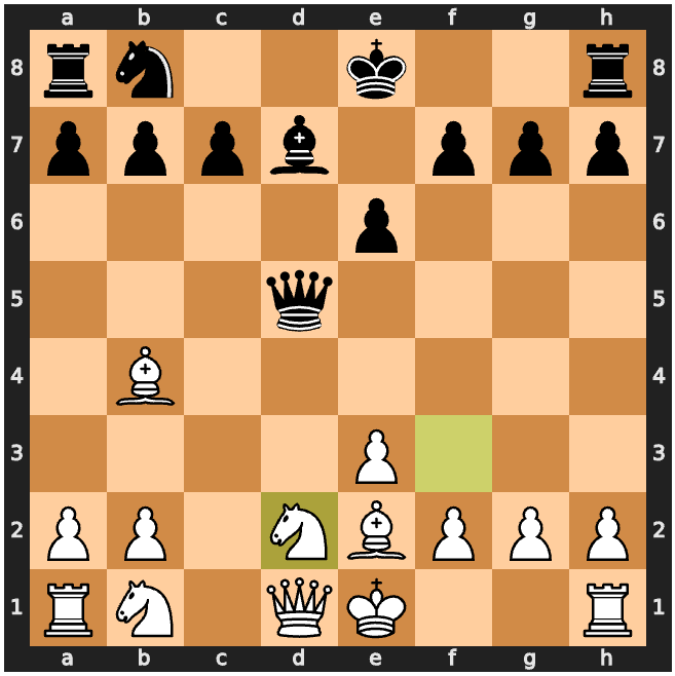

Figure 40: Legal Piece Moves Accuracy #9 – Black King's Position

Figure 41 features a king endgame scenario, where the model is prompted to suggest valid moves for White's king on "e2". Correct answers include "d1", d2", "e1", and "f1". However, the model often incorrectly predicts "d3" or f3".

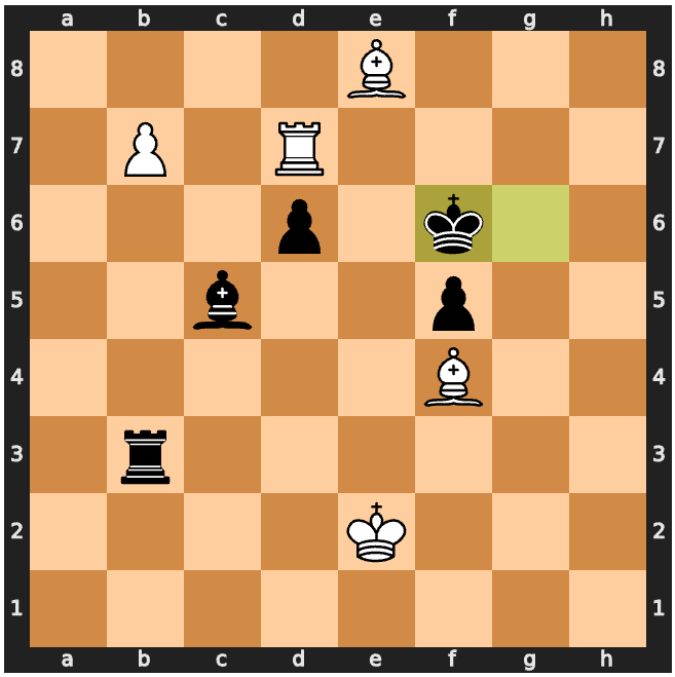

Figure 41: Legal Piece Moves Accuracy #23 – King Endgame

In Figure 42, an endgame position is presented where Black's Queen just moved to "b6", putting White's king in check. The model's task is to identify legal moves for White's king. Correct responses are "a4" or capturing Black's queen at "b6". Common incorrect answers from the model include "b4" and "c4".

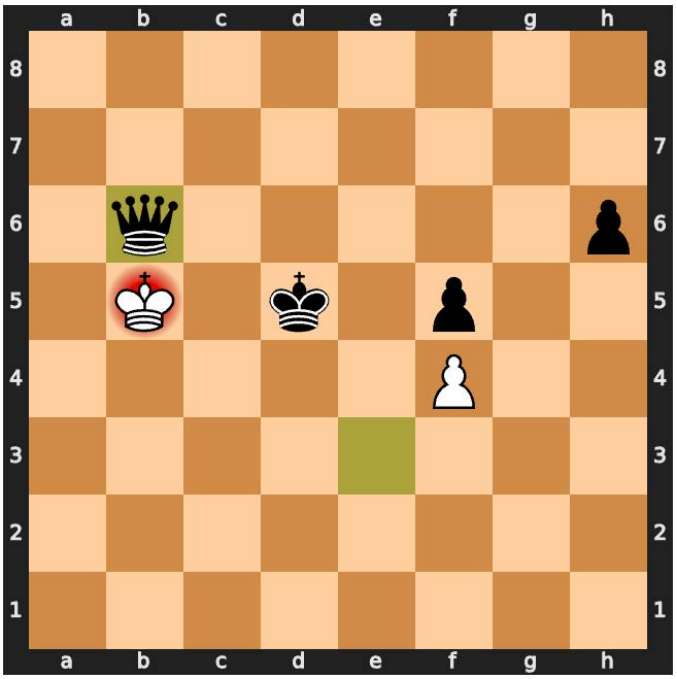

Figure 42: Legal Piece Moves Accuracy #27 – Endgame Position

The position in Figure 43 asks the model to identify valid moves for White's newly promoted queen on "b8". Valid moves include "a7", "b7", "c7", "a8", "c8", "d8", "b6", and "d6". The model often incorrectly suggests "h8" or "f8", overlooking White's bishop on "e8".

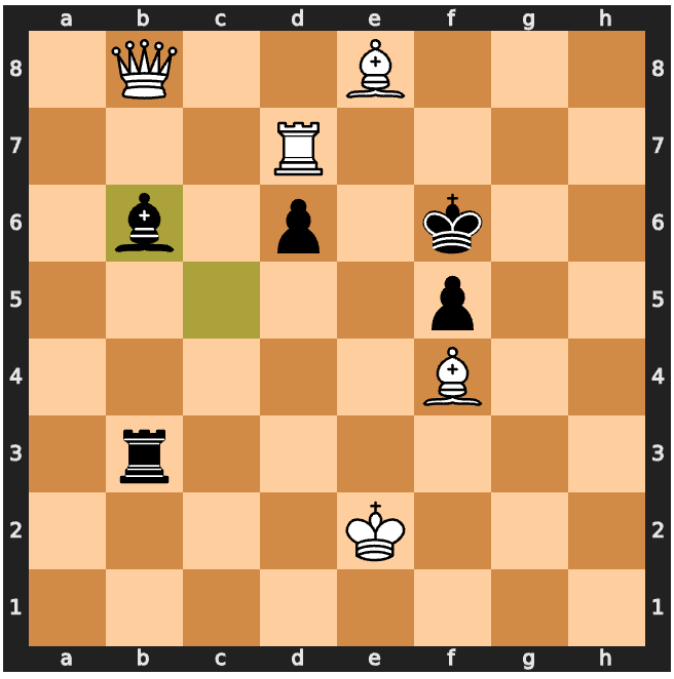

Figure 43: Legal Piece Moves Accuracy #57 – Newly Promoted Queen

In Figure 44, White's queen must block Black's attack by moving to "d3" or "e2". The model frequently predicts "a4" and "b3", which are incorrect, as White's king remains in check.

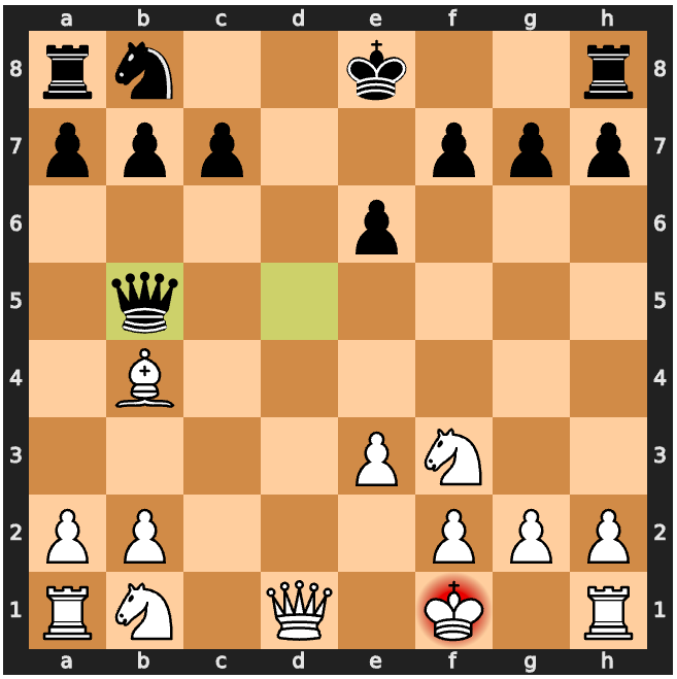

Figure 44: Legal Piece Moves Accuracy #61 – Blocking Black's Attack

In the previously discussed positions, the model generated potential target squares for a given piece and starting square. In the following two positions, the model is given a piece and must identify the corresponding starting square(s).

Figure 45 illustrates a position where the model is prompted to find legal positions for the rook. It correctly identifies "d7" as a starting square, but fails to consider the move "Rb7b8", indicating the promotion of the b-pawn to a rook, a valid but less likely move.

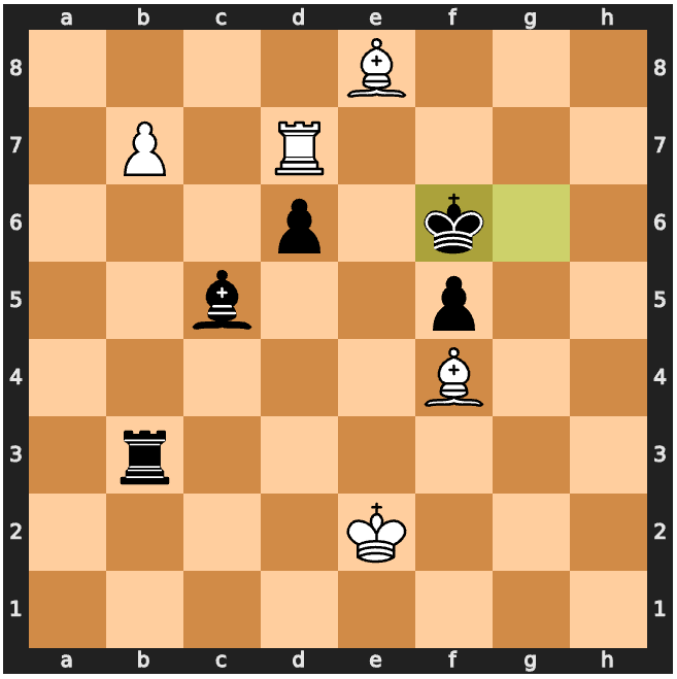

Figure 45: Legal Piece Moves Accuracy #148 – Identifying Rook's Position

In Figure 46, the model is asked to generate valid starting squares for a rook. In this scenario, White can only legally move its rook on the h-file. The model often incorrectly predicts "e1".

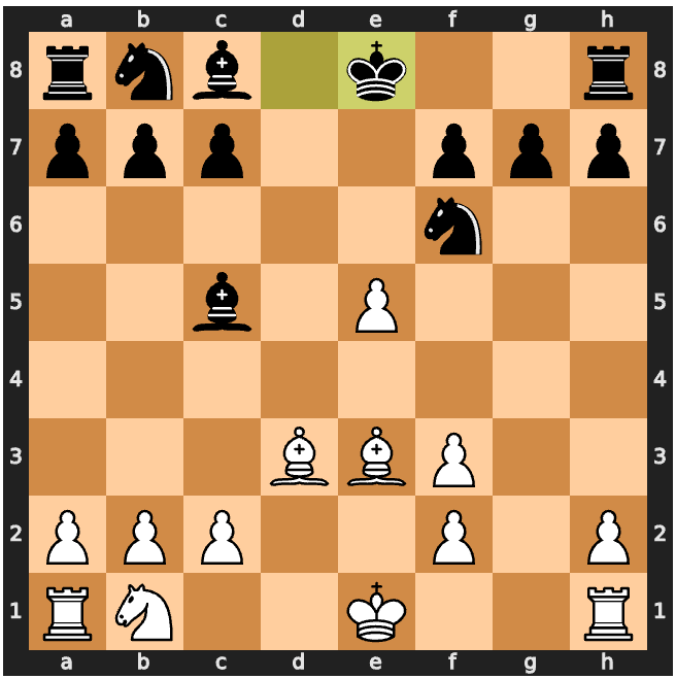

Figure 46: Legal Piece Moves Accuracy #162 – Rook's Starting Squares

5.4 Comparison of Models using BOS Token

The following chapter examines how the integration of the Beginning of Sequence (BOS) token influences the performance of the trained models.

5.4.1 "Average Number of Correct Plies" using BOS Token

The models incorporating the Beginning of Sequence (BOS) Token demonstrate a significantly higher value for the "Average Number of Correct Plies" metric, as evident in Figure 47. It is observed that the larger the training dataset and the longer the duration of training, the greater the improvement in performance.

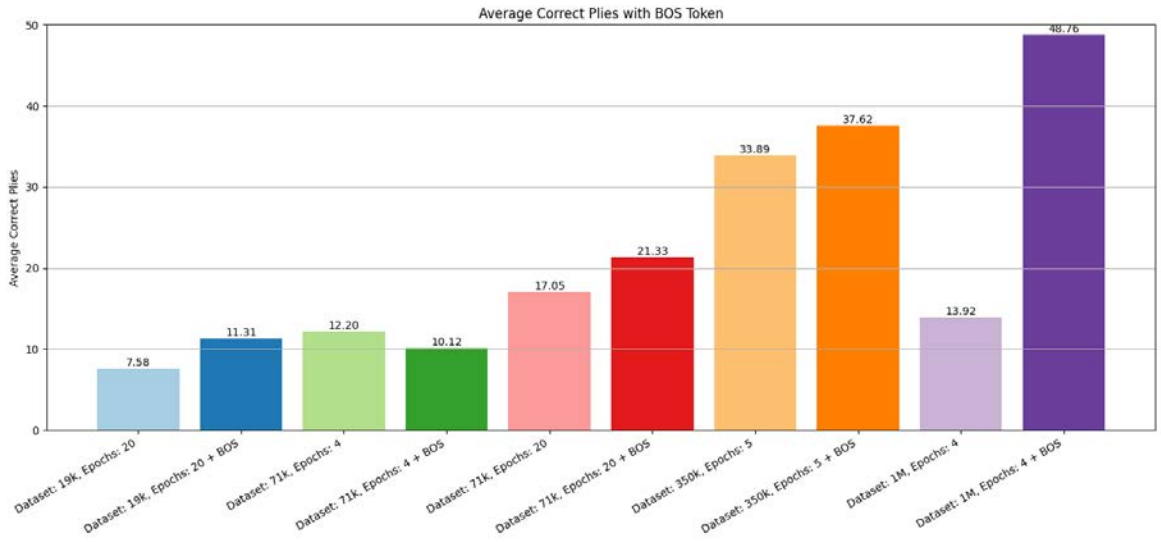

Figure 47: Comparison of "Average Number of Correct Plies" between Models With and Without BOS Token

Table 6 compares the error categories of models with and without the BOS token. A notable difference is seen in the substantial decrease in syntax errors and an increase in games without any errors when using the BOS token.

Table 6: Comparison of Error Types With and Without BOS Token

Model	Error Type	With BOS (%)	Without BOS (%)
1M 4 Epochs	Syntax	6	71
	Path Obstruction	27	15
	Pseudolegal	45	11
	Piece Logic	2	1
	No Error	20	2
350k 5 Epochs	Syntax	10	13
	Path Obstruction	44	44
	Pseudolegal	37	37
	Piece Logic	7	1
	No Error	2	5

71k 20 Epochs	Syntax	2	21
	Path Obstruction	55	49
	Pseudolegal	32	19
	Piece Logic	11	11
	No Error	0	0
71k 4 Epochs	Syntax	2	8
	Path Obstruction	60	43
	Pseudolegal	24	33
	Piece Logic	12	16
	No Error	0	0
19k 20 Epochs	Syntax	19	19
	Path Obstruction	37	35
	Pseudolegal	27	28
	Piece Logic	16	18
	No Error	1	0

Figure 48 provides a deeper examination of the error categories in the 1M model trained with the BOS token. The training process, as depicted in the figure, shows a mostly consistent decrease in both syntax and piece logic errors. The model begins to generate error-free games around 58,000 steps, and the frequency of such games steadily increases until the end of training.

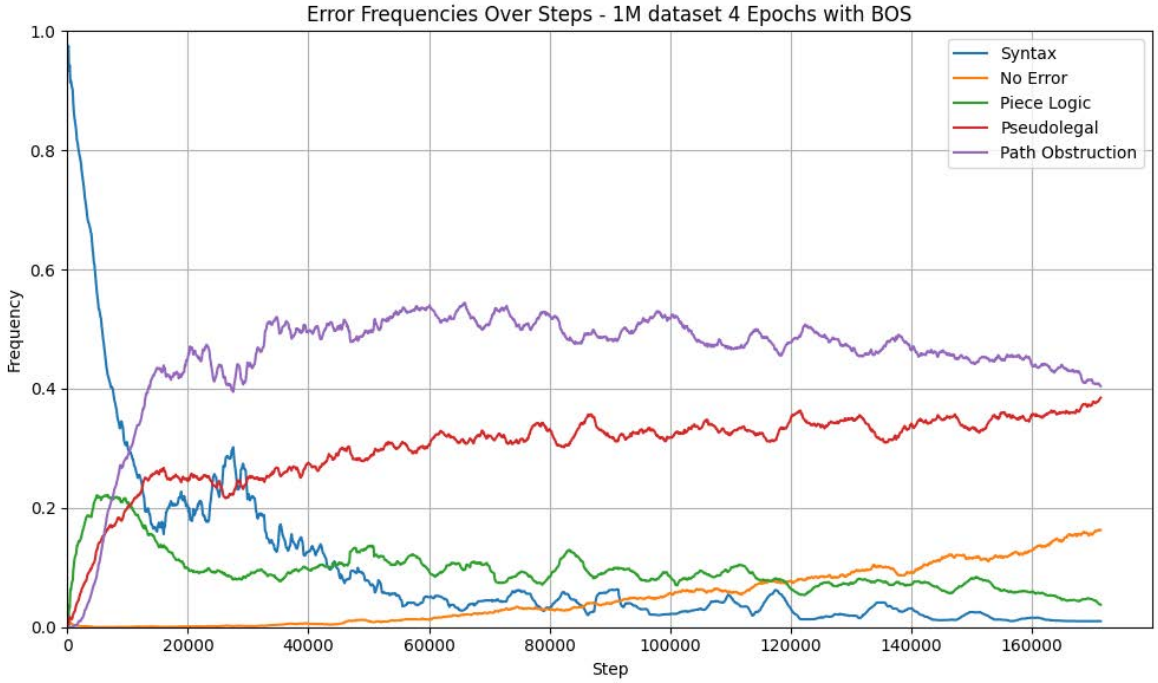

Figure 48: Frequency of Error Types During Training of 1M Dataset with BOS

Figure 49 extends the analysis from Figure 22, including a plot that measures the average number of correct plies over the training progression of the 1M model trained with the BOS token. It follows a similar trend as the model trained without the BOS token until approximately 90k steps, where the model without the BOS token starts to produce a significant number of syntax errors. Towards the end of training, the model achieves up to 50 correct plies on average. The consistent improvement in average correct plies suggests that the results would continue to increase with extended training of the model.

Figure 49: "Average Number of Correct Plies" During Training Progress for the 1M Model (With and Without BOS Token)

5.4.2 "Hard Position Accuracy" using BOS Token

When comparing the "Hard Position Accuracy" results with and without the BOS token, as depicted in Figure 50, there is only a minor difference noted, except for the 19k model. For the 19k model

trained over 20 epochs, there is a 7% decrease in accuracy, and for the 71k model with 20 epochs, a very slight decrease is observed. In contrast, for models trained with 4 or 5 epochs, a slight improvement of 2 to 3 percent is recorded.

This deterioration or only marginal improvement in models trained with more epochs may indicate that these models are potentially overfitted. Overfitting occurs when a model learns the training data too well, including its noise and outliers, which results in poor performance on new, unseen data.

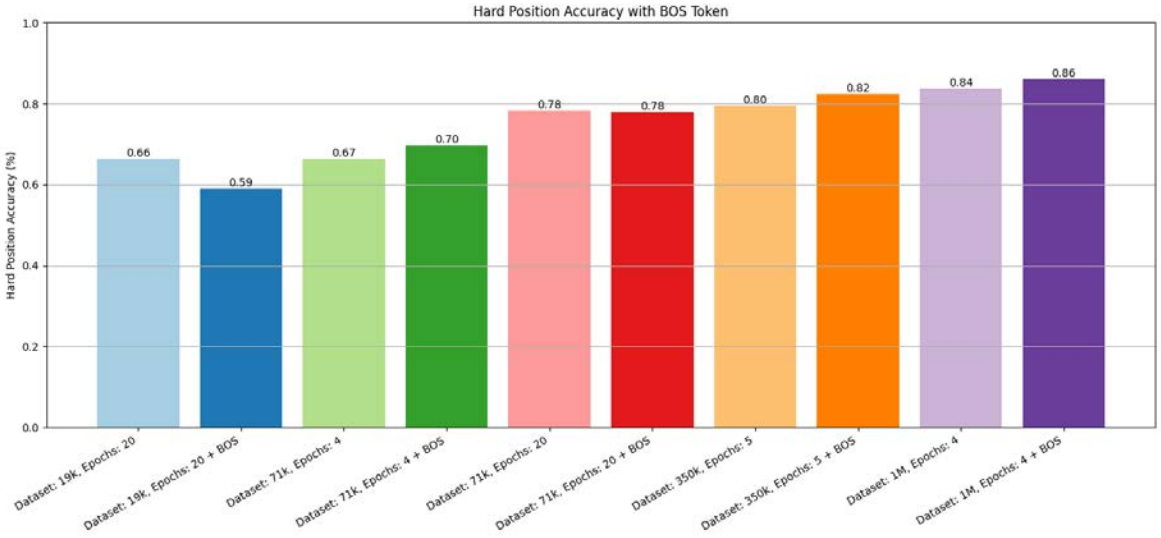

Figure 50: Comparison of "Hard Position Accuracy" Across Models With and Without BOS

5.4.3 "Legal Piece Moves Accuracy" using BOS Token

In Figure 51, when comparing "Legal Piece Moves Accuracy" values, a decrease of 1-2% is observed in most models. Exceptions are the 71K 20 epochs model and the 1M model, which show a slight improvement. These results indicate that the BOS token does not have a significant impact on this metric and can the models track the board state with similar effectiveness.

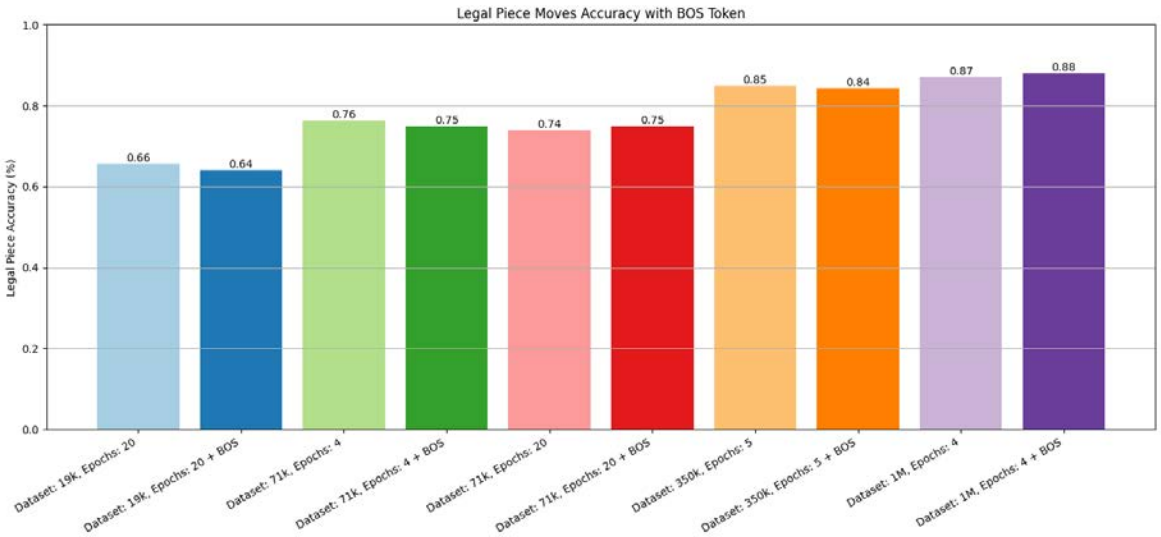

Figure 51: Comparison of "Legal Piece Moves Accuracy" Across Models With and Without BOS

5.5 Challenges and Limitations

In developing and evaluating the models trained for chess move generation, several key challenges were encountered.

5.5.1 Training with Expanded Moves Dataset

An experiment involved training models with an expanded dataset, as described in Section 4.2.3. In Figure 52, the trajectory of the "Average Number of Correct Plies" metric initially appears promising, with peak values exceeding 75. The model generates a sequence of 80 plies without error in 90% of the cases. However, a closer examination of these plies reveals that all these error-free sequences of 80 plies are identical to those in the training dataset. Although the model fulfils the requirement of generating as many legal moves as possible, it compromises model flexibility, as shown in the two additional evaluation metrics.

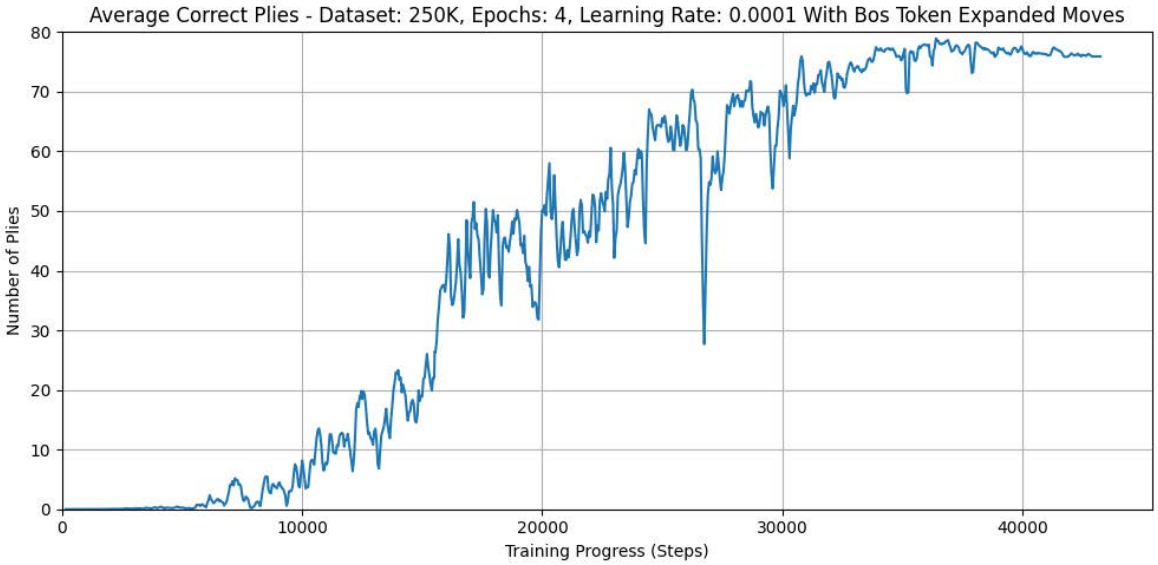

Figure 52: "Average Number of Correct Plies" over Training Progress with Expanded Moves Dataset

The results from the "Hard Position Accuracy" metric, shown in Figure 53, and the "Legal Piece Moves Accuracy", displayed in Figure 54, indicate that the model struggles with a wide spectrum of positions. In only 20-35% of cases, the model predicts a correct move in hard positions, and in less than 30% of cases, it correctly predicts the start or target squares in the "Legal Piece Moves Accuracy" positions.

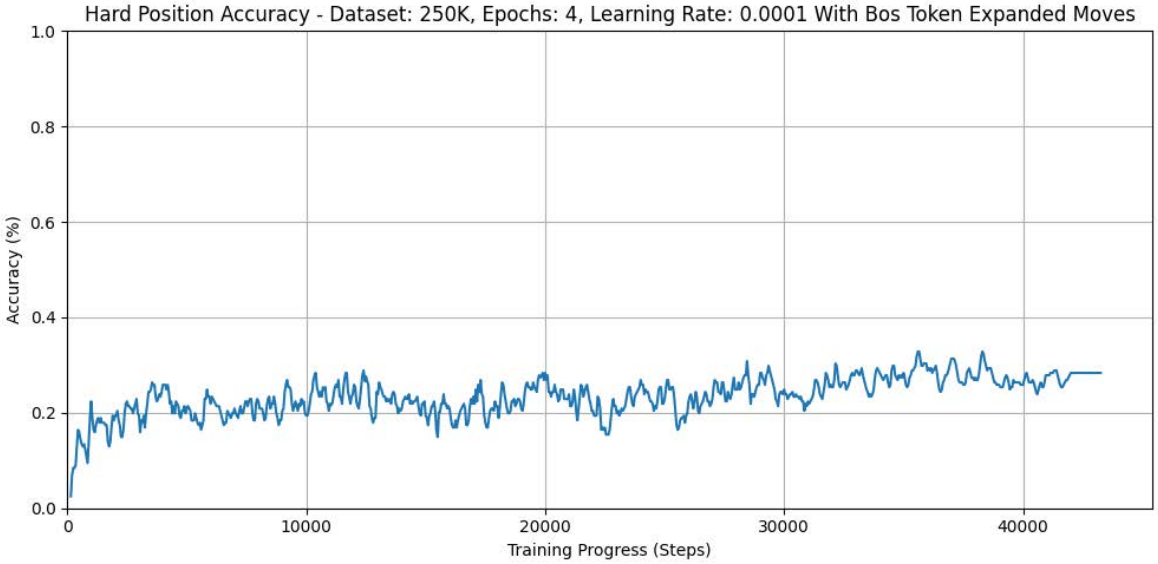

Figure 53: "Hard Position Accuracy" over Training Progress with Expanded Moves Dataset

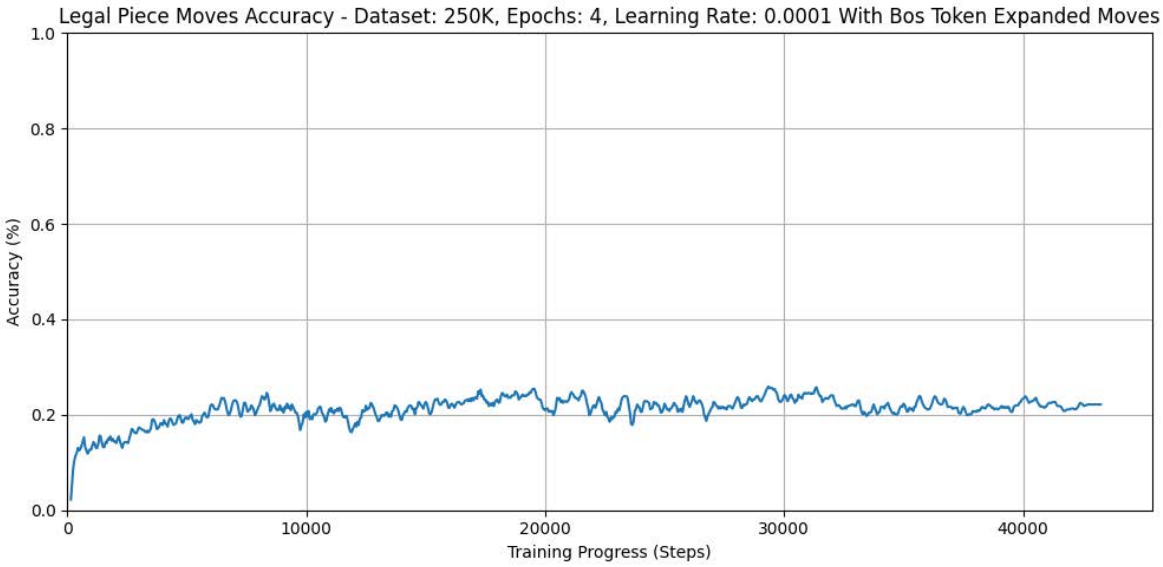

Figure 54: "Legal Piece Moves Accuracy" over Training Progress with Expanded Moves Dataset

The model trained with the "Expanded Moves" dataset struggles to learn the game's rules, often making illegal moves in unfamiliar positions and attempting implausible moves, such as moving the king two squares diagonally or the queen two squares forward and one square to the left. These poor results are attributed to the dataset's limited variance.

5.5.2 Limitations of the "Average Number of Correct Plies" Metric

The "Average Number of Correct Plies" metric showed limitations when a model generated a complete game sequence of less than 80 plies without errors. Such rare outcomes could potentially underrate the model's performance, as a shorter, but perfectly generated game would decrease the metric's value.

5.5.3 Limitations of the "Legal Piece Moves Accuracy" Metric

Another notable limitation was found in the "Legal Piece Moves Accuracy" metric, initially evaluated in a binary pass/fail manner for each position. This rigid assessment made it difficult to

fully assess the model's understanding of chess moves, especially in positions with multiple legal options. The strict validation process, which required the model to generate all valid moves, led to frequent failures, even when most of the model's output was correct.

5.5.4 Challenges Encountered with the BOS Token

Additionally, implementing the "Beginning of Sequence" (BOS) token presented challenges. Its absence led to frequent syntax errors at the start of generated sequences, impacting the overall performance metrics. Introducing the BOS token later improved accuracy but also highlighted the sensitivity of LLMs to changes in input structure and tokenization.

5.6 Discussion of Results

To summarize, the models trained on chess notation have been assessed using several metrics, including "Average Number of Correct Plies", "Hard Position Accuracy", and "Legal Piece Moves Accuracy". The models trained with the largest dataset learned to accurately generate up to 50 plies when simulating games from scratch. It is plausible that this metric would continue to improve with an increased volume of data.

Regarding the "Hard Position Accuracy" metric, the model trained with 1M games achieved a remarkable 89% accuracy. A detailed analysis of the results for both "Hard Position Accuracy" and "Legal Piece Moves Accuracy" revealed areas where errors persisted, such as in the model's understanding of certain chess rules and specific game scenarios. Generally, across all metrics, a higher volume of data correlated with better performance. The inclusion of the Beginning of Sequence (BOS) token was identified as a crucial factor in enhancing model accuracy, particularly in reducing syntax errors.

Reflecting on potential improvements, a more nuanced approach to evaluating the "Legal Piece Moves Accuracy" metric, such as using a graded assessment system instead of a binary pass/fail criterion, might have provided deeper insights into the models' capabilities. Moreover, exploring other dataset configurations or training models to simulate various skill levels could be avenues for further research.

In conclusion, this project successfully demonstrated the adaptability of Large Language Models (LLMs) to a domain traditionally dominated by specialized AI systems. It also highlighted the significance of model configuration and comprehensive evaluation methodologies in achieving meaningful and accurate outcomes in AI research.

6 Conclusions and Prospects for Further Research

6.1 Conclusion

This project represents a successful attempt to apply Large Language Models (LLMs) in the field of chess. It focused on assessing whether LLMs, specifically trained with chess notation data, could generate valid chess moves adhering to the rules of the game.

The project was conducted in two distinct phases: initial experiments with the relatively simpler game of Connect 4, followed by the more complex task of developing and training a model for chess. Connect 4 served as a basic testing ground, allowing for the refinement of methods subsequently applied to chess. This phase included creating a custom tokenizer and adapting game notation into a format suitable for LLM training, highlighting the crucial role of dataset composition and model configuration in AI research.

Several metrics were employed to evaluate the models' performance, each examining different aspects of the model's ability to understand and replicate chess moves. These included syntactical analysis, semantical evaluation, and statistical analysis. The findings revealed a strong correlation between the amount of training data and the model's effectiveness, with larger datasets typically yielding better results.

The research also emphasized the importance of input structure, particularly how the inclusion of the Beginning of Sequence (BOS) token significantly reduced syntax errors and enhanced overall accuracy. Moreover, the study underlined the need for comprehensive evaluation methods to attain meaningful and precise outcomes.

In summary, this research successfully demonstrated the LLMs' adaptability to the rule-based environment of chess. This project has not only offered valuable insights into LLMs' capabilities in gameplay but has also set a precedent for future AI applications in other complex, rule-governed systems.

6.2 Outlook

This chapter discusses into several possible strategies and ideas for future research. From enhancing the model's training dataset to incorporating advanced game state representations, each section presents a unique perspective on potential developments in AI and chess modelling.

6.2.1 Active Learning

Active Learning involves enriching the training dataset with examples specifically targeting the model's weaknesses. This method concentrates on scenarios where the model struggles to generate valid moves, such as the complex or uncommon positions depicted in Figure 28. These instances, similar to those utilized in the "Hard Position Accuracy" validation metric (Section 4.5.3), are systematically identified and added to the dataset. This approach deepens the understanding of the model's learning process, highlighting how it interprets various positions and adapts to new information. It also aligns with more efficient AI training methods, focusing on strategic data enrichment over large, generalized datasets.

6.2.2 Extended LAN Plus

Extended LAN Plus (xLAN+) is a conceptual expansion of Extended LAN (xLAN), aiming to embed additional game information into each move's notation. xLAN+ follows the same basic structure as xLAN but includes an extra "move status" token at each move's end. This token provides additional

details, such as whether the move involves a capture, check, or checkmate. The notation possibilities in xLAN+ include:

- **"-"** for standard moves
- **"x"** for captures
- **"+"** for checks
- **"#"** for checkmates
- **"$"** for capture combined with a check
- **"!"** for capture combined with checkmate

xLAN+'s advantage is its potential to enable a language model to learn chess moves more effectively by providing comprehensive contextual information. This expanded notation system presents a promising avenue for future research to explore how additional information impacts the model's learning capability.

6.2.3 Training at Different ELO Levels

Training models based on different Elo ratings represents another approach. This strategy involves creating a diverse dataset from games played at various skill levels. A chess engine can simulate games at different Elo ratings, or games from databases like Lichess can be filtered by Elo ratings. During validation, the model's performance against engines set at varying Elo levels would indicate its ability to learn and apply chess strategies. Consistent success against lower-rated engines would suggest effective strategic learning, offering insights into LLMs' potential in mastering complex games like chess.

6.2.4 Incorporating Enhanced Game State Representation in Training

A novel training methodology could focus on integrating enhanced representations of the board state at each move. This strategy departs from predicting the next move and emphasizes generating the game board state from a sequence of moves. Employing Forsyth-Edwards Notation (FEN), a technique successfully used in the study "Learning Chess with Language Models and Transformers" (2022) [10], or similar encoded representations could facilitate this training approach. An alternative within this method involves a supervised learning strategy, where the model is trained to map the current game state to all possible legal moves, enabling it to analyse a game state and enumerate potential moves.

7 Directories

7.1 References

[1] J. Bronowski, *The Ascent of Man*. British Broadcasting Corporation, 1973.

[2] S. Maharaj, N. Polson, and A. Turk, "Chess AI: Competing Paradigms for Machine Intelligence," *Entropy*, vol. 24, no. 4, p. 550, Apr. 2022, doi: 10.3390/e24040550.

[3] N. Parmar et al., "Image Transformer." arXiv, Jun. 15, 2018. Accessed: Dec. 10, 2023. [online]. Available: http://arxiv.org/abs/1802.05751

[4] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A Generative Model for Music." arXiv, Apr. 30, 2020. Accessed: Dec. 10, 2023. [online]. Available: http://arxiv.org/abs/2005.00341

[5] K. Lu, A. Grover, P. Abbeel, and I. Mordatch, "Pretrained Transformers as Universal Computation Engines." arXiv, Jun. 30, 2021. Accessed: Dec. 10, 2023. [online]. Available: http://arxiv.org/abs/2103.05247

[6] S. Chinchalkar, "An Upper Bound for the Number of Reachable Positions," *ICGA J.*, vol. 19, no. 3, pp. 181–183, Sep. 1996, doi: 10.3233/ICG-1996-19305.

[7] M. Campbell, A. J. Hoane, and F. Hsu, "Deep Blue," *Artificial Intelligence.*, vol. 134, no. 1–2, pp. 57–83, Jan. 2002, doi: 10.1016/S0004-3702(01)00129-1.

[8] D. Silver et al., "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." arXiv, Dec. 05, 2017. Accessed: Oct. 17, 2023. [online]. Available: http://arxiv.org/abs/1712.01815

[9] J. Schrittwieser et al., "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020, doi: 10.1038/s41586-020-03051-4.

[10] M. DeLeo and E. Guven, "Learning Chess with Language Models and Transformers," in *Data Science and Machine Learning*, Academy and Industry Research Collaboration Center (AIRCC), Sep. 2022, pp. 179–190. doi: 10.5121/csit.2022.121515.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv, May 24, 2019. Accessed: Oct. 17, 2023. [online]. Available: http://arxiv.org/abs/1810.04805

[12] D. Noever, M. Ciolino, and J. Kalin, "The Chess Transformer: Mastering Play using Generative Language Models," 2020, doi: 10.48550/ARXIV.2008.04057.

[13] T. Barton, "Teaching a language model to play chess." Accessed: Dec. 10, 2023. [online]. Available: https://medium.com/@tbarton_16336/teaching-a-language-model-to-play-chess-1ea69dde40fd

[14] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners." Feb. 14, 2019. Accessed: Nov. 14, 2023. [online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[15] University of Applied Sciences Upper Austria, Digital Media Department, Hagenberg, Austria and A. Stöckl, "Watching a Language Model Learning Chess," in *Proceedings of the Conference Recent Advances in Natural Language Processing - Deep Learning for Natural Language Processing Methods and Applications*, INCOMA Ltd. Shoumen, BULGARIA, 2021, pp. 1369–1379. doi: 10.26615/978-954-452-072-4_153.

[16] T. B. Brown et al., "Language Models are Few-Shot Learners." arXiv, Jul. 22, 2020. Accessed: Oct. 10, 2023. [online]. Available: http://arxiv.org/abs/2005.14165

[17] OpenAI, "GPT-4 Technical Report." arXiv, Mar. 27, 2023. Accessed: Dec. 12, 2023. [online]. Available: http://arxiv.org/abs/2303.08774

[18] A. Chowdhery et al., "PaLM: Scaling Language Modeling with Pathways." arXiv, Oct. 05, 2022. Accessed: Dec. 12, 2023. [online]. Available: http://arxiv.org/abs/2204.02311
[19] H. Touvron et al., "LLaMA: Open and Efficient Foundation Language Models." arXiv, Feb. 27, 2023. Accessed: Dec. 12, 2023. [online]. Available: http://arxiv.org/abs/2302.13971
[20] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks." arXiv, Dec. 02, 2015. Accessed: Dec. 12, 2023. [online]. Available: http://arxiv.org/abs/1511.08458
[21] R. M. Schmidt, "Recurrent Neural Networks (RNNs): A gentle Introduction and Overview." arXiv, Nov. 23, 2019. Accessed: Dec. 12, 2023. [online]. Available: http://arxiv.org/abs/1912.05911
[22] A. Vaswani et al., "Attention Is All You Need." arXiv, Aug. 01, 2023. Accessed: Dec. 11, 2023. [online]. Available: http://arxiv.org/abs/1706.03762
[23] "Transformer Model Architecture." Dec. 01, 2023. [online]. Available: https://huggingface.co/datasets/huggingface-course/documentation-images/resolve/main/en/chapter1/transformers.svg
[24] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training." Nov. 06, 2018. [online]. Available: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf
[25] "HuggingFace NLP Course." Accessed: Dec. 11, 2023. [online]. Available: https://huggingface.co/learn/nlp-course/chapter1/1
[26] R. Merritt, "What Is a Transformer Model?," NVIDIA Blog. Accessed: Dec. 11, 2023. [online]. Available: https://blogs.nvidia.com/blog/what-is-a-transformer-model/
[27] S. J. D. Prince, *Understanding deep learning*. Cambridge, Massachusetts: The MIT Press, 2023.
[28] J. Slegers, "Connect Four." Accessed: Dec. 20, 2023. [online]. Available: https://commons.wikimedia.org/wiki/File:Connect_Four,_John_Slegers.png
[29] J. Tromp, "John's Connect Four Playground," John's Connect Four Playground. Accessed: Oct. 28, 2023. [online]. Available: https://tromp.github.io/c4/c4.html
[30] V. Allis, "A Knowledge-Based Approach of Connect-Four: The Game Is Solved: White Wins," *ICGA J.*, vol. 11, no. 4, pp. 165–165, Dec. 1988, doi: 10.3233/ICG-1988-11410.
[31] S. Edelkamp and P. Kissmann, "Symbolic Classification of General Two-Player Games," in *KI 2008: Advances in Artificial Intelligence*, A. R. Dengel, K. Berns, T. M. Breuel, F. Bomarius, and T. R. Roth-Berghofer, Eds., in Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 185–192. doi: 10.1007/978-3-540-85845-4_23.
[32] H. Wang, M. Emmerich, and A. Plaat, "Monte Carlo Q-learning for General Game Playing." arXiv, May 21, 2018. doi: 10.48550/arXiv.1802.05944.
[33] E. Alderton, E. Wopat, and J. Koffman, "Reinforcement Learning for Connect Four," 2019. [online]. Available: https://web.stanford.edu/class/aa228/reports/2019/final106.pdf
[34] S. Toshniwal, S. Wiseman, K. Livescu, and K. Gimpel, "Chess as a Testbed for Language Model State Tracking." arXiv, May 13, 2022. Accessed: Dec. 10, 2023. [online]. Available: http://arxiv.org/abs/2102.13249
[35] E. Steven J., "PGN-Standard.txt." Accessed: Nov. 28, 2023. [online]. Available: https://opensource.apple.com/source/Chess/Chess-110.3.4/Documentation/PGN-Standard.txt.auto.html

7.2 List of Figures

7.3 List of Tables

8 Appendix

8.1 Links

GitHub Repository Chess:

- https://github.zhaw.ch/schmila7/leon-llm/

GitHub Repository Connect Four:

- https://github.zhaw.ch/schmila7/connect4-llm

Hugging Face Chess Models and Dataset Repository:

- https://huggingface.co/collections/Leon-LLM/leon-llm-chess-6584387dbef870ffa4a7605f

Hugging Face Connect Models and Datasets Repository:

- https://huggingface.co/collections/Leon-LLM/leon-llm-connect-four-65849f9db0b0e7e38a19cfc0

8.2 GitHub Readme

Language Models Explore the Linguistics of Chess

Introduction

This repository contains the research work and codebase for training a Large Language Model (LLM) solely on chess game sequences. The goal is to train the LLM to understand and replicate the rules of chess, make legal moves, and predict chess game outcomes without explicit rule-based programming.

The datasets used to train the models are encoded in our unique chess Notation xLAN.

Quick Start

This guide will help you get started with the Chess Language Model Research project. Follow these simple steps to set up the environment and run your first example.

Prerequisites

- Ensure you have Python 3.11.6 installed on your system. You can download it from Python's official website.
- Basic knowledge of Python programming and familiarity with chess game rules.

Installation

1. Clone the repository to your local machine:
   - Open your terminal.
   - Navigate to the directory where you want to clone the repository.
   - Run `git clone https://github.zhaw.ch/schmila7/leon-llm`.
   - Navigate to the cloned repository by running `cd leon-llm`.
2. Install the required dependencies:
   - In the repository's root directory, run `pip install -r requirements.txt`.

Running Your First Example

1. Open a notebook:
   - In the repository's root directory, run `jupyter notebook`.
   - In the Jupyter Notebook interface, navigate to the notebooks folder.
   - Open the `useModel.ipynb` notebook.
2. Run the notebook:
   - Follow the instructions in the notebook to interact with the trained chess model.
   - Experiment with playing a chess game against the model or use the model to predict the next move in a given chess position.

Notebooks

In this section, you'll find Jupyter notebooks designed to facilitate various stages of the LLM development and use. These notebooks serve as an interactive interface to run processes and execute code in a step-by-step fashion. The notebooks are interconnected with the Python files, calling upon them as needed. Instead of running Python scripts directly, it is recommended to perform all actions through these notebooks.

1. `analysis.ipynb`: Analysis of trained models and datasets, including visualizations and statistics.
2. `dataPreProcessing.ipynb`: Conversion tools between different chess notations and dataset preparation.
3. `train.ipynb`: Notebook for training and evaluating the chess model.
4. `uploadDownload.ipynb`: Utilities to upload and download datasets and models from/to Hugging Face.
5. `useModel.ipynb`: Interface to play chess against the model, for self-play, and to predict the next move from a given position.

Python Files

This section includes Python scripts that contain specific functionalities needed for the LLM's training and operation. These files are typically not run directly. Instead, they are integrated into the Jupyter notebooks, providing the underlying logic and processing power required for tasks such as data preprocessing, model training, and validation.

- `pgn_to_xlan.py`: Conversion of PGN format files into the custom xLAN format.
- `tokenizer.py`: Tokenizer utilizing a JSON mapping file for dataset tokenization.
- `detokenizer.py`: Reverses the tokenization process for datasets.
- `train.py`: Script to train models using GPT-2 configurations on chess datasets.
- `validate_model.py`: Validates models using various metrics like "Average Number of Correct Plies", "Hard Position Accuracy", and "Legal Piece Moves Accuracy".
- `validate_position.py`: Validates models using specific positions defined in a JSON file.
- `validate_sequence.py`: Validates generated move sequences for their legality.

- check_duplicates_and_common_lines.py: Checks and removes common lines and duplicates between datasets.
- chess_game.py: Framework for playing chess games.
- generate_prediction.py: Generates predictions using trained models.
- notation_converter.py: Converts between different chess notations (e.g., xLAN to UCI, UCI to xLAN).

Data Folder

The data folder contains training and validation files required for the project.

xLAN Format

xLAN, an adaptation of LAN (Long Algebraic Notation), was developed to provide a uniform and fixed-length format, facilitating the prompting process for our Language Model. Unlike standard LAN, xLAN explicitly specifies the piece type for each move. This results in a consistent three-token structure per move: {piece}{start_square}{end_square}.

Format Illustration

- **PGN Example:** 1. g3 e6 2. Bg2 Qf6 3. f4 Bc5 4. e4 Qd4 5. e5 Qf2# 0-1
- **LAN Conversion:** 1. g2-g3 e7-e6 2. Bf1-g2 Qd8-f6 3. f2-f4 Bf8-c5 4. e2-e4 Qf6-d4 5. e4-e5 Qd4-f2# 0-1
- **xLAN Adaptation:** 1. Pg2-g3 Pe7-e6 2. Bf1-g2 Qd8-f6 3. Pf2-f4 Bf8-c5 4. Pe2-e4 Qf6-d4 5. Pe4-e5 Qd4-f2# 0-1

Special Moves in xLAN:

- **Castling:** Indicated explicitly by the king's movement, e.g., {king}{start_square}{end_square}.
- **Pawn Promotion:** Represented as {goal_piece}{start_square}{end_square}.

Tools and Resources Used

1. **Weights & Biases:** A machine learning experiment tracking tool used for visualizing and comparing our models' training progress.
2. **Python Chess:** A chess library for Python, providing essential functionalities for chess manipulations and move generation.
3. **Lichess Database:** Source of comprehensive chess game data, utilized for training and testing our LLM.
4. **Hugging Face:** A platform for sharing and collaborating on machine learning models, used for hosting our trained models and datasets.

Feel free to contribute or use this research for academic purposes. For any questions or collaboration, please open an issue or pull request or send us an email:
maagjer@students.zhaw.ch / schmila7@students.zhaw.ch

8.3 Project Management

The project was organized using the project management tool Jira. A Kanban approach was utilized to define and manage workflow stages, ensuring that each development phase received adequate focus and resources. The primary tool for managing the project was a Kanban board, which assisted in prioritizing and tracking the progress of identified action items.

Figure 55 shows the timeline of the key milestones with their anticipated start and end dates.

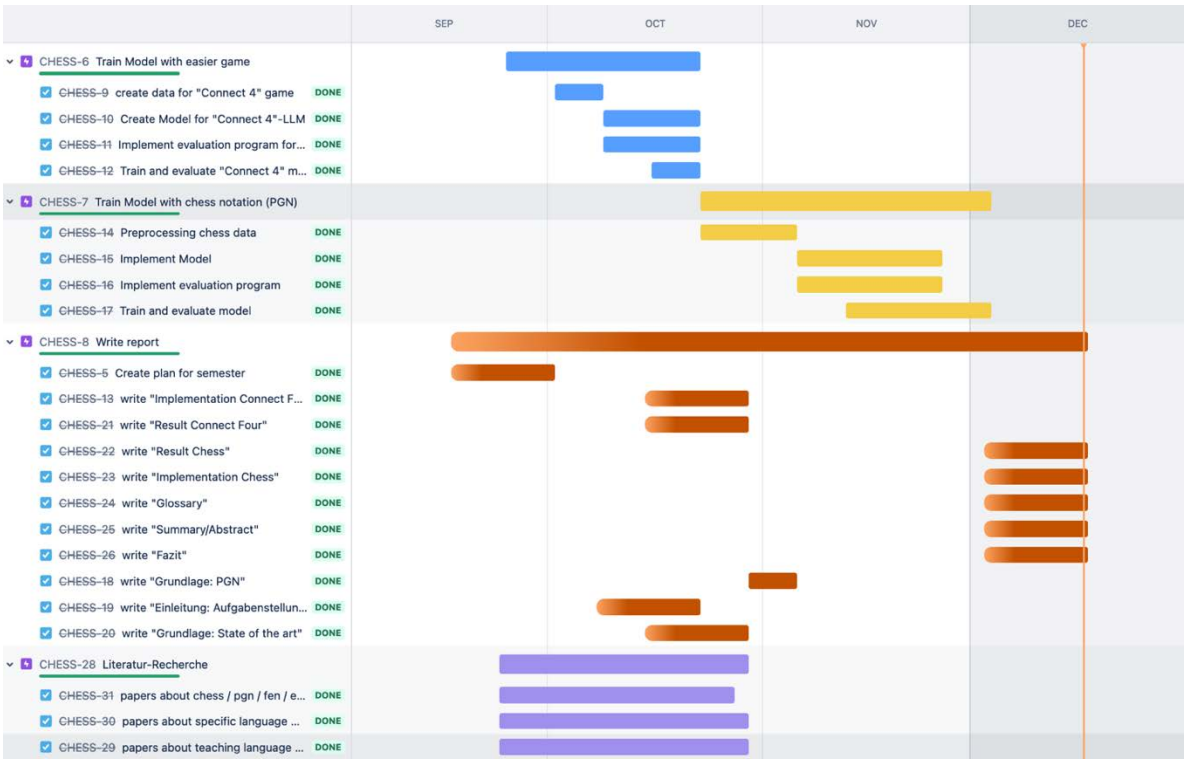

Figure 55: Project Timeline

Figure 56 illustrates the project management workflow, as created in the first weeks of the project, detailing the planned phases of the project, namely "Training the Connect 4 Model", "Training the Chess Model", and "Report Writing". The illustration additionally includes key questions and identifies the tools and technology to be employed in each phase.

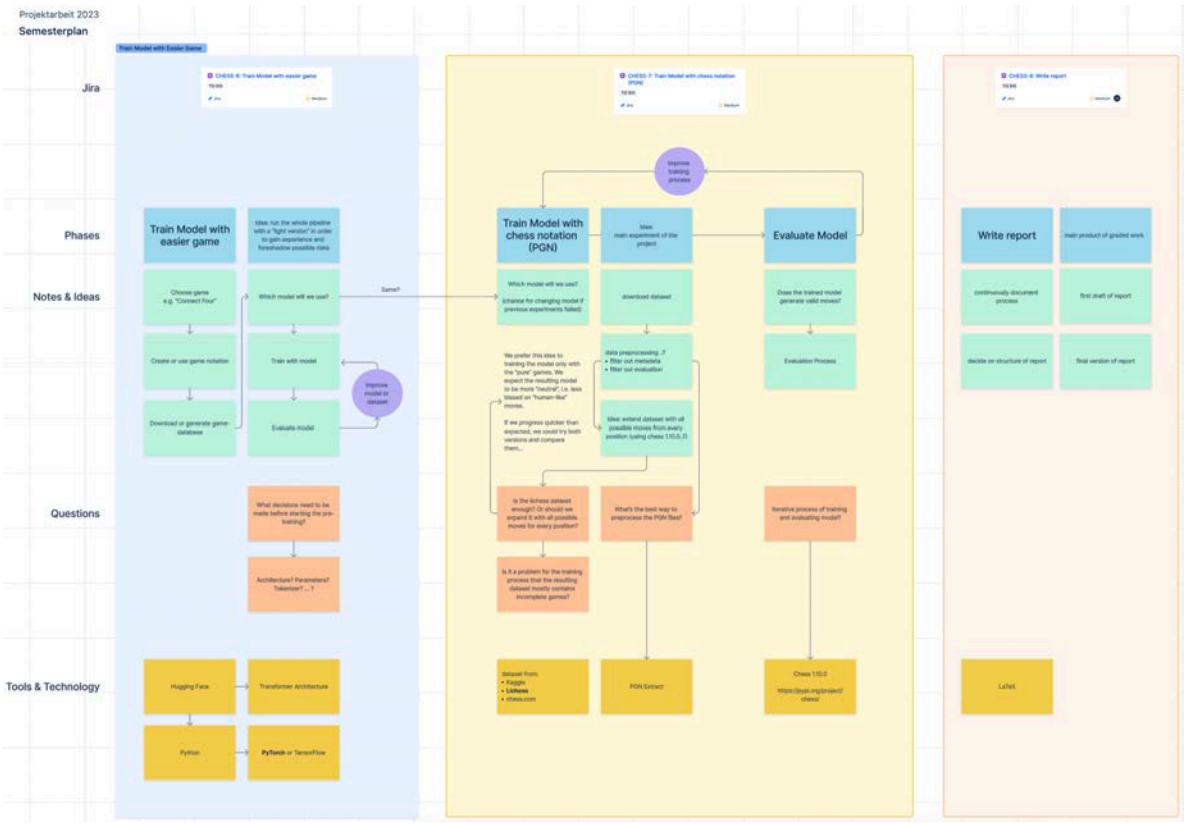

Figure 56: Project Management Workflow

8.4 Formal Syntax of Chess Notation

8.4.1 Formal Syntax of PGN notation [35]

```
<PGN-database> ::= <PGN-game> <PGN-database>
                   <empty>
<PGN-game> ::= <tag-section> <movetext-section>
<tag-section> ::= <tag-pair> <tag-section>
                  <empty>
<tag-pair> ::= [ <tag-name> <tag-value> ]
<tag-name> ::= <identifier>
<tag-value> ::= <string>
<movetext-section> ::= <element-sequence> <game-termination>
<element-sequence> ::= <element> <element-sequence>
                       <recursive-variation> <element-sequence>
                       <empty>
<element> ::= <move-number-indication>
              <SAN-move>
              <numeric-annotation-glyph>
<recursive-variation> ::= ( <element-sequence> )
<game-termination> ::= 1-0
                       0-1
                       1/2-1/2
                       *
<empty> ::=
```

8.4.2 Formal Syntax of SAN notation[32]

<SAN-move> ::= <PawnMove>
| <PieceMove>
| <PawnCapture>
| <PieceCapture>
| <Castle>
| <Promotion>
| <Check>
| <Checkmate>
<PawnMove> ::= <file>? <rank>
<PieceMove> ::= <Piece> <disambiguation>? <file><rank>
<PawnCapture> ::= <file>x<file><rank> <Promotion>?
<PieceCapture> ::= <Piece> <disambiguation>? x <file><rank>
<Castle> ::= O-O | O-O-O
<Promotion> ::= =<Piece>
<Check> ::= +
<Checkmate> ::= #
<Piece> ::= K | Q | R | B | N
<disambiguation> ::= <file> | <rank> | <file><rank>
<file> ::= a | b | c | d | e | f | g | h
<rank> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

[32] This syntax is simplified, not considering all the edge cases.