Optimizing Language Models for Chess: The Impact of Custom Notation and Elo-Based Fine-Tuning

Authors: Jerome Maag, Lars Schmid
Supervisors: Mark Cieliebak, Pius von Däniken
Institution: ZHAW School of Engineering

Date: 7th June 2024

Abstract

This research investigates the potential for large language models to learn to generate valid chess moves solely through pre-training on chess game data. The primary objective of this study is to investigate the impact of custom notation systems and tokenisation methods specifically designed for use with chess games. The study aims to improve the models' understanding of game states and move sequences by developing and implementing a custom notation system, xLAN+. This notation system incorporates additional information about captures, checks and checkmates in order to increase the performance of the models. Furthermore, the strategic gameplay capabilities of the models are to be enhanced by fine-tuning them on datasets filtered by Elo rating. This approach postulates that filtering games by skill level can help models develop a deeper understanding of strategic gameplay, thereby improving their ability to generate high-quality moves. The research uses two different architectures, Transformer, based on OpenAI's GPT-2 configuration, and Mamba, a State Space Model (SSM) optimised for long sequence processing. Initial findings indicate that the custom notation xLAN+ significantly improves the models' ability to generate valid moves and maintain game state accuracy over extended sequences. The comparison between GPT-2 and Mamba reveals that while both architectures can learn chess rules and generate plausible moves, the SSM offers slight advantages in handling long-range dependencies and maintaining game context. This project demonstrates the potential of language models to learn complex tasks like chess through data-driven approaches, paving the way for their application in other strategic and decision-making domains.

Keywords: AI; Chess; GPT-2; LLM; Mamba; NLP

Contents

1 Introduction

1.1 Glossary and Abbreviations

This section provides a glossary of terms and abbreviations to assist the reader in understanding the specialized language and concepts related to Artificial Intelligence (AI), Machine Learning (ML), and chess, as utilised throughout this research.

Table 1: List of Abbreviations

Term	**Definition**
Artificial Intelligence (AI)	The simulation of human intelligence in machines, enabling them to perform tasks that typically require human cognition, such as learning, reasoning, problem-solving, and understanding language.
Average Centipawn Loss (ACPL)	A metric used in chess to measure the accuracy of a player's moves compared to the optimal moves suggested by a chess engine. It represents the average difference in centipawns (one hundredth of a pawn) between the chosen move and the best move.
Extended Long Algebraic Notation (xLAN)	A custom version of LAN specific to this project's methodology. In contrast to LAN, it consistently includes the piece that is moved, e.g., `Pd2d4`. It uses the following scheme: `{piece}` `{start_square}` `{end_square}`.
Extended Long Algebraic Notation Plus (xLAN+)	An enhanced version of xLAN using the tokens `{piece}` `{start_square}` `{end_square}` `{move_status}`.
Generative Pre-trained Transformer (GPT)	LLM by OpenAI[1] that uses a transformer architecture and is pre-trained on a large corpus of data to generate coherent and contextually relevant text based on input prompts.
Long Algebraic Notation (LAN)	A method of recording chess moves in a detailed format, specifying both the starting and ending squares of a move, e.g., `d2d4`.
Large Language Model (LLM)	An AI model trained on vast amounts of text data to understand, generate, and manipulate human language, using deep learning techniques to perform tasks like text completion, translation, summarization, and question answering.
Learning Rate	In ML, the learning rate determines how quickly a model learns. A higher learning rate results in faster learning, whereas a lower rate results in slower, potentially more comprehensive learning.
Low-Rank Adaptation (LoRA)	A technique in ML that reduces the number of parameters needed for fine-tuning LLMs, making the process more efficient and faster without significantly sacrificing performance.
Machine Learning (ML)	A subset of AI where systems learn from data to improve their performance on tasks without being explicitly programmed.
Natural Language Processing (NLP)	A field of AI focused on the interaction between computers and human language. It involves techniques for analyzing, understanding, and generating natural language text.
Ply / Plies	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.
Portable Game Notation (PGN)	A plain text format used for recording chess games, including move sequences, player information, and game results.

[1] `https://openai.com/`

Term	Definition
Stockfish[2]	An open-source, powerful chess engine known for its high level of play and frequent use in chess analysis and competitions. It is considered one of the strongest chess engines available.
Standard Algebraic Notation (SAN)	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 square 'e4'.
Temperature	A parameter that influences the randomness of predictions by a model. Higher temperatures generally result in more varied and less predictable outputs.
Tokenisation	The process of breaking down text into smaller numerical units called tokens, which are used in language processing and model training.
Transformer Architecture	A deep learning model architecture primarily used for NLP tasks. It utilises self-attention mechanisms to process input data in parallel, enabling more efficient and accurate handling of long-range dependencies in text.

1.2 Motivation

The motivation for this project arises from recent advances in AI and Natural Language Processing (NLP), particularly in the development of Large Language Models (LLMs) such as GPT-2. These models have demonstrated remarkable potential in understanding and generating human language, while simultaneously opening up possibilities for their use in a range of domains, including chess. Chess, as a game of strategy, logic, and complex decision-making, serves as an ideal testbed for evaluating the capabilities of LLMs. These models can learn and predict complex sequences, making them suitable for learning chess rules, strategies, and moves purely from data.

Another key motivation is to explore the impact of different chess notation systems on model performance. Notation systems like Extended Long Algebraic Notation Plus (xLAN+) are expected to improve the model's understanding of game states and move sequences, leading to more accurate and strategic move generation. The development and evaluation of these notations are driven by the necessity to identify the most effective methods for encoding and utilising game information.

This project examines the potential for enhancing the strategic gameplay of these models. Conventional chess engines rely on search algorithms and handcrafted evaluation functions, whereas LLMs provide a data-driven approach that can streamline the development process and offer new insights into game strategy. By fine-tuning models on datasets of chess games filtered by Elo rating, the project aims to demonstrate that LLMs can enhance their chess-playing proficiency without relying on expensive search algorithms.

In summary, this project is aligned with the objective of making AI more accessible and effective across various domains. By applying LLMs to chess, the project seeks to bridge the gap between theoretical AI research and practical applications, thereby demonstrating the versatility and potential of these models.

[2] `https://stockfishchess.org/`

1.3 Task

The primary task of this project is to apply LLMs in the domain of chess. This will be achieved by developing and training language models using extensive datasets of chess games, followed by fine-tuning these models to enhance their understanding of strategic gameplay.

A crucial component of this task is the evaluation of model performance. This will include implementing metrics to assess the accuracy and effectiveness of the models in generating legal and strategic chess moves. Comparing the performance of the different architectures, namely Transformer and Mamba will provide insights into their capabilities in handling complex game scenarios and generating valid move sequences.

The data preparation phase involves the collection and preprocessing of large datasets of chess games to create a robust training set. Various notation systems, including Portable Game Notation (PGN) and Long Algebraic Notation (LAN), will be used to encode chess games for model training. Extended notation systems such as xLAN+ will be developed and evaluated to enhance the model's comprehension of game states and move sequences. Experimentation with these notation systems is essential to investigate their impact on the model's performance. Additionally, by fine-tuning the models on data sets filtered by Elo rating, the project aims to evaluate the ability of the models to generate strategic chess moves. The evaluation of the quality of the move generation will be a key part of this evaluation.

Another crucial task is the optimisation of the training process, which involves the selection of appropriate hardware configurations and training methodologies. Techniques such as Low-Rank Adaptation (LoRA) will be applied to enhance the efficiency of model fine-tuning without compromising performance. These efforts aim to demonstrate the potential of language models in learning and playing chess, providing insights into the capabilities and limitations of AI in strategic game modelling.

1.4 Related Work

Recent advances in the application of language models to chess have demonstrated their ability to learn complex game rules and strategies through extensive training on game data. This section reviews key studies that highlight the progress and capabilities of language models in mastering chess.

Noever et al.[1] explored how natural language transformers can be applied to strategic modelling in chess. The study utilises GPT-2, fine-tuned on 2.8 million chess games in PGN. Their model has demonstrated the ability to generate plausible chess strategies and recognisable openings. The study makes a significant contribution to the field by demonstrating that the Transformer can learn chess rules and strategies without the need for expert intervention, relying solely on extensive text-based game data. The model generated approximately 1,000 games per run, with a failure rate of 2-8% for non-viable gameplay or illegal moves.

DeLeo and Guven (2022)[2] used the language model BERT (Bidirectional Encoder Representations from Transformers) to learn and play chess by training on text-based representations of game positions and moves. Using Forsyth-Edwards Notation (FEN) and algebraic notation, the model was trained on datasets of grandmaster games and evaluated against the chess engine *Stockfish*. The authors demonstrated that the BERT model could learn chess rules and strategies by observing sequences of moves, reaching a level of proficiency to survive games against *Stockfish* for an average of 32

moves, with the longest game lasting over 200 moves. Furthermore, the model showed an accuracy rate of 75% in generating valid chess moves.

Stöckl (2021)[3] analysed how a transformer-based language model learns chess rules from text data of recorded games. The study evaluated the influence of model size, training time, and data volume on learning success using GPT-2, trained on varying amounts of chess game data ranging from 100k games up to 2.1M games. The research demonstrated that the quantity of training data has a significant impact on the results, while the impact of model size is less clear. Models trained on larger datasets consistently demonstrated superior performance in generating legal moves and handling typical and random chess positions.

In a study published in 2024, Ruoss et al. (2024)[4] investigated a 270M parameter transformer model to ascertain whether it was possible to learn and play chess at the level of a grandmaster without the use of explicit search algorithms. The model, trained on 10 million chess games annotated with action values provided by *Stockfish* (Version 16) achieved a *Lichess*[3] blitz Elo rating of 2895 against humans. This research demonstrated that a neural network can approximate a strong chess engine like *Stockfish* through supervised learning at scale, excelling at generating legal and strategic moves and solving chess puzzles up to Elo 2800.

Karvonen (2024)[5] extended previous work by exploring how language models can develop internal representations and latent variable estimations in chess. The study demonstrated that a Generative Pretrained Transformer (GPT) model trained on 16 million chess games from *Lichess* could learn to play legal and strategic moves and develop an internal representation of the chessboard state using character-level tokenisation. Key contributions include the development of linear probes to recover the model's internal board state and the introduction of interventions to edit this state. The study showed that a 16-layer model had a 64% win rate against *Stockfish* 16 level 0 (arround 1300 elo), while an 8-layer model achieved a 46% win rate. Including skill vectors into the model resulted in a 2.6 fold increase in the win rate on random board states.

These studies have demonstrated that language models are capable of generating plausible chess strategies, recognising openings, and even play at grandmaster level without the need for explicit search algorithms. The results of these studies highlight the progress and capabilities of language models in mastering chess, including their ability to learn chess rules and strategies, generate legal moves, and solve chess puzzles. The findings also suggest that larger datasets and more advanced model architectures can lead to improved performance.

[3] `https://www.lichess.org/`

2 Foundations

2.1 Large Language Models

This chapter discusses the fundamental concepts and specific models employed in this research to train a Large Language Model (LLM) on chess data. LLMs are a class of models that are particularly adept at processing large amounts of textual data and generating natural language from that data. These models have made significant progress in recent years and are utilised in a variety of applications, including machine translation and automated text generation.

Two different architectures were used in this research: GPT-2 [6] and Mamba [7]. Generative Pretrained Transformer 2, abbreviated as "GPT-2", is a transformer model. In contrast, Mamba is a State Space Model (SSM), an architecture that is fundamentally different from transformers.

Although the architectures of GPT-2 and Mamba are notably different, the methods used to train and deploy them in this project are very similar. The experiments in this project can be understood by considering these architectures as black boxes, trained to recognise patterns in the data and generate new data based on those patterns.

The model configurations have been chosen to be approximately the same size and dimensions, and therefore have very similar training times.

2.1.1 GPT-2 (Transformer Architecture)

A comprehensive account of the transformer model was presented in our previous project work[8], which can be found in Section 7.1.1.

2.1.2 Mamba (State Space Model)

Mamba is a state-space model architecture designed to address the computational inefficiencies of the Transformer architecture, particularly in handling long sequences. Mamba was proposed by researchers at Carnegie Mellon University and Princeton University as a solution for LLMs to efficiently process and generate text from large data sequences. Mamba is based on the concept of State Space Models (SSMs), which incorporate elements of Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs) and continuous-time models. The core innovation of Mamba is its ability to efficiently manage long-term dependencies through linear-time sequence modelling, making it better suited to handle long and complex data sequences.[7]

Selective State Spaces The key component of Mamba is the Selective State Space layer. Unlike traditional SSMs that use fixed parameters, Mamba's SSM parameters are functions of the input, allowing the model to dynamically adjust its state based on the relevance of each token. This selection mechanism allows Mamba to effectively propagate or discard information, focusing on only the critical data within sequences. This approach helps to avoid the quadratic complexity associated with the self-attention mechanism in transformers and allows linear scaling with sequence length.[7]

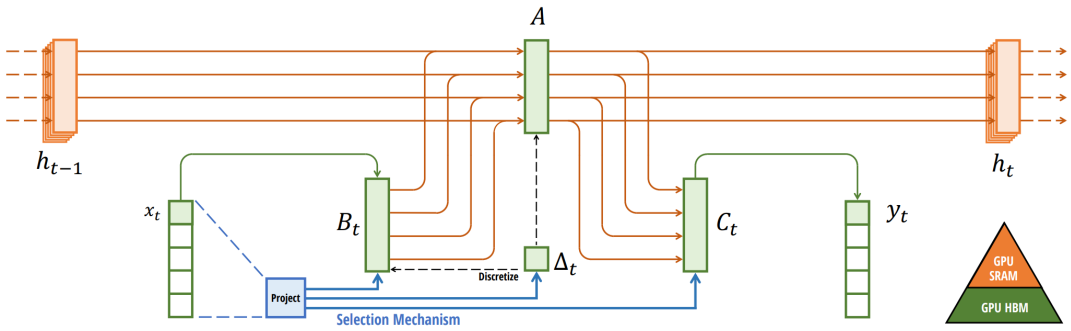

Figure 1: Mamba Architecture [7]

Architecture Figure 1 provides a visual representation of Mamba's architecture, with the following key components and their interactions:

- **State transition ($h_{t-1} \rightarrow h_t$):** The hidden state h_{t-1} from the previous time step is transformed into the current hidden state h_t using the matrix A that defines the dynamics of the state transition.
- **Input Projection (x_t):** The current input x_t is projected by a selection mechanism that determines its relevance. This projection modifies the parameters B_t and C_t based on the input.
- **Selection mechanism (Δ_t):** The parameter Δ_t is adjusted based on the input x_t. This adjustment is critical for dynamic modulation of state transitions and is achieved by a discretisation process that adapts Δ_t to the current input.
- **Output (y_t):** The output y_t is generated from the current state h_t using the matrix C_t, which is also set by the input through the selection mechanism.

Comparison with Transformer Models Mamba's simplified architecture, without complicated attention and Multilayer Perceptron (MLP) blocks, results in lower complexity and faster inference speeds. In addition, Mamba's design is optimised for modern GPU hardware, ensuring efficient parallel processing and memory usage. In contrast, Transformers, despite their high complexity, are effective for tasks requiring deep contextual understanding and retrieval from long contexts. However, Mamba's innovative approach offers a promising alternative, especially for applications that require efficient processing of very long sequences.[7][9]

2.1.3 LoRA (Fine-Tuning Methodology)

LoRA is a method for adapting large pre-trained language models to downstream tasks in an efficient manner. It functions by freezing the original model weights and introducing trainable low-rank matrices to approximate the updates needed for adaptation. This approach significantly reduces the number of trainable parameters in comparison to full fine-tuning. The LoRA technique achieves this by decomposing the weight update matrices into a product of two smaller matrices, thereby imposing a low-rank structure. This method preserves the performance advantages of full fine-tuning while reducing computational and storage demands, and it avoids adding inference latency. Empirical evidence suggests that LoRA performs at a comparable or superior level to traditional fine-tuning in a range of experiments involving models such as GPT-3 and RoBERTa.[10]

Parameters of LoRA

- **Parameter** r: This parameter represents the rank of the low-rank matrices introduced by LoRA. It controls the dimensionality of the adaptation matrices. A smaller value of r signifies a reduction in the number of trainable parameters, which can result in a reduction in computational requirements and storage needs.
- **LoRA Alpha** α: This scaling factor is employed to adjust the influence of the low-rank matrices on the overall adaptation process. The value of α determines the extent to which the low-rank updates contribute to the final adapted model. Higher values of α may enhance the model's ability to adapt to new tasks but may also increase the risk of overfitting.
- **LoRA Dropout**: This parameter introduces dropout regularisation to the low-rank matrices during training. Dropout helps prevent overfitting by randomly dropping a fraction of the adaptation units during each training step.

2.2 Metrics for Evaluating Chess Strategy

Rating systems are essential tools for evaluating the performance of chess players and chess models. This chapter provides an overview of the main rating systems used in our project to assess the quality of the moves generated by the models. The primary systems discussed are the Elo rating system, which measures the relative skill levels of players, and Average Centipawn Loss (ACPL), which evaluates the precision of individual moves made by a chess engine.

The Elo rating system was employed to filter the datasets utilised for fine-tuning the chess models (see Section 4.4). According to the scores provided in the PGN data, while ACPL served the purpose of evaluating the strategic proficiency of the models.

It is possible to conceive of alternative ways for evaluating the quality of game play by chess engines. For instance, game positions could be analysed with heuristic evaluation metrics, which consider aspects such as pawn structure, piece activity, and king safety. However, such metrics are often subjective and challenging to quantify accurately. Alternatively, many models could compete against each other many times in order to evaluate their Elo ratings, as exemplified by the approach employed by Computer Chess Rating Lists (CCRL)[4]. This approach was not considered for this research project due to the computational effort and time required.

2.2.1 Elo Rating System

The Elo rating system is the most widely recognised method for ranking chess players. It is utilised by online platforms such as chess.com[5] and lichess.org[6], as well as by the World Chess Federation (FIDE) for global player rankings. The Elo rating system was developed by Arpad Elo in the 1960s [11] and was initially designed for chess. However, it has since been adapted to rate players in other competitive sports, including American football and baseball. More recently, it has even been applied to LLMs in research contexts [12].

The primary objective of the Elo rating system is to quantify the relative skill levels of players in zero-sum games. It operates on the fundamental principle that players gain or lose rating points based on the outcomes of their matches, with these points being

[4] `https://computerchess.org.uk/ccrl/4040/index.html`
[5] `https://www.chess.com/`
[6] `https://www.lichess.org/`

adjusted according to the relative strength of their opponents. For instance, winning against a higher-rated opponent results in a greater gain in points than winning against a lower-rated one.

The outcome of each match leads to a recalibration of ratings through a formula that considers the players' current ratings and the match result. This dynamic adjustment process helps to maintain an accurate reflection of a player's current form and skill level.

One criticism of the Elo system is that it does not explicitly account for the probability of draws, which can be a significant factor, especially in chess. This has led to research aimed at refining and pushing the limits of the Elo rating algorithm to better represent the nuances of competitive play.[13]

2.2.2 Average Centipawn Loss

The Centipawn Loss (CPL) metric quantifies the precision of a chess move by measuring the loss in pawn value compared to the optimal move available according to a robust chess engine such as *Stockfish*. A centipawn is one hundredth of a pawn's value and serves as a standard unit to evaluate the strength of chess moves. Centipawn loss measures the degradation in a player's position following a move, calculated by noting the difference in the game's evaluation before and after the move as determined by a chess engine.

The ACPL metric extends this concept to evaluate a player's performance across an entire game. By averaging the value lost per move in terms of centipawns, ACPL provides a comprehensive assessment of how a player's decisions compare to those suggested by a powerful chess engine. This reflects the precision and quality of the player's performance over an entire game.

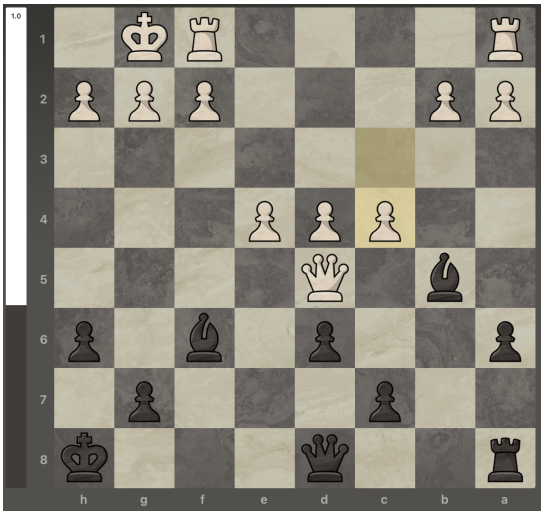

Figure 2: Analysis of Position on chess.com, Showing an Advantage for White of +1.00 Centipawn

The metric ACPL is frequently employed by online chess platforms, such as chess.com and lichess.org, to assist players in analysing their games and comprehending the quality of their moves in relation to optimal play. This analysis is typically represented in the form of a sidebar (see Figure 2). The number indicates the advantage in terms of pawn units. For instance, a +1.00 evaluation suggests that white has an advantage roughly equivalent to one pawn, all other factors being equal.

A crucial difference between evaluating the quality of a game using the Elo rating system and ACPL is that ACPL does not specifically depend on the outcome of a game. Nevertheless, it has been demonstrated that ACPL is correlated with Elo rating [14]. Furthermore, it offers the advantage of being directly quantifiable and easily comprehensible, thereby facilitating straightforward comparisons between models.

It should be noted that the ACPL metric is not without its limitations. For instance, not all errors are equally significant. A single misstep can result in the loss of a game, yet this may not be accurately reflected by the ACPL metric. Additionally, the resulting value of this metric depends on the quality and type of chess engine that is used, as well as the computational depth used for evaluating the game position.

It is important to note that one might intuitively assume that playing the optimal move is independent of the strength of the opponent. However, a strong opponent often plays moves leading to positions where the number of moves not resulting in a worse position for their opponent is very small. This decreases the probability of the opponent playing a move that will not result in an increase in ACPL. Conversely, a strong player may elect not to play the optimal move in a winning position if it leads to positions with a limited number of available moves that do not increase ACPL. Instead, they may opt for "safer" moves that also lead to victory but in a less risky manner.[15]

2.3 Chess Notation

The following section provides a detailed description of xLAN+, a custom chess notation specifically developed for this research. For a more comprehensive overview of the different notations, including Portable Game Notation (PGN), Long Algebraic Notation (LAN), and Extended Long Algebraic Notation (xLAN), please refer to Section 7.1.2.

2.3.1 xLAN+

This chapter builds on the foundation laid by Extended Long Algebraic Notation (xLAN) (for a detailed description of xLAN, please refer to Appendix, Section 7.1.2). It introduces xLAN+, a conceptual expansion that includes additional information in the notation of each move.

In addition to the tokens used in xLAN, namely `{piece}`, `{start_square}`, and `{end_square}`, xLAN+ includes an additional `{move_status}`-token at the end of each move. This additional token contains information about captures, checks, and checkmates.

- `-` denotes standard moves, where no capture, check, or checkmate occurs.
- x signifies captures, indicating the taking of an opponent's piece.
- `+` denotes checks, where the king is put under threat but not captured.
- `#` indicates checkmates, marking the end of the game with the king's capture being inevitable.
- `$` represents captures combined with a check, signaling both the capture of a piece and the placement of the opponent's king in check.
- `!` is used for captures combined with checkmate, indicating the dual action of capturing a piece while simultaneously delivering checkmate.

The conversion from PGN to xLAN and subsequently to xLAN+ can be illustrated as follows:

```
PGN   1.  e4     e5     2.  Bc4    Nc6    3.  Qh5    Nf6    4.  Qxf7#   1-0

xLAN  1.  Pe2e4  Pe7e5  2.  Bf1c4  Nb8c6  3.  Qd1h5  Ng8f6  4.  Qh5f7   1-0

xLAN+ 1.  Pe2e4- Pe7e5- 2.  Bf1c4- Nb8c6- 3.  Qd1h5- Ng8f6- 4.  Qh5f7!  1-0
```

Following the tokenisation process (for further details, please refer to Section 7.1.4), the final input to be used to train the model will take the following form:

```
Tokens 75 6 40 42 76 6 45 43 76 4 47 26 76 5 22 28 76 2 31 67 76 5 62 52
76 2 67 53 80 71 74
```

3 Implementation

This section outlines the practical aspects of the research, detailing the steps and methodologies employed to implement the chess models. This includes data preparation, training processes, and the hardware used for model training.

3.1 Data Preparation

This section covers the processes of data collection, cleaning, and preprocessing, which are crucial for ensuring that the models are trained on relevant and high-quality datasets.

3.1.1 Data Collection

The data used for training the models was sourced from the *Lichess* database[7], which is a comprehensive repository of chess games played on the *Lichess* platform. Games played in September 2023 were used to train the baseline model, while games played in March 2024 were used for the fine-tuning dataset. The datasets were downloaded in PGN format. For a detailed description of the different notations for chess, please refer to Section 7.1.2. The September 2023 dataset comprises a total of 93,218,629 games, while the March 2024 dataset contains 95,810,349 games.

3.1.2 Data Processing

The data processing pipeline for the foundation models, which transforms the raw PGN data into tokenised datasets, follows the methodology of our previous project work. For a detailed description of the data filtering and conversion process, please refer to the Appendix, Section 7.1.3. The data processing involves several key steps: filtering the dataset to select suitable games, converting the data into custom chess notation, filtering based on start sequences, and tokenising the data.

Data Processing for Fine-tuning In order to fine-tune the models, the dataset "March 2024" was utilised. The data preprocessing procedure was identical to that employed in the creation of the foundation model datasets, with the exception of one key difference: prior to the conversion of the notation, the dataset was divided into two subsets, one comprising all games with Elo ratings of up to 1,000, and the other containing games with Elo ratings above 2,000. These subsets were then converted, tokenised, and filtered. Figure 3 illustrates the distribution of games played per Elo rating on *Lichess* in March 2024. The data indicates that 76% of games have Elo ratings between 1,000 and 2,000, 18% of games are played by players with Elo ratings above 2,000, while the remaining 6% are played by players with Elo ratings below 1,000.

[7] `https://database.lichess.org/`

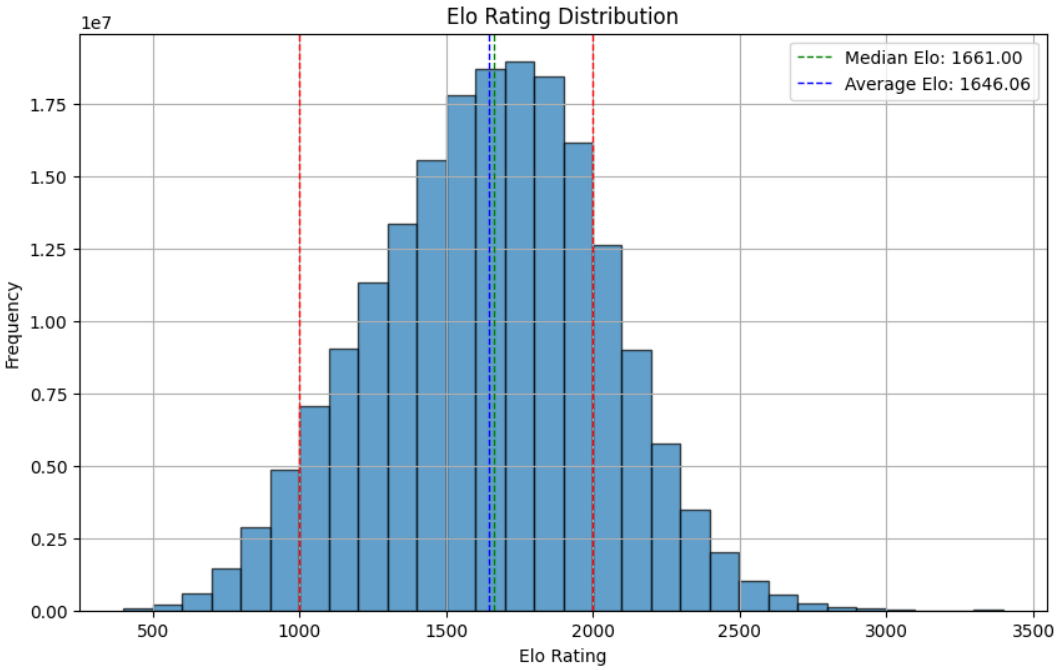

Figure 3: Elo Distribution of Dataset "March 2024"

Table 2 provides a comprehensive overview of the number of games in the datasets at each stage of data processing. Despite a higher total number of games above 2000 Elo, the datasets for high Elo games become smaller after filtering for specific openings. This is because players with a higher Elo rating tend to explore fewer opening variations, preferring well-studied and highly effective openings. In order to achieve a comparable size to the high Elo game dataset, the lower Elo game datasets were adjusted by truncating the most recent games.

Table 2: Overview Preprocessing of the Datasets for Fine-tuning

Preprocessing Step	**Size (Games)**	
	0-1000 Elo	**2000+ Elo**
Raw Lichess dataset March 2024	95,810,349	
Splitting into Elo boundaries	5,059,626	17,448,688
Filtering suitable games and conversion to xLAN	1,742,970	3,890,524
Removing duplicated lines	1,694,628	3,738,837
Tokenising dataset	1,694,628	3,738,837
Removing all games with more than 500 Tokens	1,618,405	3,414,202
Keeping only one game for the first 20 Tokens	130,271	98,711
Keeping only one game for the first 16 Tokens	37,411	29,430
Keeping only one game for the first 13 Tokens	13,249	10,564

Tokenisation for xLAN+ In order to accommodate the additional symbols required for the xLAN+ notation, we utilised the library `python-chess` to determine specific game conditions, such as whether a piece is captured, a check is given, or the game has ended by checkmate. The corresponding symbols were then appended to the moves. Furthermore, the tokenisation structure was extended to include the following indicators:

```
{
    "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},
    "plusTokens": {"-": 76, "+": 77, "#": 78, "$": 79, "!": 80, "x": 81}
}
```

To illustrate, the move `Pe2e4-` is now represented by four tokens: `P`, which denotes the piece type pawn; `e2`, which indicates the starting square; `e4`, which denotes the ending square; and `-`, which denotes the move indicator.

3.2 Training

The library `transformers`[8], available from *Hugging Face*[9], was employed for the training of the models. This library was used to store and manage both datasets and models. The chosen platform was selected for its user-friendly interface and direct access to model configurations on *Hugging Face*, as well as its straightforward training class, which simplifies the process of training the models. Furthermore, *PyTorch*[10] was employed to facilitate training, while *Weight & Biases (W&B)*[11] was employed for logging the validation metrics during training. The fundamental training parameters employed across all models are as follows:

- **Batch size:** 16
- **Number of epochs:** 4
- **Learning rate:** 0.0001
- **Learning rate scheduler:** linear
- **Mixed Precision Training (FP16):** Yes
- **Optimiser:** AdamW

3.2.1 GPT-2

For the transformer models, the standard configuration provided by *Hugging Face* for the GPT-2 model was employed. This will be referred to as GPT-2 throughout the remainder of this report. The key parameters of the model are as follows:

- **Hugging Face Architecture:** GPT2LMHeadModel
- **Activation function:** `gelu_new`
- **Number of layers:** 12
- **Number of heads:** 12
- **Embedding dimensions:** 768
- **Maximum number of positions:** 1024
- **Vocabulary size:** 82 (for xLAN+)

[8] `https://huggingface.co/docs/transformers/en/index`
[9] `https://huggingface.co/`
[10] `https://pytorch.org/`
[11] `https://wandb.ai/`

3.2.2 Mamba

The Mamba model is based on the pre-configured version of the `mamba-130m-hf` model. The key parameters of the model are as follows:

- **Hugging Face Architecture:** MambaForCausalLM
- **Activation function:** Sigmoid Linear Unit (SiLU)
- **Number of layers:** 24
- **Number of hidden layers:** 24
- **Model size:** 768
- **Inner size:** 1536
- **Vocabulary size:** 82 (for xLAN+)

3.2.3 LoRA

For fine-tuning the models using LoRA, the following hyperparameters were employed:

- **Rank of the update matrices (r)**: 64
- **Scaling Factor** α: 32
- **Dropout**: 0.1

3.2.4 Hardware and Training Duration

The training hardware comprised a Linux server on the ZHAW APU cluster, equipped with a Tesla T4 graphics card with 16 GB of memory, 16 GB of RAM, and 2 TB of HDD storage for the models and training data.

Training durations varied across datasets. The approximate training times for 4 epochs with a batch size of 16 were as follows:

- 1.5 hours for the 19k dataset
- 6 hours per session for the 71k dataset
- 22 hours for 350k dataset

4 Experiments

This chapter presents a comprehensive overview of the experiments conducted to evaluate the performance of our chess models. The experiments are designed to test various aspects of model performance, including an exploration of the xLAN+ and its impact on model performance, an examination of various padding strategies, the comparative performance of the Mamba architecture, and an analysis of the impact of fine-tuning on strategic gameplay.

4.1 Extended Long Algebraic Notation Plus (xLAN+)

The primary objective of this experiment was to ascertain the impact of the additional token `move_status` on the performance of models trained using the xLAN+ notation in generating valid chess moves.

Previous work by the same authors [8] has demonstrated that the trained chess models exhibited deficiencies in generating moves in specific positions defined by the metric "Hard Position Accuracy". These positions involved the model being compelled to block an attack on the king or to react to a move leading to a discovered check in order to generate a valid move. It is anticipated that the model will be able to resolve such scenarios more effectively by incorporating additional information pertaining to captures, checks, and checkmates into each move.

4.1.1 Comparison to Extended Long Algebraic Notation (xLAN)

In order to ascertain whether the use of the xLAN+ notation results in enhanced performance, GPT-2 models were trained on datasets of 19k, 71k and 350k in size, which were then evaluated and using the metrics "Average Number of Correct Plies", "Hard Position Accuracy", and "Legal Piece Moves Accuracy". The results of these experiments are described in the following paragraphs.

Average Number of Correct Plies The metric "Average Number of Correct Plies", as developed in previous work by the same authors [8], measures the capability of the models to generate game sequences starting from the initial board setup. Details about the metric "Average Number of Correct Plies" can be found in Section 7.1.5. In previous research, this metric was calculated by generating 100 sequences with a maximum number of 80 plies each. For this project, the number of generated sequences is 1,000, with a maximum number of 125 plies each, the temperature parameter remaining at 0.7.[12]

In addition to the error categories for xLAN (namely, "Syntax", "Piece Logic", "Path Obstruction", "Pseudolegal", and "No Error"), an additional error category called "Indicator Error" is required for the notation xLAN+, which signifies the failure of the model to correctly generate the fourth token of a ply, containing information about captures, checks, and checkmates. This implies that the models trained on this novel notation not only gain access to more information per ply during training, but they must also learn how to generate this information correctly with each ply.

As illustrated in Figure 4, the xLAN+ notation has been demonstrated to result in a notable enhancement in the models' capacity to generate move sequences without

[12]The temperature parameter of 0.7 was chosen to balance the diversity and determinism of the generated game sequences. A temperature of 0.0 would result in the model always generating the same game, whereas a higher temperature would introduce too much variance, resulting in less coherent sequences.

any errors, in comparison to the models trained on the same data using the xLAN notation.

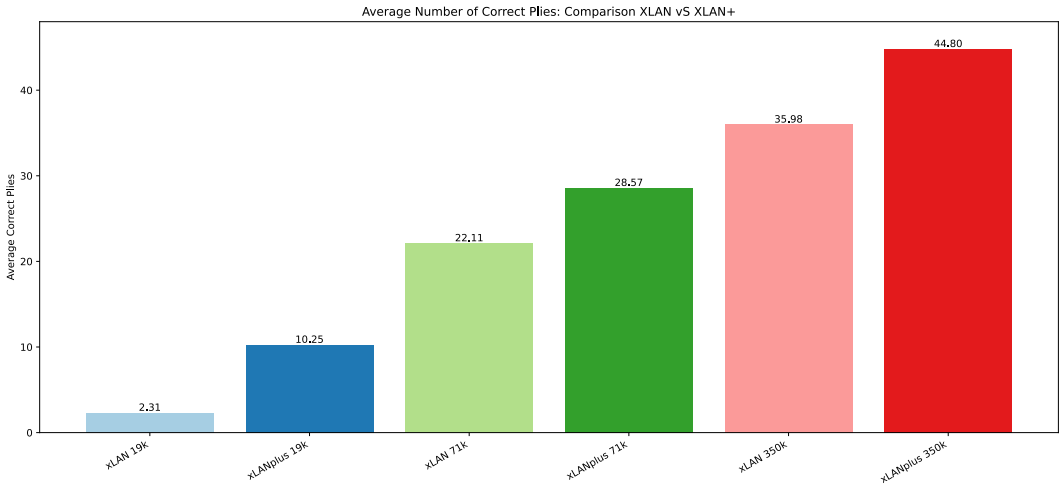

Figure 4: Comparison of Model Performance for xLAN and xLAN+ ("Average Number of Correct Plies")

Hard Position Accuracy The metric "Hard Position Accuracy" encompasses 67 manually selected positions. A detailed description of the metric "Hard Position Accuracy" can be found in Section 7.1.5. The metric is designed to evaluate the model's capability to generate correct plies in challenging scenarios, including castling, promotions, or being forced to avoid checkmate.

Figure 5 illustrates the significant enhancements observed when utilising xLAN+ instead of xLAN for the metric "Hard Position Accuracy". However, the performance of xLAN+ is marginally inferior to that of xLAN in the case of models trained on 71k games.

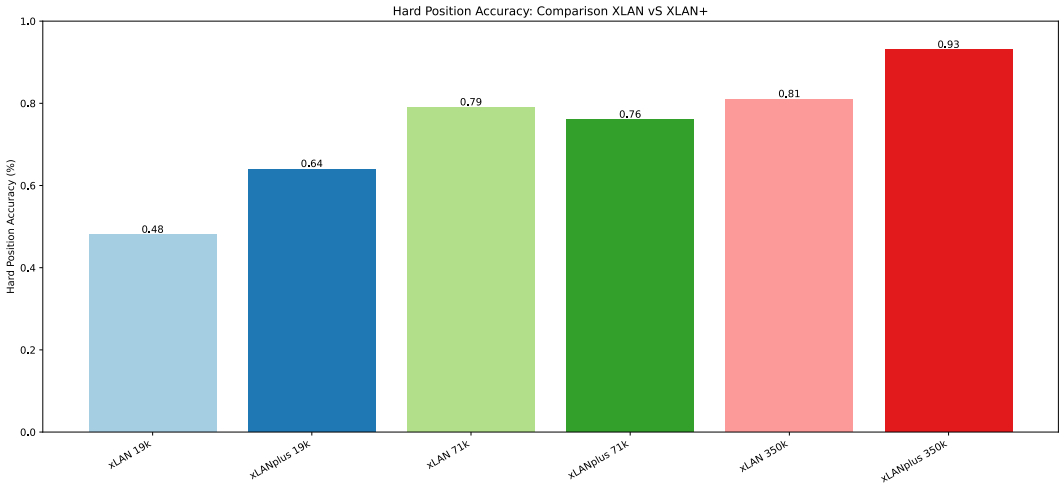

Figure 5: Comparison of Model Performance for xLAN and xLAN+ ("Hard Position Accuracy")

The model trained on a dataset comprising 350k games achieved an accuracy of 93%, which was superior to the model trained on 1 million games using the xLAN notation by approximately 10% (for further details, please refer to [8], Section 5.2).

A clear illustration of the impact of the `move_status` token can be observed in the position depicted in Figure 6, where White has castled queenside, placing the rook on ‘d1’ and checking the black king on ‘d7’. The xLAN models do not recognise this check. The moves predicted by the xLAN models are ‘Ne4d2’ (19k), ‘Bc8d7’ (71k) and ‘Pb7b5’ (350k). The move ‘Ne4d2’ by the 19k model would prevent the check, but it is not permissible for the black player to move a white piece. The 71k model attempts to move the bishop from ‘c8’ to ‘d7’, which would result in the black king capturing its own piece, an illegal move. The 350k model suggests moving the pawn to ‘b5’, a legal move if the black king were not in check. In contrast, the xLAN+ models correctly recognise the check and successfully move the king from ‘d7’ to ‘c7’. In this position, it is challenging to recognise the impending check, as the rook move to ‘d8’ is not explicitly stated in the castling move, ‘Ke1c1’.

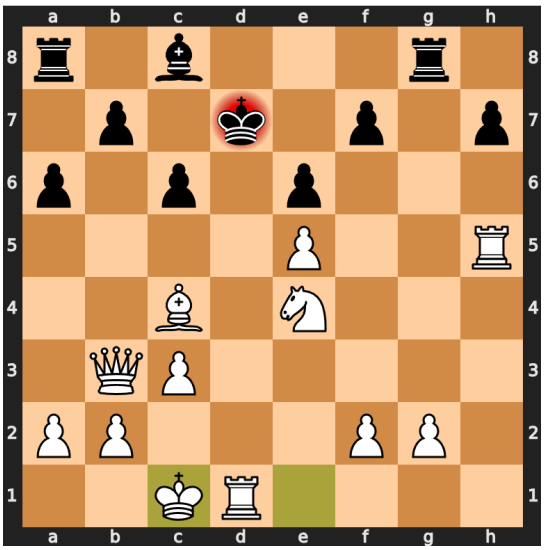

Figure 6: Hard Position Accuracy #20 - Castling into check

Figure 7 illustrates another instance where the xLAN model fails to detect a check. In this position, Black moves the knight from ‘c7’, thereby revealing a discovered check by the rook on c8 against the white king on ‘c1’. The xLAN model does not realise that it is in check and attempts to move its pawns. Conversely, the xLAN+ model, trained with the `move_status` token, correctly identifies the check and moves the white king from ‘c1’ to ‘b1’. This example demonstrates the enhanced ability of the xLAN+ model to accurately identify checks, particularly those that are not immediately apparent. The knight move from ‘c7’ to ‘b5’ does not directly place the white king in check, thereby presenting a more challenging threat identification problem for the xLAN model.

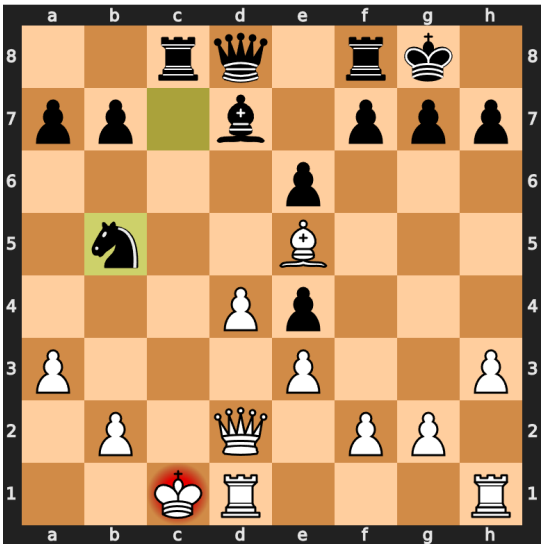

Figure 7: Hard Position Accuracy #36 - Discovered Check

As in the previous scenario, in the position depicted in Figure 8, the white queen on ‘d3’ places the black king on ‘h7’ in check by moving the pawn from ‘e4’ to ‘d5’. The move ‘Pe4d4’ in the xLAN+ notation is annotated with the `move_status` token `$`, indicating that a piece has been captured and a check has been delivered. While the 71k and 350k xLAN models incorrectly suggest the move ‘Qe7d7’, even the smaller xLAN+ models are unable to recognise the check and attempt to recapture the newly positioned pawn on ‘d5’ with the knight from ‘f6’. The 350k xLAN+ model is the only one to correctly address the check by moving the black king from ‘h8’ to ‘h7’.

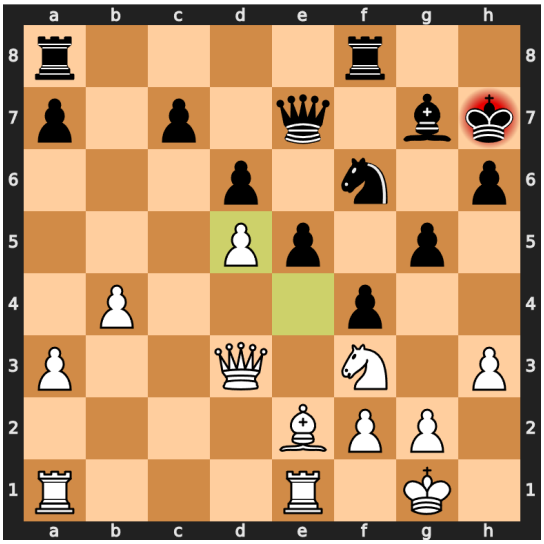

Figure 8: Hard Position Accuracy #38 - Capture into Check

Legal Piece Moves Accuracy The metric “Legal Piece Moves Accuracy” has been devised with the specific objective of evaluating the models’ capacity to accurately represent the board state. For a detailed description of the metric, please refer to Section 7.1.5.

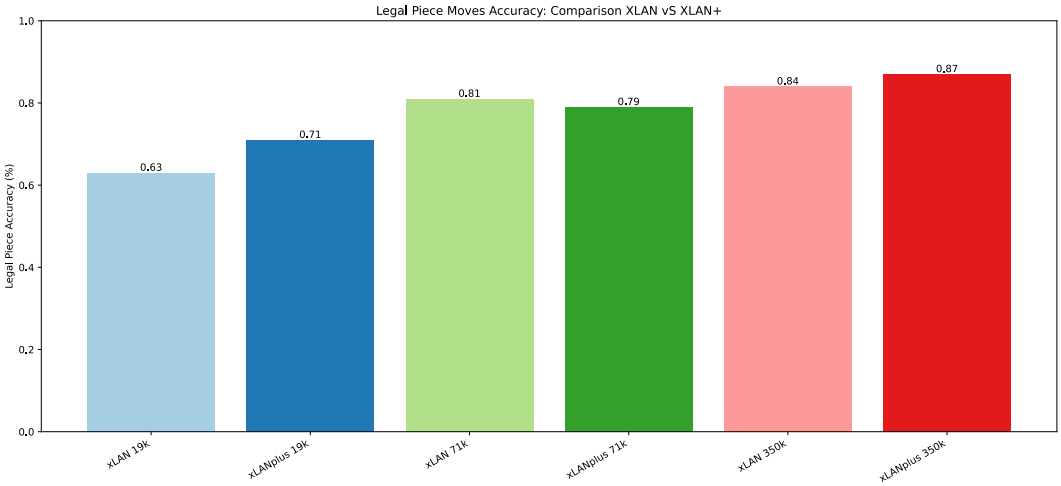

Figure 9: Comparison of Model Performance for xLAN and xLAN+ ("Legal Piece Moves Accuracy")

The results in Figure 9 correlate with the previous two metrics: The additional information provided by the xLAN+ notation generally improves the capability of the models to correctly generate tokens for the positions evaluated by this metric. However, the differences are much less pronounced than with the previous metrics.

There are several positions where the models gave different predictions. To illustrate, Figure 10 depicts the task of identifying all possible destinations for the knight on 'b8'. The models trained with the xLAN notation consistently predicted that the knight should move to 'd7', which is not possible because this square is occupied by a rook of the same colour. In contrast, the models trained with the xLAN+ notation successfully predicted the correct target squares for the knight, namely 'c6' and 'a6'.

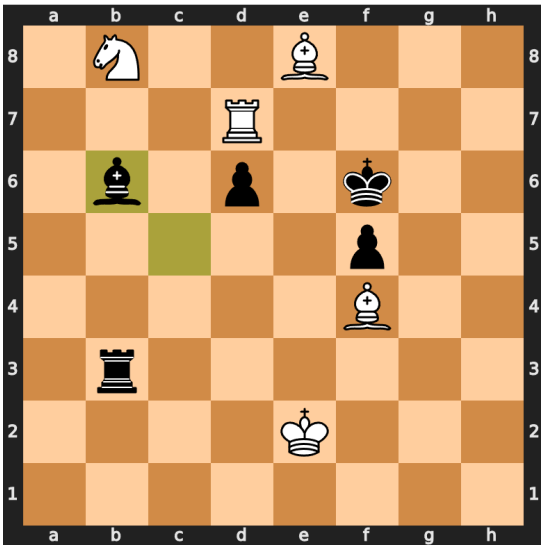

Figure 10: Legal Piece Moves Accuracy #193 - After Promotion to Knight

Figure 11 illustrates a scenario where the xLAN model outperforms the xLAN+ model. All xLAN models correctly identified the king's starting square on 'b5' without any problems, whereas the xLAN+ models predicted other squares such as 'b3', 'b4', or 'e7'. This discrepancy may be attributed to the length of the games being longer than those typically found in the training dataset for xLAN+. Both models were trained

on games with a maximum length of 500 tokens. However, the game in Figure 11 is 131 plies long, which corresponds to 394 tokens in xLAN and 525 tokens in xLAN+. The longer game length may have affected the performance of the xLAN+ models, as they may not have been adequately trained on such extended sequences.

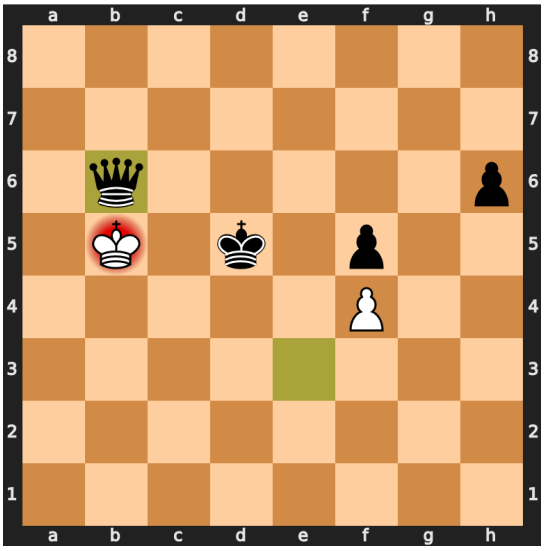

Figure 11: Legal Piece Moves Accuracy #26 - King in Check

In numerous instances, it is possible to predict a substantial number of possible target or starting squares for a piece. In such cases, the xLAN+ models tend to demonstrate superior performance in accurately predicting all these squares. An illustrative example can be found in Figure 12, where the objective is to predict the starting positions of all the white pieces. The 350k xLAN+ model correctly identifies the five pawns that can move: 'a2', 'c2', 'c5', 'f2' and 'h2'. In contrast, the 350k xLAN model predicts the squares 'c4', 'c5', 'f2', 'g3' and 'h2'. Although there are indeed white pawns on 'c4' and 'g3', these pawns are obstructed by other pieces, rendering the aforementioned predictions erroneous.

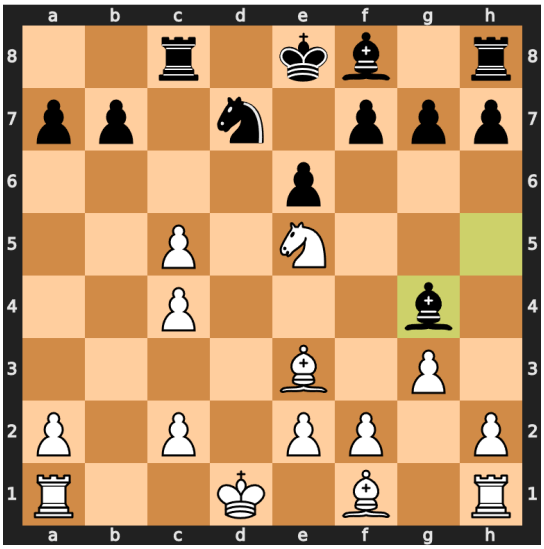

Figure 12: Legal Piece Moves Accuracy #72 - Pawn Moves

4.1.2 Variations of xLAN+

In order to ascertain which of the additional tokens exerts the greatest influence on the performance of the model, two variants of xLAN+ were constructed: xLANchk and xLANcap.

xLANchk only includes information about checking in the fourth token of each move, i.e. `-` (no capture, check, or checkmate), `+` (check), and `#` (checkmate) are used. xLANchk only includes information about capturing in the fourth token of each ply, i.e. only `-`, and `x` (capture) are used.

GPT-2 models have been trained using the same datasets of size 19k and 71k, the only difference being the notation used. The results of comparing these models trained for four epochs on 71,000 games, using the notations xLAN, xLAN+, xLANchk, and xLANcap, are presented in the following figures.

The results in Figure 13 demonstrate that both xLANcap and xLANchk significantly enhance the models' capacity to generate move sequences without errors. However, xLANcap exhibited a slightly superior performance compared to xLANchk.

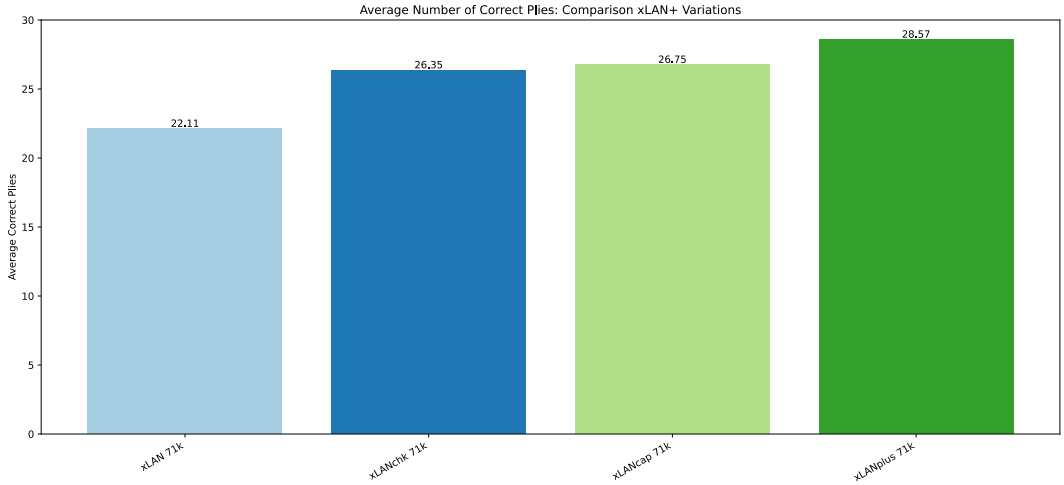

Figure 13: Comparison of Model Performance for xLAN+ Variations ("Average Number of Correct Plies")

Figure 14 presents the results of evaluating the models' performances on the metric "Hard Position Accuracy." It demonstrates that xLANchk outperforms xLANcap, and even outperforms xLAN+. However, the differences between the scores for this metric are much smaller than for the previous metric. The observed fluctuations of up to 0.05 are within the typical variance between subsequent checkpoints during model training.

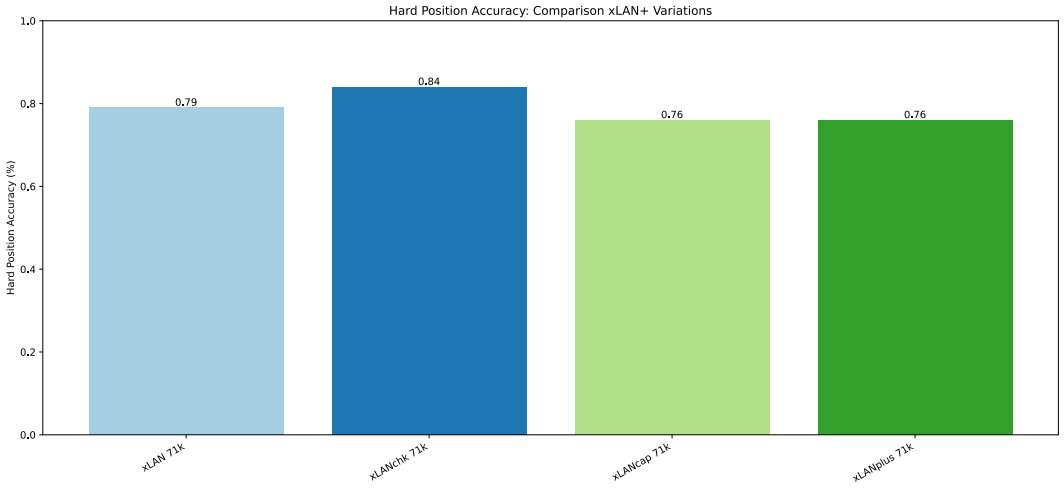

Figure 14: Comparison of Model Performance for xLAN+ Variations ("Hard Position Accuracy")

The metric "Legal Piece Moves Accuracy" indicates an improvement in xLANcap over xLANchk, as illustrated in Figure 15. However, as in the previous figure, the differences between these results are too small to draw any meaningful conclusions.

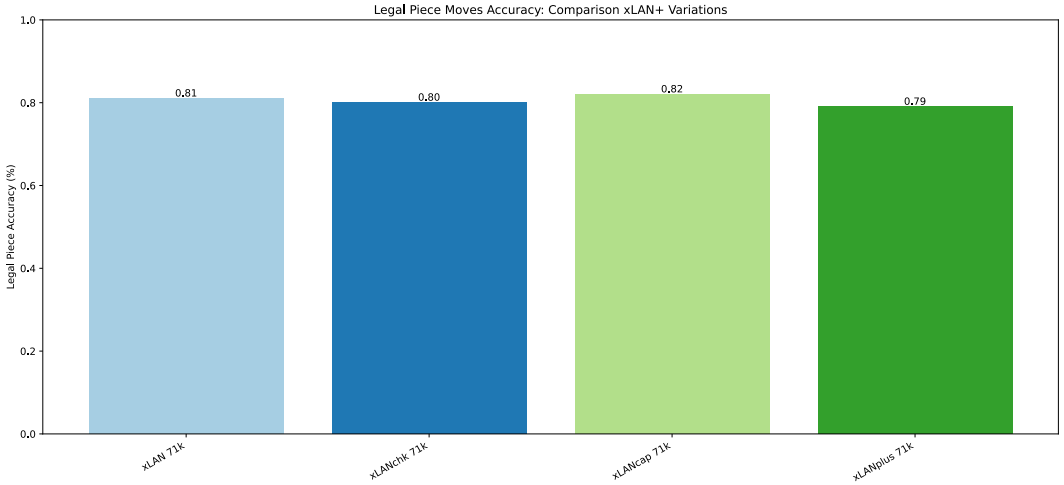

Figure 15: Comparison of Model Performance for xLAN+ Variations ("Legal Piece Moves Accuracy")

4.1.3 Conclusion

The experiments conducted in this chapter demonstrate that the xLAN+ notation provides significant improvements over the standard xLAN notation across multiple metrics. The inclusion of an additional `move_status` token that conveys information about captures, checks, and checkmates, the models trained with xLAN+ show enhanced performance in generating valid chess moves, particularly in complex situations. The metric "Average Number of Correct Plies" indicates that models trained with xLAN+ generate longer sequences of correct moves without errors, thereby demonstrating an enhanced ability to maintain accurate game states. This is further supported by the results from the "Hard Position Accuracy" metric, where xLAN+ models exhibit superior performance in scenarios involving checks, castling, and promotions. It is noteworthy that the xLAN+ models trained on the largest

dataset (350,000 games) achieved an accuracy of 93%, significantly outperforming the xLAN models trained on the same dataset.

Furthermore, the "Legal Piece Moves Accuracy" metric highlights the enhanced capability of the xLAN+ models to maintain legal game states and predict correct moves. However, the improvements are less pronounced than in other metrics. The inclusion of additional game state information in the xLAN+ notation allows the models to better understand and predict legal moves, particularly in positions requiring recognition of discovered checks and intricate maneuvers.

To further refine the understanding of the impact of the `move_status` token, two variants of xLAN+ were tested: xLANchk and xLANcap. Both variants demonstrated significant improvements over the standard xLAN, with xLANcap performing slightly better in generating correct move sequences and xLANchk demonstrating better performance in handling hard positions.

4.2 Padding Variations

Training LLMs on chess data necessitates a preprocessing step to ensure that all game sequences are of equal length. A common method to achieve this uniformity is padding, whereby shorter sequences are lengthened to match the length of the longest sequence in the batch. Initially, left-padding was employed, but this approach led to significant issues with the generation of the end-of-game token.

In left-padding, padding tokens are added to the beginning of shorter sequences to make their total length equal to that of the longest sequence, which in this case was 125 plies. However, during training, the models learned to associate the end-of-game token with reaching a particular position within the sequence, specifically the 125th ply. This was a result of the padding scheme, where the end-of-game token frequently appeared at this fixed position. Consequently, the models developed a proclivity to generate the end-of-game token exclusively after processing 125 plies, irrespective of the actual state of the game. This discrepancy resulted in the models encountering difficulties in accurately predicting the end of games shorter than 125 plies. Rather than learning to identify the natural end of a game based on the game state, the models relied on their position within the padded sequence, which led to erroneous predictions. This behaviour also affected the models' capacity to generate sequences that encompassed valid game endings.

To address this issue, a right-padding strategy was adopted. This approach involves the addition of padding tokens to the end of shorter sequences, ensuring that the relative positions of meaningful game tokens remain intact. This strategy preserves the integrity of the game sequences and allows the models to learn patterns based on actual game play rather than an arbitrary sequence length.

Upon retraining the models with correct padding, significant improvements were observed. The models demonstrated an enhanced capacity to correctly generate long sequences and to conclude games at an earlier stage, including the ability to effectively predict checkmates. The bias towards generating the end-of-game token after a specific number of plies was removed, allowing the models to develop a more comprehensive understanding of game dynamics. This enabled them to predict game endings with greater reliability, irrespective of sequence length.

This resulted in a 10% increase in the number of error-free games for the models trained on 19k and 350k games respectively. Table 3 presents the results of the metric "Average Number of Correct Plies", which compares the effectiveness of left-padding

and right-padding for the GPT-2 and Mamba models, respectively. The results of this metric demonstrate a notable increase for all the models.

Table 3: Comparison of Different Paddings ("Average Number of Correct Plies")

	Left-Padding	**Right-Padding**
GPT-2		
19k	13.54	17.91
71k	30.81	29.38
350k	49.30	51.86
Mamba		
19k	17.11	18.63
71k	25.40	30.80
350k	53.93	55.32

4.3 Mamba Architecture

Previous experiments have demonstrated that the models frequently encounter difficulties in maintaining the complete state of the chessboard over extended sequences. This limitation impacts their ability to generate accurate moves, particularly in longer games where tracking the entire board is crucial. GPT-2 models encounter challenges with long sequences due to their tendency to forget the positions of pieces that have not been moved for an extended period.

The Mamba architecture offers a potential solution to this problem. In contrast to traditional Transformer models, Mamba has been designed to handle long sequences more effectively. Its Selective State Space model is capable of dynamically adapting and retaining relevant information over long sequences, rendering it particularly well suited for tasks that require a deep understanding of context over long periods of time, such as chess. The objective of utilising the Mamba architecture is to enhance the model's capacity to retain and utilise the complete board state throughout the game. This should result in more accurate move generation.

4.3.1 Comparison to GPT-2

Legal Piece Moves Accuracy Figure 16 presents a comparison of GPT-2 models with Mamba models, measuring their performance in the metric "Legal Piece Moves Accuracy". While the GPT-2 models trained on smaller datasets (19k and 71k) demonstrate a higher accuracy compared to their Mamba counterparts, the Mamba model trained on 350k games outperforms the GPT-2 model trained on the same amount of data. In particular, the smaller Mamba models exhibit difficulty in identifying all the correct destination squares when the piece and starting square are given. Nevertheless, when the piece is provided and the objective is to predict the starting square, the models demonstrate comparable performance.

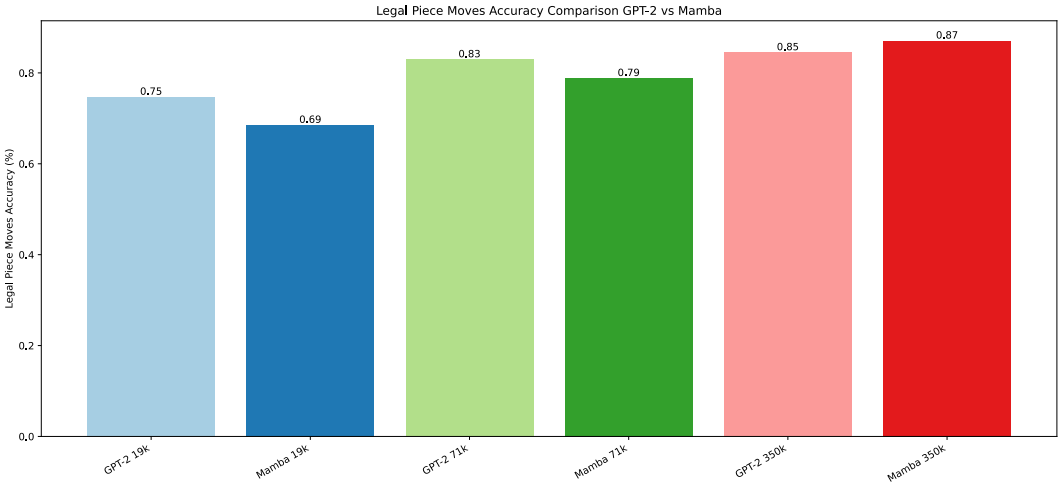

Figure 16: Comparison of Model Performance for GPT-2 and Mamba ("Legal Piece Moves Accuracy")

In certain positions tested in the metric "Legal Piece Moves Accuracy", the GPT-2 and Mamba models provide different answers. These differences are explained in more detail in the following sections.

In the position shown in Figure 17, the model's task is to identify all possible starting squares for a pawn move by White. White's only pawn is located on its starting square 'h2'. All three GPT-2 models (19k, 71k, 350k) correctly identified this square. However, the Mamba models exhibited considerable inaccuracies. The Mamba 19k model predicted 'g7', occupied by a black pawn, as the most probable starting square, with 'h2' not appearing in the top 25 predictions. The 71k model predicted the square 'f4', which is occupied by a black pawn, and the square 'a3', which is occupied by a white knight. The probability of 'h2' occurring was less than 0.01%. The 350k model predicted the knight on 'a3' with a probability of 97.5%, and correctly identified 'h2' with a probability of 2.3%.

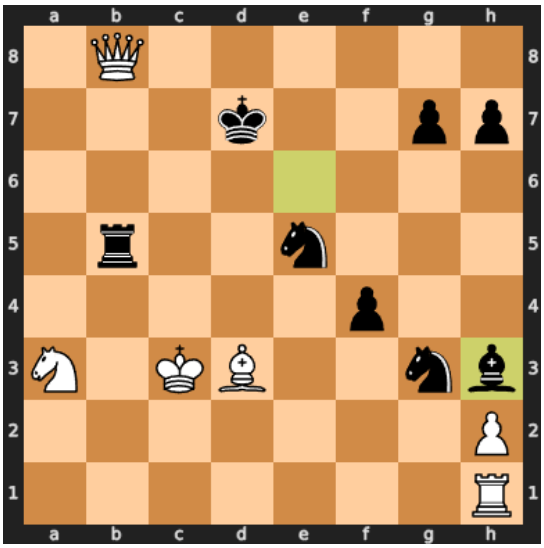

Figure 17: Legal Piece Moves Accuracy #94 - Possible Start Squares for Pawn

It is possible that the Mamba model is unaware that the pawn on 'h2' has not yet moved, as the game has already progressed to 68 plies. Another possible reason is

that the Mamba models do not predict the pawn's move because the square directly in front of the pawn is occupied, and the only legal move for the pawn would be a diagonal capture, which may not be immediately recognised by the model.

In the position depicted in Figure 18 the objective is to identify all legal pawn moves for White. The only model that successfully solved this task was GPT-2 350k. While the majority of models, including GPT-2 71k and all Mamba models, correctly identified 'e2' as a potential move, only the GPT-2 350k model correctly identified both 'e2' and 'd2' as legal pawn moves. The Mamba 350k model predicted 'f5' with a probability of 42.9% when a pawn is unable to move. The model predicted 'e2' 41.9% of the time and 'b3' only 7.8% of the time. This indicates that while some models can partially identify legal moves, comprehensive recognition of all possible pawn moves remains a challenge, particularly for the Mamba models.

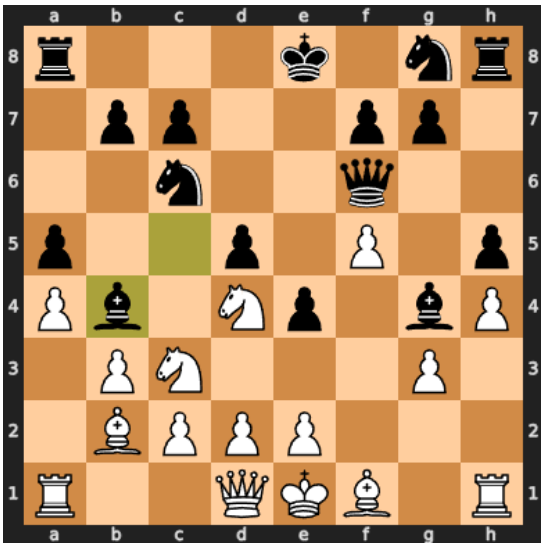

Figure 18: Legal Piece Moves Accuracy #100 - Legal Pawn Moves for White

In the position depicted in Figure 19, the objective is to identify all potential destination squares for the knight on 'd4'. The correct squares for the knight are 'e6', 'c6', 'f5', 'b5', 'f3', 'b3' and 'c2'. Of the theoretically possible moves for the knight, the GPT-2 models incorrectly predicted 'e2' (occupied by the white bishop) with an even higher probability than 'e6' (occupied by a black pawn). In contrast, the Mamba models with 71k and 350k parameters correctly identified all seven possible squares for the knight. The probability distribution from the Mamba exhibits a pronounced decline following the top seven predictions, effectively distinguishing the correct legal moves from the rest. Nevertheless, in all models the target squares 'b5' and 'b3' were consistently preferred.

Figure 19: Legal Piece Moves Accuracy #183 - Target Squares for Knight

Hard Positions Accuracy The accuracy for the metric "Hard Position Accuracy" demonstrates comparable accuracy across all three model sizes for both the GPT-2 and Mamba models. As illustrated in Figure 20, the GPT-2 and Mamba models exhibit consistent performance with only minor variations in accuracy. In particular, the GPT-2 350k model exhibit the highest degree of accuracy with 88% of positions correctly predicted. This is closely followed by the Mamba 350k with 87%. These results indicate that both architectures exhibit comparable performance in handling hard positions. To illustrate the distinctions in their performance, specific positions were examined where the models' predictions diverge.

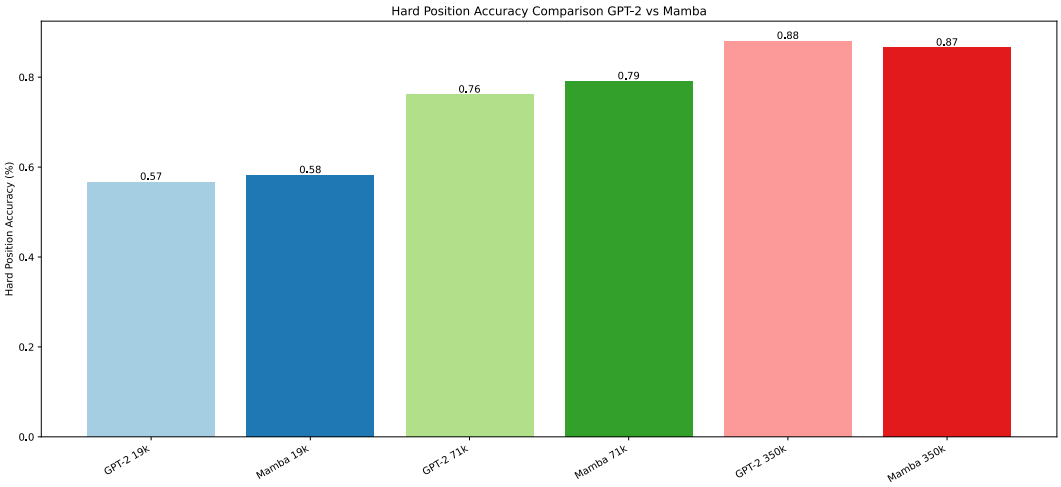

Figure 20: Comparison of Model Performance for GPT-2 and Mamba ("Hard Position Accuracy")

In the position shown in Figure 21, all GPT-2 models correctly predicted a legal rook move for Black. In contrast, the Mamba models attempted to move the queen, erroneously assuming it to be on 'g3'. This error persists despite the most recent moves being 'Rg1g3x', 'Ph4g3x', capturing White's queen, and 'Ph3g4x', in which the queen was recently captured on 'g3'. The Mamba models demonstrated a failure to recognise this capture and subsequent absence of the queen.

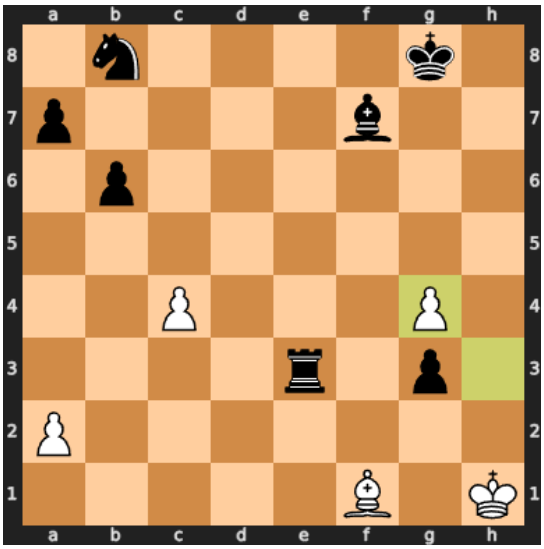

Figure 21: Hard Position Accuracy #13 - Endgame

Figure 22 illustrates a position where the only legal move for Black is to promote the pawn on 'd2' to a queen, rook, bishop, or knight. The GPT-2 models (19k, 71k, 350k) failed to recognise the need to promote the pawn on 'd2' and instead attempted to move the king. In contrast, the 71k and 350k Mamba models correctly identified the necessity for a pawn promotion and successfully promoted the pawn to a queen. This indicates that the Mamba models are more capable of handling specific rule-based scenarios such as pawn promotion, whereas the GPT-2 models demonstrated difficulty with this task.

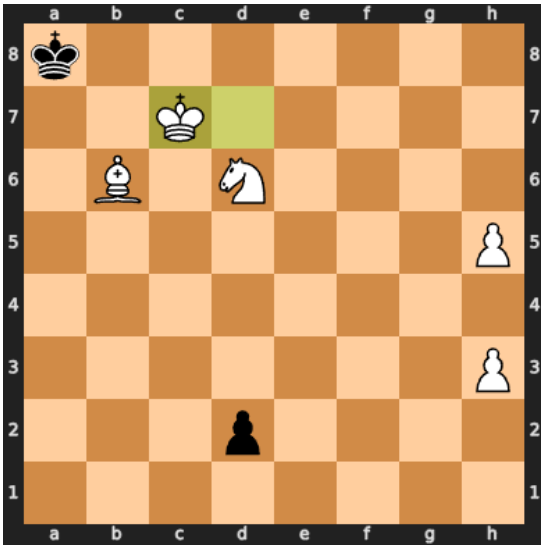

Figure 22: Hard Position #48 - Forced Pawn Promotion

Average Number of Correct Plies The most notable difference between Mamba and GPT-2 can be observed in Figure 23, which shows the results for the metric "Average Number of Correct Plies". This metric measures the consistency with which a model generates correct moves over the course of a game. The 350k Mamba model exhibits an average of 55.32 correct plies, while the GPT-2 350k model demonstrates an average of 51.86 plies.

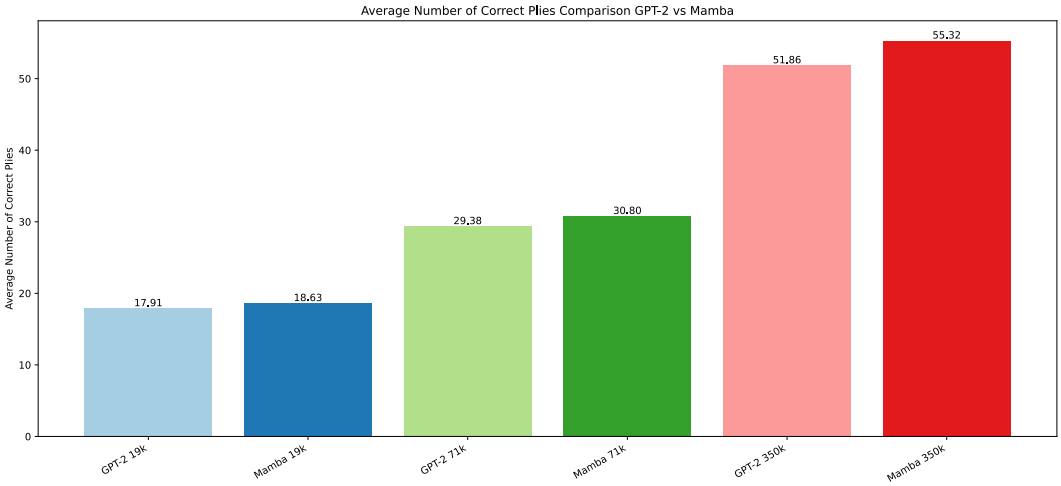

Figure 23: Comparison of Model Performance for GPT-2 and Mamba ("Average Number of Correct Plies")

Error Categories In order to gain a deeper understanding of the results of the generated sequences for the metric "Average Number of Correct Plies", it is necessary to analyse the specific categories that lead to an error in more detail.

The error categories are detected hierarchically in the following order: "Syntax", "Piece Logic", "Path Obstruction", "Pseudolegal", "Indicator Error", "No Error". This means that the first error in the hierarchy is identified and categorised, with only a single error category being attributed to each sequence. The error categories are defined as follows:

- **Syntax**: Errors in the basic syntactical structure of the move notation.
- **Piece Logic**: Errors where the move violates the logical movement of the chess pieces.
- **Path Obstruction**: Errors where an otherwise legal move is attempted but there is a piece blocking the path.
- **Pseudolegal**: Moves that are logically possible but violate chess rules (e.g., moving into check).
- **Indicator Error**: Errors in the indicators of the move (e.g., missing or incorrect check/checkmate indicators).
- **No Error**: No errors detected, indicating that the game has ended with a valid move.[13]
- **Max Length**: The sequence reaches the maximum number of plies without errors.

Table 4 presents the frequencies of errors across all sequences. The Mamba model exhibits a significantly lower error rate than the GPT-2 model in the categories "Syntax", "Piece Logic", and "Path Obstruction". Conversely, it generates a higher number of errors in the categories "Pseudolegal" and "Indicator Error". Additionally, the model generates more game sequences without any errors, either ending the game before the maximum number of plies ("No Error") or reaching the maximum number of 125 plies ("Max Length").

[13] In this case, the number of plies for calculating the metric "Average Number of Correct Plies" has been set to 125, which is equal to "Max Length".

Table 4: Error Types (% of Total Errors) for GPT-2 and Mamba (350k)

Error Type	**% of total errors**	
	GPT-2	**Mamba**
Syntax Error	0.4	0.1
Piece Logic Error	4.1	1.5
Path Obstruction Error	39.5	28.6
Pseudolegal Error	28.5	33.9
Indicator Error	19.7	24.7
No Error	7.8	11.1
Max Length	0.0	0.1

The hierarchical ordering of the error categories indicates that as the model learns to correctly generate syntactically valid moves (referred to as "Syntax Error") in accordance with the rules governing the movement of pieces (referred to as "Piece Logic Error"), the subsequent categories of "Path Obstruction Error" and "Pseudolegal Error" become increasingly probable. As the models continue to generate valid moves in these two categories, the category of "Indicator Error" becomes more likely. The shift in distribution of the errors can thus be interpreted as the Mamba models having a better grasp on the rules of chess, which is also reflected in the result of the metric "Average Number of Correct Plies" mentioned previously, which outperforms the GPT-2 models on average by approximately 4%.

Figure 24 illustrates the distribution of error frequencies for the GPT-2 model across different buckets of ten plies. The data shows how the errors are distributed throughout the course of a game, with each bucket representing a segment of ten plies. The frequency of the error categories varies throughout the course of the game, indicating specific stages where the GPT-2 model encounters the greatest difficulties. Notably, "Path Obstruction Errors" are the most frequent, suggesting that the GPT-2 model faces significant challenges in maintaining accurate path calculations as the game progresses.

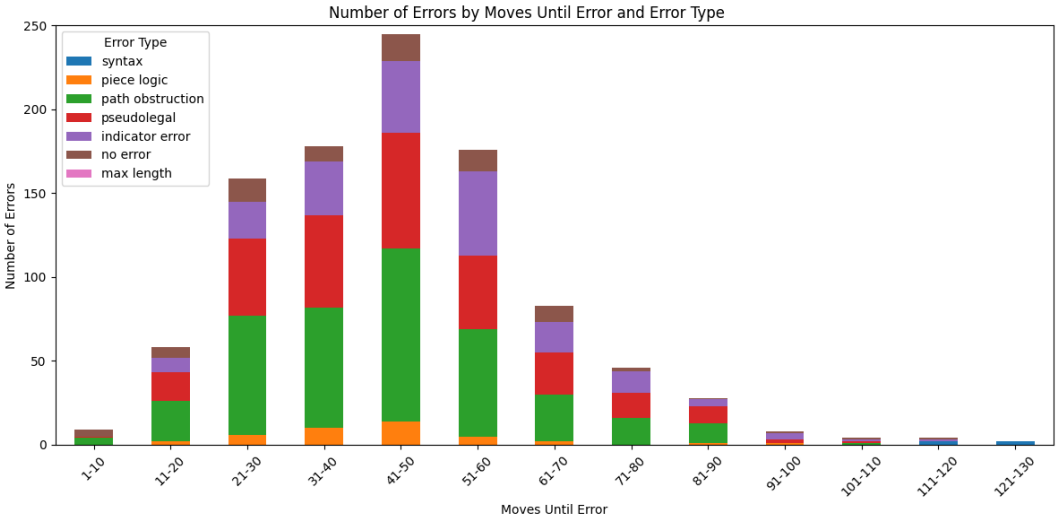

Figure 24: Error Frequencies of GPT-2 Model Trained on 350k Games

Figure 25 displays the error frequency distribution for the Mamba model, also trained on 350k games, across buckets of ten plies. This figure illustrates the Mamba model's

performance in generating accurate chess moves throughout the game. The Mamba model demonstrates improvements in reducing errors, particularly in the early and middle stages of the game. The frequency of "Path Obstruction" errors is lower than that of the GPT-2 model, indicating enhanced capabilities to analyse correct paths or to maintain the correct state of the game board. Furthermore, the Mamba model exhibits a reduction in "Piece Logic Errors", suggesting improved adherence to chess piece movement rules.

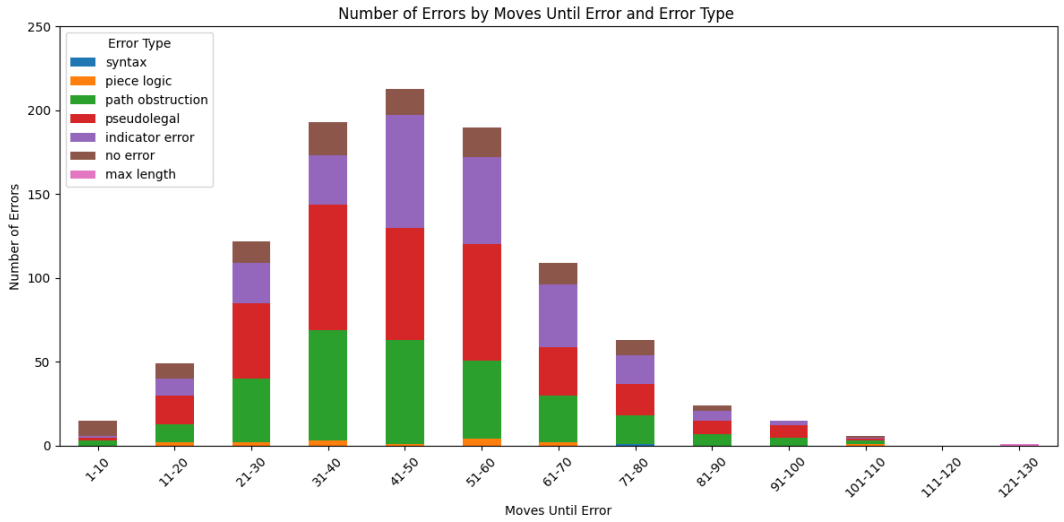

Figure 25: Error Frequencies of Mamba Model Trained on 350k Games

4.3.2 Conclusion

These experiments demonstrate that the Mamba architecture has a slight advantage over traditional GPT-2 models in chess move prediction tasks. Mamba is particularly effective at maintaining the complete state of the board over long sequences, rendering it more effective at generating accurate moves in extended games.

In the metric "Legal Piece Moves Accuracy", Mamba models demonstrate superior performance compared to GPT-2 models trained on larger datasets (350k games), suggesting enhanced scalability with increasing data. However, the smaller Mamba models exhibit greater difficulty in identifying all correct target squares. With regard to the handling hard positions, both architectures exhibit comparable performance. The Mamba 350k model exhibits a performance that is nearly equivalent to that of the GPT-2 350k model in terms of accuracy.

Additionally, the Mamba model demonstrates a higher value in the metric "Average Number of Correct Plies", indicating greater consistency in move generation. An analysis of errors reveals that Mamba makes fewer errors in categories such as "Syntax Error", "Piece Logic Error" and "Path Obstruction Error", although it has a greater number of "Pseudolegal Error" and "Indicator Error" types.

Overall, while the Mamba architecture shows some advantages, particularly in handling long sequences and maintaining game state accuracy, its performance is only slightly better than that of the GPT-2 models.

4.4 Fine-Tuning for Strategy

The central tenet of this experiment is to ascertain whether a chess model can develop a more sophisticated strategic understanding when fine-tuned on games of superior

quality, as gauged by Elo ratings. Elo ratings represent a standardised measure of a player's skill level (for further insights into the Elo rating system, please refer to Section 2.2). It is hypothesised that training a model on games played by higher-rated players will enhance its strategic proficiency.

The strategic proficiency of the fine-tuned models was assessed using two primary metrics: Average Centipawn Loss (ACPL) and "Survival Rate". For the evaluation of the models, a low temperature setting (0.1) was used for sampling, which causes the model to generate moves that are very close to the distribution of the training data.

The GPT-2 model trained on 350,000 games was fine-tuned using data from chess games filtered by Elo ratings. Different models were fine-tuned on two distinct Elo level ranges ("Low": 0-1000 Elo, "High": 2000+ Elo) and three different dataset sizes (10k, 30k, and 98k) in order to observe the effects of both the quality and quantity of training data.

The objective of this experiment is to determine whether models trained on higher-quality games (as indicated by higher Elo ratings) demonstrate improved strategic play. Additionally, the impact of varying the dataset sizes on the models' strategic learning capabilities will be explored.

4.4.1 Metrics for Strategy

Survival Rate against Stockfish The metric "Survival Rate" quantifies the number of plies a model can play against *Stockfish*[14] (limited in search depth to 0.1 seconds per move) before the game ends. Two conditions may result in the termination of a game: Firstly, the model fails to produce a legal move[15], and secondly, the model loses the game. In order to calculate the final value for the metric "Survival Rate", the number of plies executed by the model before meeting either of these conditions was recorded and averaged over 100 games. The average number of games that ended in checkmate was 67.9%[16].

The underlying concept of this metric is that a model with a higher estimated Elo rating will be able to sustain longer games against *Stockfish* before being defeated. Consequently, a higher survival rate would suggest that the model is more strategically proficient. This methodology for evaluating strategic proficiency is comparable with the approach used in Deleo et al.[2].

Average Centipawn Loss The ACPL metric is employed to quantify the discrepancy between the moves selected by the model and the optimal moves as determined by a chess engine. Lower ACPL values indicate superior strategic play, as the model's moves are more closely aligned with the optimal moves.

In these experiments, the ACPL metric was implemented by calculating the centipawn loss for each move in each of the 100 games played against *Stockfish* (Version 16.1). The centipawn loss for a move is defined as the difference in evaluation (measured in hundredths of a pawn) between the model's chosen move and the best move according to *Stockfish*. In order to ensure the accuracy of the results, mate scores were excluded from the calculations, as they can disproportionately skew the average. For a more

[14] `https://stockfishchess.org/`

[15] The models were given 15 attempts to generate a valid move.

[16] Analysing the following outcomes only on games ending in checkmate made no difference to the results.

detailed description of the theoretical foundations of the ACPL metric, please refer to Section 2.2.

4.4.2 Results

Average Centipawn Loss The results for the ACPL metric are presented in Table 5. The base model had an ACPL of 7.20 as White and 8.62 as Black.

For the models fine-tuned on low Elo games, it was expected that ACPL would increase. However, the results do not consistently show this trend. When the model plays as White, ACPL only increases for the dataset containing 98k games, but not for the two smaller ones. When the model plays as Black, ACPL generally increases, reaching the highest value when fine-tuned on the dataset containing 30k games.

For the models fine-tuned on high Elo games, it was expected that ACPL would decrease. This was the case for the 10k and 30k datasets when the model played as White, and for the 10k and 98k dataset when the model played as Black.

These results do not indicate that fine-tuning on higher Elo games generally led to a decrease in ACPL, which would suggest an improvement in strategic play. The data do not show a clear trend in the expected direction, so the results are not conclusive.

Table 5: Results for Fine-Tuning on Elo-Levels (ACPL)

	White	**Black**
Base Model	7.20	8.62
Low-Elo Fine-Tuning		
10k	6.83	9.42
30k	6.79	10.20
98k	7.23	9.54
High-Elo Fine-Tuning		
10k	7.14	8.28
30k	6.45	8.66
98k	7.44	7.56

Survival Rate against Stockfish The results for the metric "Survival Rate" are presented in Table 6. The base model exhibited a survival rate of 47.72 plies as White and 37.98 plies as Black.

For the models fine-tuned on games played by players with an Elo rating of 1000 or lower, it was anticipated that the survival rate would decline. When the model played as White, this was the case for the datasets containing 10k and 98k games respectively, but not for the one containing 30k games. When the model played as Black, the Survival Rate showed an increase in comparison to the base model for all datasets.

For the models fine-tuned on games played by players with an Elo rating of 2000 or higher, the survival rate was expected to increase. When the model played as White, all datasets exhibited a higher survival rate than the base model, with the 30k dataset demonstrating the highest value of 51.54. Conversely, when the model played as Black, the survival rate increased for all datasets, with the exception of the smallest one containing 10k games. The 98k dataset exhibited the highest increase with a value of 44.94. The longest game against *Stockfish* was played by the model trained on 30k games, containing 101 plies.

Similarly to the metric ACPL, these results did not conclusively indicate that fine-tuning on higher Elo games generally increased the survival rate, which would indicate better strategic endurance.

Table 6: Results for Fine-Tuning on Elo-Levels ("Survival Rate")

	White	**Black**
Base Model	47.72	37.98
Low-Elo Fine-Tuning		
10k	45.90	38.72
30k	48.36	39.44
98k	43.88	38.84
High-Elo Fine-Tuning		
10k	49.24	39.70
30k	51.54	39.98
98k	48.70	44.94

4.4.3 Conclusion

The results of fine-tuning the 350k GPT-2 model on datasets filtered by Elo ratings have provided insights into the effectiveness of different strategies for enhancing strategic gameplay. Several key findings have emerged from this experiment:

The results did not consistently show that fine-tuning on higher Elo games led to significant improvements in the models' strategic proficiency, as measured by ACPL and survival rate against *Stockfish*. The anticipated trend of lower ACPL and higher survival rate for models fine-tuned on high Elo games was not clearly observed. This suggests that merely fine-tuning on high Elo games is not sufficient to significantly enhance strategic understanding.

One potential explanation for these findings is the inherent complexity of strategic play, which may not be adequately captured by statistical similarities to high-level moves. It appears that simply playing moves that resemble those of high-Elo players does not necessarily result in superior performance in evaluation metrics. This highlights that strategy in chess involves more than just replicating moves; it requires a deeper understanding of the game state and context.

Pre-training on Elo-filtered datasets from the outset, rather than fine-tuning, might offer a more effective approach. Starting with a foundation built on high-quality games could help models develop a more nuanced understanding of strategic elements from the outset.

5 Conclusion and Prospects for Further Research

5.1 Conclusion

The experiments evaluated various chess model architectures and notations, focusing on the comparison between xLAN and xLAN+, the two architectures Mamba and GPT-2, and fine-tuning on games filtered by Elo rating. The results revealed several key insights into their performance and potential for improvement.

The xLAN+ notation, which includes a `move_status` token for captures, checks, and checkmates, significantly outperformed the standard xLAN notation. Models trained with xLAN+ generated longer sequences of correct moves, as evidenced by the "Average Number of Correct Plies" metric. The "Legal Piece Moves Accuracy" and "Hard Position Accuracy" metrics also demonstrated improvements, although to a lesser extent. Variants xLANchk and xLANcap corroborated these findings, with xLANcap excelling in generating correct move sequences and xLANchk in handling hard positions.

The Mamba architecture demonstrated a slight advantage over GPT-2 models, particularly in maintaining the complete state of the board over long sequences. This was reflected in the "Legal Piece Moves Accuracy" and "Average Number of Correct Plies" metrics. The Mamba models made fewer syntax and piece logic errors but had more pseudolegal and indicator errors. Despite these differences, Mamba's overall performance was marginally superior to that of GPT-2, particularly in the handling of long sequences and the maintenance of game state accuracy.

The results of the experiments indicated that fine-tuning on datasets filtered by Elo ratings did not consistently enhance strategic proficiency. This was evidenced by the results for ACPL and the survival rate against *Stockfish*. The anticipated improvements from fine-tuning on high Elo games were not observed, suggesting that strategic gameplay involves more than replicating high-level moves. Consequently, it can be concluded that pre-training on high-quality datasets might be a more effective approach for developing strategic understanding from the outset.

The experiments demonstrate the potential and limitations of different strategies for improving chess model performance. The xLAN+ notation offers significant benefits for generating valid moves and handling complex scenarios. The Mamba architecture provides slight improvements in maintaining game state accuracy over long sequences. Fine-tuning alone is insufficient for enhancing strategic understanding, suggesting the need for a more foundational approach in future research. These findings provide a foundation for further model refinements and the development of new evaluation metrics that can more effectively capture the strategic elements of gameplay.

5.2 Outlook

This section outlines potential avenues for future research and improvements, based on the findings of this project.

Model Evaluation against Stockfish One promising avenue for future work is to let the trained models play against different levels of *Stockfish*. This approach, inspired by Karvonen[5], would help evaluate the models' capabilities in real-time gameplay against progressively stronger opponents. By analysing performance across varying difficulty levels, deeper insights into the strengths and weaknesses of the models could be gained.

Pre-Training on Elo-Filtered Datasets An alternative approach to fine-tuning models on datasets filtered by Elo rating could be to directly pre-train the models on these datasets. This strategy might help the models to develop a deeper and more nuanced understanding of chess strategies associated with different skill levels, potentially leading to better overall performance in generating strategic moves.

Refining Inference with Top-p and Top-k Sampling In order to enhance the quality of the moves generated by the models, inference techniques such as top-p sampling and top-k sampling could be employed. These methods regulate the diversity of the generated moves by limiting the sampling pool to the most probable options, which can enhance the reliability and strategic soundness of the moves.

Generating Moves from Different Opening Sequences Furthermore, evaluating the models' performance not only by generating move sequences from the initial position but also from various opening sequences could provide a more robust assessment. This method allows for consistent evaluation without the need to adjust temperature settings, thereby ensuring that the most accurate predictions of the model are always considered, enhancing the reliability of the evaluation process.

Introducing the "Move Status Accuracy" Metric A new metric, "Move Status Accuracy" could be incorporated into the evaluation framework. This metric would specifically assess the models' proficiency in generating valid tokens for `move_status`. By providing positions that require a specific `move_status` token, such as capture or checkmate indicators, the models' accuracy in predicting these crucial aspects of the game can be evaluated.

"Winner Prediction Accuracy" Another potential metric is "Winner Prediction Accuracy," where the models are presented with board positions where a checkmate or a draw has occurred. The objective is to predict the correct game outcome (win, loss, or draw) based on the final position. This would assist in determining the models' ability to recognize game-ending scenarios accurately and avoid suggesting unnecessary moves. The same authors have previously employed this methodology in [8] for the game "Connect 4".

6 Directories

6.1 References

[1] David Noever, Matt Ciolino, and Josh Kalin. "The Chess Transformer: Mastering Play using Generative Language Models". In: (2020). DOI: 10.48550/ARXIV.2008.04057. URL: https://arxiv.org/abs/2008.04057 (visited on 10/17/2023).

[2] Michael DeLeo and Erhan Guven. "Learning Chess with Language Models and Transformers". In: *Data Science and Machine Learning*. Academy and Industry Research Collaboration Center (AIRCC), Sept. 2022, pp. 179–190. ISBN: 978-1-925953-75-6. DOI: 10.5121/csit.2022.121515. URL: https://aircconline.com/csit/papers/vol12/csit121515.pdf (visited on 10/17/2023).

[3] Andreas 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. ISBN: 978-954-452-072-4. DOI: 10.26615/978-954-452-072-4_153. URL: https://acl-bg.org/proceedings/2021/RANLP%202021/pdf/2021.ranlp-1.153.pdf (visited on 10/17/2023).

[4] Anian Ruoss et al. *Grandmaster-Level Chess Without Search*. Feb. 2024. arXiv: 2402.04494. URL: https://arxiv.org/abs/2402.04494 (visited on 04/21/2024).

[5] Adam Karvonen. *Emergent World Models and Latent Variable Estimation in Chess-Playing Language Models*. Mar. 2024. arXiv: 2403.15498. URL: https://arxiv.org/abs/2403.15498 (visited on 03/29/2024).

[6] Alec Radford et al. *Language Models are Unsupervised Multitask Learners*. Feb. 2019. URL: https://api.semanticscholar.org/CorpusID:160025533 (visited on 11/14/2023).

[7] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. English. Dec. 2023. arXiv: 2312.00752. URL: https://arxiv.org/abs/2312.00752.

[8] Lars Schmid and Jerome Maag. "Language Models Explore the Linguistics of Chess". Projektarbeit, Zürcher Hochschule für Angewandte Wissenschaften (ZHAW). Dec. 2023.

[9] Badri Narayana Patro and Vijay Srinivas Agneeswaran. *Mamba-360: Survey of State Space Models as Transformer Alternative for Long Sequence Modelling: Methods, Applications, and Challenges*. Apr. 2024. DOI: 10.48550/arXiv.2404.16112. URL: http://arxiv.org/abs/2404.16112 (visited on 06/07/2024).

[10] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. English. June 2021. arXiv: 2106.09685. URL: https://arxiv.org/abs/2106.09685.

[11] Arpad E. Elo. *The Rating of Chessplayers, Past and Present*. 2nd ed. Second Edition published under the CACDEC Program of FIDE. New York: Arco Publishing, Inc., 1986. ISBN: 0-668-04721-6.

[12] Yuntao Bai et al. *Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback*. Apr. 2022. URL: http://arxiv.org/abs/2204.05862 (visited on 05/12/2024).

[13] Leszek Szczecinski and Aymen Djebbi. *Understanding and Pushing the Limits of the Elo Rating Algorithm*. 2019. arXiv: 1910.06081. URL: https://arxiv.org/abs/1910.06081v1 (visited on 04/01/2024).

[14] Enzo Leon Solis Gonzalez. *Data Science and Chess: Centipawn Loss Elo Correlation*. English. June 2024. URL: https://medium.com/@enzo.leon/data-science-and-chess-centipawn-loss-elo-correlation-e06089efd8b8 (visited on 04/02/2024).

[15] Matej Guid and Ivan Bratko. "Computer Analysis of World Chess Champions". In: *J. Int. Comput. Games Assoc.* 29.2 (2006), pp. 65–73.

[16] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. Dec. 2015. URL: http://arxiv.org/abs/1511.08458 (visited on 12/12/2023).

[17] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. Nov. 2019. URL: http://arxiv.org/abs/1912.05911 (visited on 12/12/2023).

[18] Ashish Vaswani et al. *Attention Is All You Need*. Aug. 2017. URL: http://arxiv.org/abs/1706.03762 (visited on 12/11/2023).

[19] Josh Achiam et al. *GPT-4 Technical Report*. Mar. 2023. URL: http://arxiv.org/abs/2303.08774 (visited on 12/12/2023).

[20] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. May 2019. URL: http://arxiv.org/abs/1810.04805 (visited on 10/17/2023).

[21] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. en. Nov. 2018. URL: https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

[22] Abubakar Abid et al. *HuggingFace NLP Course*. URL: https://huggingface.co/learn/nlp-course/chapter1/1 (visited on 12/11/2023).

[23] Rick Merritt. *What Is a Transformer Model?* en. Mar. 2022. URL: https://blogs.nvidia.com/blog/what-is-a-transformer-model/ (visited on 12/11/2023).

[24] Simon J. D. Prince. *Understanding deep learning*. Cambridge, Massachusetts: The MIT Press, Nov. 2023. ISBN: 978-0-262-04864-4.

[25] Shubham Toshniwal et al. *Chess as a Testbed for Language Model State Tracking*. May 2022. URL: http://arxiv.org/abs/2102.13249v2 (visited on 12/10/2023).

6.2 List of Figures

6.3 List of Tables

6.4 Index

7 Appendix

7.1 Extracts from Previous Project

For the sake of convenience, the following subchapters contain verbatim extracts from previous work by the same authors [8], which are relevant for this project.

7.1.1 Transformer Architecture

Transformer models represent a significant departure from traditional neural network architectures like CNNs [16] and RNNs [17]. Introduced in 2017 [18], 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.

Transformer models consist of two main components: the encoder and the decoder. The encoder analyzes 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 26 illustrates the general architecture of a transformer model.

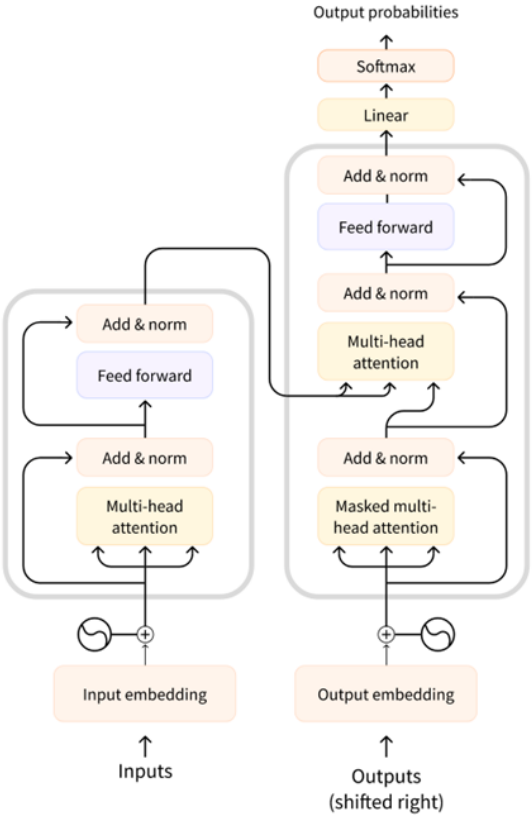

Figure 26: Transformer Model Architecture [19]

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) [20], 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 [21][18], are optimised 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: [22][23][24].

For the chess application developed in this project, a decoder-only model was employed, optimised 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.

7.1.2 Chess Notation

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.

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.

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.

7.1.3 Data Collection

The dataset for training the chess models was sourced from online games, played on the *Lichess* platform, specifically from games played in September 2023. Accessible through the *Lichess* database, 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 format of this dataset includes detailed information such as move sequences, links to online games, dates, times, player names, Elo ratings, openings played, time controls, and game termination modes (e.g., time forfeit or conventional chess game endings like checkmate).

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 library `python-chess`, the dataset was converted into xLAN format, though some viable games were lost due to conversion limitations. The resulting dataset, after eliminating duplicates, comprised 23,521,034 games.

Preparation of the Tokenised Dataset The dataset was tokenised for model training, including games up to 601 plies. Given the model's maximum sequence length of 512 tokens, games exceeding 500 tokens (166 plies considering that one ply consists of three tokens) 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 formed 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.

Comparison of Datasets Table 7 and Table 8 provide comprehensive overviews of the datasets. Table 7 describes dataset sizes and formats, while Table 8 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 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 7: Overview of the Datasets Used to Train the Chess Models

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

Table 9 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 "Tokenised", but their frequency diminishes in smaller datasets. For instance, the sequence `Pe2e4 Pe7e5 Ng1f3` appears in every sixth game of the tokenised 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.

Table 8: 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
Tokenised	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

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 9: Occurrence of Opening Sequences in Percentage (%)

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

7.1.4 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.

Chess Data Tokenisation The process of tokenisation plays a significant role in the pre-training of LLMs. Tokenisation 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 tokenisation was employed, tailored specifically for the domain of chess move generation. This customized approach, in contrast to standard tokenisers 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 xLAN. 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 tokenisation 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 tokenisation approach offers several benefits. It ensures the model's outputs adhere strictly to the defined chess vocabulary, crucial for accurately assessing its performance. 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.

7.1.5 Model Performance Metrics

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[17] 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.

[17] These error categories have been slightly adapted from the categories used in [25].

- **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.[18]

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 generalise, rather than solely relying on memorisation 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.
- **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.

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. 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}`.

7.2 Links

Important Links:

- GitHub Repository

[18] For the evaluations of the experiments in this research, this number has been increased to 125.

- Hugging Face Chess Models and Dataset Repository

7.3 GitHub Readme

Introduction This repository contains the research work and codebase for training LLMs solely on chess game sequences. The goal is to train an LLM to understand and replicate the rules of chess, make legal moves, and predict chess game outcomes without explicit rule-based programming. Two architectures were employed: a Transformer model based on OpenAI's GPT-2 configuration and a Mamba model with comparable dimensions.

Papers The following papers have been written:

- "Language Models Explore the Linguistics of Chess" (Lars Schmid, Jerome Maag, December 2023). GitHub Repository: Project Thesis Branch
- "Optimizing Language Models for Chess: The Impact of Custom Notation and Elo-Based Fine-Tuning" (Lars Schmid, Jerome Maag, June 2024)

The datasets used to train the models are encoded in two different chess notations specifically designed for this project xLAN/xLAN+.

Models and datasets can be found here: Link to Hugging Face Repository.

Quick Start This guide will help you get started with the project. Follow these steps to set up the environment and run your first example. This guide covers using the models (Transformer and Mamba) and training a Transformer model. To train the Mamba model, follow the installation guide on Mamba's GitHub repository.[19].

Prerequisites Ensure you have Python installed on your system. We used Python version 3.10.9. You can download it from Python's official website[20]. For training and efficient predictions, you should use CUDA. We used CUDA Version 12.1. For *PyTorch* to work with CUDA, you need to download the correct version for your machine from the *PyTorch* website[21].

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`.
   - Install the correct *PyTorch* version if you want to use CUDA (see Prerequisites).

[19] Training the Mamba model is currently only possible on Linux.
[20] `https://www.python.org/`
[21] `https://pytorch.org/`

Running Your First Example

1. Open a notebook:
    - 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.

- `analysis.ipynb`: Analysis of trained models and datasets, including visualizations and statistics.
- `dataPreProcessing.ipynb`: Conversion tools between different chess notations and dataset preparation.
- `evaluation.ipynb`: Evaluate models on our metrics and inspect the evaluation results.
- `finetune.ipynb`: Notebook for fine-tuning a model using LoRA.
- `huggingface.ipynb`: Use to upload and download models and datasets from HuggingFace.
- `train.ipynb`: Notebook for training and evaluating the chess model.
- `useModel.ipynb`: Interface to play chess against the model, for self-play, and to predict the next move from a given position.

Files This section includes scripts and files that provide specific functionalities or information necessary for the training and operation of the LLMs. 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.

- `notation.json`: Contains all training, prediction, and evaluation configurations for all notations.
- `xlan_tokens.json` and similar: Each notation has a corresponding `.json` file that includes the mappings of the chess notation into tokens.
- `pgn_to_xlan.py`: Converts PGN format files into the custom xLAN or xLAN+ format.
- `tokenizer.py`: Tokenizer that uses 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 correct plies, hard position accuracy, and legal piece move 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 for 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.

Notations

xLAN xLAN, an adaptation of LAN, 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}`.

xLAN+ xLAN+ is an extension to xLAN. In addition to the tokens used in xLAN, namely `{piece}`, `{start_square}`, and `{end_square}`, xLAN+ includes an additional `{move_status}` token (referred to as the indicator in code) at the end of each move. This additional token contains information about captures (`x`), checks (`+`), and checkmates (`#`).

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. Pg2g3 Pe7e6 2. Bf1g2 Qd8f6 3. Pf2f4 Bf8c5 4. Pe2e4
   Qf6d4 5. Pe4e5 Qd4f2 0-1}
```

- **xLAN+ Adaptation:**

```
1. Pg2g3- Pe7e6- 2. Bf1g2- Qd8f6- 3. Pf2f4- Bf8c5- 4.
   Pe2e4- Qf6d4- 5. Pe4e5- Qd4f2# 0-1}
```

Special Moves in xLAN/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}`.
- **Combination in move_status:** `$` represents captures combined with a check, and `!` is used for captures combined with checkmate.

Tools and Resources Used

1. **Weights & Biases**[22]**:** A machine learning experiment tracking tool used for visualizing and comparing our models' training progress.
2. **Python Chess**[23]**:** A chess library for Python, providing essential functionalities for chess manipulations and move generation.
3. **Lichess Database**[24]**:** Source of comprehensive chess game data, utilized for training and testing the LLMs.
4. **Hugging Face**[25]**:** 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: maag.jerome@gmail.com / lars.schmid@gmx.ch

7.4 Project Management

The project was organised using the software tool Jira[26], which facilitated the definition and management of workflow stages according to a Kanban approach. This methodology ensured that each phase of development received adequate focus and resources. The primary tool for managing the project was a Kanban board, which assisted in the prioritisation and tracking of identified action items.

Figure 27 presents the timeline of key project milestones, indicating their anticipated start and end dates.

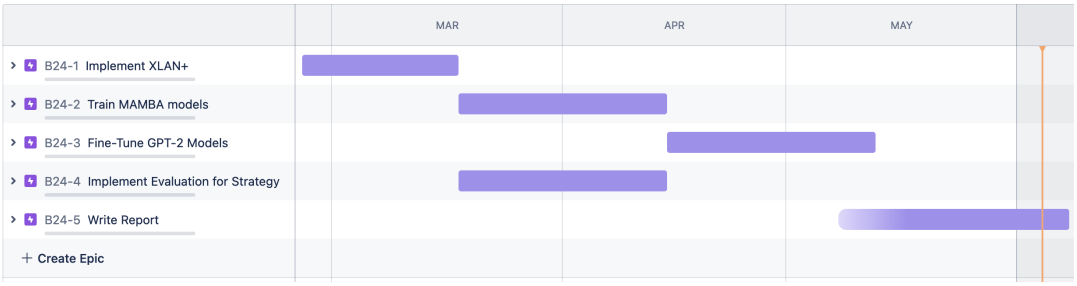

Figure 27: Project Timeline

[22] https://wandb.ai/
[23] https://python-chess.readthedocs.io/
[24] https://database.lichess.org/
[25] https://huggingface.co/
[26] https://www.atlassian.com/software/jira/

7.5 Use of Generative AI

Generative AI systems and AI tools were employed in various stages of this project. Specifically, ChatGPT[27], Llama 3[28] and DeepL Write[29] were employed for the generation of texts and program code, as well as for the improvement of wording and proofreading. The software GitHub Copilot[30] was employed to generate parts of the source code. Finally, the image generation software DALL-E 3[31], Midjourney[32], and Stable Diffusion[33] were employed to generate the image on the title page.

[27] https://chatgpt.com/
[28] https://llama.meta.com/llama3/
[29] https://www.deepl.com/en/write
[30] https://github.com/features/copilot
[31] https://openai.com/index/dall-e-3/
[32] https://www.midjourney.com/
[33] https://stability.ai/stable-image