Next-Move Prediction in Chess Using Sequence Models

A Comparative Study of LSTM and Transformer Architectures

Project Report

Sanjana S. Joshi
MS2405
Machine Learning 2, Semester 3
Department of Scientific Computing, Modeling and Simulation
Savitribai Phule Pune University

Supervisor: Dr. Kaveri Kale

November, 2025

Abstract

Predicting the next move in a chess game is a challenging sequential decision-making problem characterized by a large state space, long-term dependencies, and strict legality constraints. In this project, chess move prediction is formulated as a sequence modeling and multi-class classification task, where a model predicts the next move given a sequence of previous moves expressed in Standard Algebraic Notation (SAN).

Two deep learning architectures are implemented and compared: a Long Short-Term Memory (LSTM) network as a baseline recurrent model, and a Transformer model utilizing self-attention. Both models are trained using PyTorch on a dataset of chess games, with a vocabulary size of 3672 unique moves. The models are evaluated using training and test loss, top-1 and top-5 prediction accuracy, inference time, and qualitative analysis of predicted moves.

Experimental results indicate that while the Transformer model offers faster inference and improved handling of long-range dependencies compared to the LSTM, both models struggle with generating legally valid chess moves due to the absence of explicit rule constraints. This study highlights the strengths and limitations of sequence-based neural models in structured game domains and motivates the integration of domain knowledge in future work.

Chapter 1

Introduction

Chess has historically served as a central benchmark in artificial intelligence research due to its deterministic rules, immense search space, and strategic complexity. Early successes in chess AI relied on brute-force search combined with handcrafted evaluation functions, culminating in classical engines such as Stockfish.

Parallel to advances in game-playing AI, deep learning techniques for sequence modeling have achieved remarkable success in natural language processing tasks, including machine translation, text generation, and speech recognition. Models such as Long Short-Term Memory (LSTM) networks and Transformers are capable of learning long-range dependencies within sequences and modeling complex probabilistic relationships between tokens.

A chess game can naturally be represented as a sequence of moves, where each move depends not only on the immediate board state but also on strategic patterns established earlier in the game. This resemblance to language modeling motivates the application of sequence models to chess move prediction. In this context, moves written in Standard Algebraic Notation (SAN) are treated as tokens, and a game is viewed as a sentence composed of these tokens.

This project explores the application of deep learning-based sequence models to the task of next-move prediction in chess without explicitly encoding chess rules or board representations. Instead, the models are trained purely on historical move sequences, allowing them to implicitly learn statistical regularities present in real games.

Two architectures are considered in this study. The first is an LSTM-based model, which serves as a baseline due to its widespread use in sequential data modeling. The second is a Transformer-based model that employs self-attention mechanisms to capture global dependencies within a sequence. Both models are implemented using PyTorch and trained under similar experimental settings to enable a fair comparison.

The goal of this project is not to compete with traditional chess engines, but rather to analyze the effectiveness, efficiency, and limitations of generic sequence models when applied to a highly structured and rule-constrained domain such as chess.

Chapter 2

Problem Statement

The objective of this project is to predict the next move in a chess game given a sequence of previous moves. Let a chess game be represented as an ordered sequence of moves in Standard Algebraic Notation (SAN):

$$M = (m_1, m_2, m_3, \dots, m_t),$$

where each m_i is a token drawn from a fixed vocabulary V of size 3672.

Given the sequence $(m_1, m_2, \dots, m_t)$, the task is to predict the next move m_{t+1}. This is formulated as a multi-class classification problem, where the model outputs a probability distribution over all possible moves in the vocabulary:

$$P(m_{t+1} \mid m_1, m_2, \dots, m_t).$$

The predicted move is obtained by selecting the token with the highest probability. During training, sequences are padded to a fixed length, and padding tokens are masked to avoid influencing the model's predictions.

A key challenge in this formulation arises from the fact that not all moves in the vocabulary are legally valid in a given board position. However, in this project, no explicit chess rules or board state information are provided to the models. As a result, the models are trained to optimize predictive likelihood rather than move legality.

The problem setting intentionally emphasizes statistical learning from data over rule-based reasoning, allowing an analysis of how far generic sequence models can perform in a structured domain without domain-specific constraints.

Chapter 3

Motivation

Chess has long been regarded as a benchmark domain for evaluating sequential decision-making and pattern recognition systems. Unlike many natural language tasks, chess move prediction requires understanding long-range dependencies, structured rules, and highly constrained legal actions. These properties make chess an ideal testbed for studying the strengths and limitations of different sequence modeling architectures.

Recurrent neural networks, particularly Long Short-Term Memory (LSTM) networks, have historically been used for modeling sequential data due to their ability to maintain temporal state. However, LSTMs process sequences sequentially, which limits parallelization and makes learning long-term dependencies challenging as sequence length increases.

Transformer-based architectures, which rely on self-attention mechanisms, have recently become the dominant approach in sequence modeling tasks such as natural language processing. By enabling direct interaction between all tokens in a sequence, Transformers can capture long-range dependencies more effectively and allow for parallel computation. Despite their success, Transformers are computationally more expensive and require careful evaluation in resource-limited settings.

The motivation behind this project is to empirically compare a traditional LSTM-based model and a Transformer-based model for the task of chess move prediction. By using the same dataset, preprocessing pipeline, and evaluation metrics, this study aims to analyze the trade-offs between model accuracy, training time, inference speed, and architectural complexity. Such a comparison is valuable for understanding when more complex architectures like Transformers are justified over simpler recurrent models, especially in structured domains such as chess.

Chapter 4

Goals

The primary goal of this project is to perform a comparative analysis of two sequence modeling architectures - a baseline LSTM model and a Transformer-based model - for predicting the next move in a chess game given a sequence of prior moves.

The specific objectives of this project are as follows:

- To construct a clean and structured dataset of chess games by parsing Portable Game Notation (PGN) files and extracting move sequences in Standard Algebraic Notation (SAN).
- To design a preprocessing pipeline that converts variable-length move sequences into numerical representations suitable for neural network training, including vocabulary construction and sequence padding.
- To implement a baseline LSTM-based model for next-move prediction using PyTorch, serving as a reference point for performance comparison.
- To implement a Transformer-based model using PyTorch's optimized Transformer encoder modules, enabling efficient training and inference [6].
- To evaluate and compare both models using quantitative metrics such as training loss, test accuracy, training time, and inference time.
- To analyze the strengths and weaknesses of each model through qualitative error analysis, including cases where predicted moves are illegal or strategically implausible.
- To visualize and demonstrate model predictions by mapping predicted moves onto a chessboard representation for interpretability.

Ultimately, this project aims to provide insights into the practical trade-offs between recurrent and attention-based models for structured sequence prediction tasks, contributing to informed model selection in similar problem domains.

Chapter 5

Literature Survey

Sequence modeling has been a central problem in machine learning, with applications ranging from natural language processing to game playing. This section reviews prior work relevant to chess move prediction, recurrent neural networks, and Transformer-based architectures.

5.1 Neural Networks for Chess and Board Games

Early work in computer chess relied on handcrafted evaluation functions and tree search algorithms. With the advent of deep learning, neural networks began to replace manually engineered features. Silver et al. introduced AlphaZero [7], which combined deep neural networks with Monte Carlo Tree Search to achieve superhuman performance in chess, shogi, and Go. While highly effective, AlphaZero requires enormous computational resources and is unsuitable for lightweight sequence prediction tasks.

Other studies have explored chess move prediction as a supervised learning problem. McIlroy-Young et al. [5] demonstrated that neural networks trained on human games can learn meaningful representations of player style and skill, suggesting that move sequences alone contain rich information. These works motivate the use of chess as a structured sequential dataset.

5.2 Recurrent Neural Networks and LSTM Models

Recurrent Neural Networks (RNNs) were among the first neural architectures designed to handle sequential data. However, standard RNNs suffer from vanishing and exploding gradient problems when modeling long sequences. Hochreiter and Schmidhuber proposed the Long Short-Term Memory (LSTM) architecture [3] to address this limitation by introducing gated mechanisms for controlling information flow.

LSTM networks have been widely applied to sequence prediction tasks, including language modeling and time-series forecasting. In the context of games, LSTMs have been used to model move sequences due to their ability to capture temporal dependencies. However, LSTMs process input sequences sequentially, which limits parallelization and increases training time for long sequences such as full chess games.

5.3 Transformer Architecture and Self-Attention

The Transformer architecture was introduced by Vaswani et al. [8] and marked a significant shift in sequence modeling. Unlike recurrent models, Transformers rely entirely on self-attention mechanisms to model relationships between tokens, allowing direct access to all positions in a sequence regardless of distance.

Transformers have achieved state-of-the-art performance in natural language processing tasks and have been extended to various domains, including vision and reinforcement learning. Their ability to model long-range dependencies and support parallel computation makes them attractive for tasks involving long sequences, such as chess move prediction.

Recent work has applied Transformers to board games and structured decision-making problems. For example, Guez et al. [2] demonstrated that attention-based models can learn effective policies in complex environments. These findings suggest that Transformers may offer advantages over recurrent models in capturing global context within move sequences.

5.4 Comparative Studies of LSTM and Transformer Models

Several studies have compared LSTM and Transformer architectures across different sequence modeling tasks. Research has consistently shown that Transformers outperform LSTMs on tasks requiring long-context modeling, while LSTMs remain competitive on smaller datasets or shorter sequences. However, Transformers typically incur higher computational costs in terms of memory usage and training time.

In the context of this project, a direct comparison between an LSTM baseline and a Transformer model is particularly relevant. By controlling for dataset size, preprocessing steps, and evaluation metrics, this study aims to provide an empirical comparison focused on efficiency, predictive accuracy, and practical deployment considerations.

5.5 Summary

Prior literature indicates that both LSTM and Transformer architectures are suitable for sequence modeling tasks, but each exhibits distinct trade-offs. LSTMs offer simplicity and lower computational overhead, while Transformers provide superior capacity for modeling long-range dependencies. This project builds upon existing work by applying both architectures to the task of chess move prediction under a unified experimental framework.

Chapter 6

Methodology

6.1 Dataset Description

The dataset used in this project consists of publicly available chess games stored in Portable Game Notation (PGN) format. PGN is a standard textual representation for chess games that includes both metadata (such as player names, ratings, and game results) and the complete sequence of moves played.

A subset of games was extracted from a large online chess database to ensure feasibility of training within limited computational resources. Each game consists of a sequence of legal chess moves played by human players under standard tournament conditions.

Only games containing a minimum number of moves were retained to ensure meaningful learning of move sequences. Games with malformed or illegal move records were discarded during preprocessing.

Dataset Source: Lichess PGN rated games (https://database.lichess.org/) for September 2014 which contains 1,000,056 chess games (179 MB PGN file) and number of games used from the data were 10,000.

6.2 Data Preprocessing

The raw PGN data cannot be directly used for training neural networks. Therefore, a multi-step preprocessing pipeline was designed to transform raw chess games into numerical representations suitable for sequence modeling.

6.2.1 Parsing PGN Files

Each PGN file was parsed using the `python-chess` library. This library provides reliable utilities for reading PGN files and validating move legality [1]. For each game, the header information was read, followed by traversal of the main move line.

Games that could not be parsed correctly or contained illegal moves were skipped to maintain data consistency.

6.2.2 Extracting Move Sequences

For each valid game, the sequence of moves was extracted in Standard Algebraic Notation (SAN). SAN is a compact, human-readable representation that uniquely describes chess moves, including captures, promotions, and check or checkmate indicators.

Each game was represented as an ordered list: $(m_1, m_2, ..., m_n)$ where each m_i corresponds to a SAN move. The board state was updated incrementally to ensure that the SAN representation was generated correctly for each move.

6.2.3 Vocabulary Construction

All extracted move sequences were aggregated, and a vocabulary of unique SAN moves was constructed. Each unique move was assigned a unique integer identifier.

Special tokens were added to handle padding and unknown moves:

- `<PAD>` - used for sequence padding
- `<UNK>` - used for unseen or rare moves

The final vocabulary size was 3672 unique tokens, including special tokens. This vocabulary defines the output space for both models.

6.2.4 Sequence Encoding and Padding

Each game's move sequence was converted into a sequence of integer token IDs using the constructed vocabulary. To enable batch training, sequences of varying lengths were padded to a fixed maximum length using the `<PAD>` token.

Padding ensures that all sequences in a batch have the same length while preserving the original order of moves. Padding tokens were masked during training to prevent them from influencing the model's predictions.

6.2.5 Input - Target Pair Construction

To train the models for next-move prediction, each game sequence was transformed into multiple input-target pairs. Given a sequence of moves: $(m_1, m_2, ..., m_t)$ training samples were created as: $(m_1) \rightarrow m_2$, $(m_1, m_2) \rightarrow m_3$

This formulation allows the models to learn conditional probabilities of the next move given all previous moves.

6.2.6 Train - Test Split

The processed dataset was split into training (80%) and testing (20%) sets using random shuffling. The training set was used for parameter optimization, while the test set was reserved strictly for evaluation.

No separate validation set was used in this project due to computational constraints.

6.3 Model Architectures

Two sequence modeling architectures were implemented and compared: a baseline LSTM model and a Transformer-based model.

6.3.1 Baseline LSTM Model

The baseline model uses a Long Short-Term Memory (LSTM) network, which is well-suited for modeling sequential data with temporal dependencies.

The architecture consists of:

- An embedding layer that maps token IDs to dense vectors
- A single-layer LSTM that processes the embedded sequence
- A fully connected output layer that maps the final hidden state to vocabulary logits

Only the final hidden state of the LSTM is used to predict the next move. The model is trained using cross-entropy loss.

6.3.2 Transformer Model

The Transformer-based model employs a self-attention mechanism to model dependencies between all positions in the input sequence simultaneously.

The architecture includes:

- Token embeddings combined with sinusoidal positional encodings
- A stack of Transformer encoder layers
- Multi-head self-attention and feedforward sublayers
- A linear output layer projecting to the vocabulary size

Padding masks are used to prevent attention over padded tokens. The representation corresponding to the final non-padding token is used for next-move prediction.

6.4 Model Training

Both models were trained using the Adam optimizer and cross-entropy loss. Training was conducted for a fixed number of epochs on a GPU-enabled environment.

The same batch size and training data were used for both models to ensure a fair comparison. Training time per model was recorded to assess computational efficiency.

6.5 Training Setup

Both models were trained using supervised learning for next-move prediction. The training setup was kept consistent across models to ensure a fair comparison.

6.5.1 Optimization and Loss Function

Cross-entropy loss was used as the objective function, treating next-move prediction as a multi-class classification problem over the move vocabulary. The Adam optimizer was employed due to its adaptive learning rate properties.

6.5.2 Hyperparameters

The key hyperparameters used during training are summarized below:

- Batch size: 64
- Number of epochs: 5
- Optimizer: Adam
- LSTM learning rate: 1×10^{-3}
- Transformer learning rate: 1×10^{-4}

The Transformer model uses a smaller learning rate to ensure stable optimization, as attention-based models are more sensitive to learning rate selection.

6.5.3 Hardware and Environment

All experiments were conducted using Google Colab with GPU acceleration. Both models were trained on the same hardware environment to ensure comparability. Training times were recorded empirically.

6.5.4 Padding and Masking

Input sequences were padded to a fixed maximum length using a special padding token. For the Transformer model, padding masks were applied to prevent attention over padded positions. Padding tokens were ignored during loss computation where applicable.

6.6 Evaluation Metrics

The models were evaluated using the following metrics:

- **Training Loss**: Average cross-entropy loss over the training set
- **Test Loss**: Cross-entropy loss computed on unseen test data
- **Top-1 Accuracy**: Percentage of cases where the predicted move matches the ground-truth next move
- **Top-5 Accuracy**: Measures how often the correct label is present within the model's top five most probable predictions
- **Inference Time**: Average time required to generate a prediction for a single input sequence

6.7 Visualization and Qualitative Analysis

Training loss curves were plotted for both models to analyze convergence behavior. Additionally, predicted moves were qualitatively inspected by decoding model outputs back to SAN notation and visualizing them on a chessboard.

Predicted moves were also compared against legal move sets for given positions to assess move validity.

Chapter 7

Benchmark Dataset

7.1 Dataset Source

The dataset used in this project consists of publicly available chess games obtained from the Lichess online chess platform [4]. Lichess provides a large-scale collection of human-played chess games in Portable Game Notation (PGN) format, covering a wide range of player skill levels, time controls, and opening variations. These datasets are commonly used in chess-related machine learning research due to their size, diversity, and standardized representation.

For this project, a subset of rated standard games was selected from the Lichess database to ensure consistency in game format and move quality.

7.2 Dataset Format

Each game in the dataset is stored in PGN format, which includes:

- Metadata headers such as player names, Elo ratings, game result, and date
- A sequential list of moves represented using Standard Algebraic Notation (SAN)

SAN encodes chess moves in a compact and human-readable textual form, making it suitable for treating chess games as sequences analogous to natural language sentences.

7.3 Dataset Preprocessing

The raw PGN files were preprocessed to extract only the information required for next-move prediction. The preprocessing steps included:

- Parsing PGN files using the `python-chess` library
- Extracting move sequences from each game

- Discarding games with illegal moves or insufficient number of plies
- Converting move sequences into space-separated SAN tokens

Games with fewer than six plies were removed to avoid trivial or incomplete games.

7.4 Dataset Size and Statistics

After preprocessing, a total of 10,000 chess games were retained for experimentation. From these games, multiple input–target pairs were generated using a sliding window approach, where each prefix of a move sequence serves as input and the subsequent move is treated as the prediction target.

Key dataset statistics are summarized below:

- Number of games: 10,000
- Total move sequences (training samples): Several hundred thousand
- Vocabulary size (unique SAN moves): 3,672
- Average game length: Approximately 80 moves

7.5 Vocabulary Construction

A vocabulary was constructed by collecting all unique SAN move tokens across the dataset. Each move was assigned a unique integer identifier. Special tokens were also included:

- `<PAD>` for sequence padding
- `<UNK>` for unknown or unseen moves

This vocabulary enables the conversion of symbolic chess moves into numerical representations suitable for neural network models.

7.6 Train-Test Split

The processed dataset was divided into training and testing sets using an 80:20 split. The split was performed at the sample level to ensure a sufficient number of training examples while preserving a held-out test set for unbiased evaluation.

The same train-test split was used consistently across both the LSTM and Transformer models to ensure a fair comparison.

7.7 Rationale for Dataset Selection

The Lichess dataset was chosen due to:

- Its large scale and diversity of playing styles
- Availability of high-quality human-played games
- Standardized SAN representation suitable for sequence modeling
- Widespread usage in chess-related machine learning research

Using this dataset allows the models to learn realistic move distributions and enables meaningful evaluation of next-move prediction performance.

Chapter 8

Evaluation and Results

This chapter presents a comprehensive evaluation of the proposed sequence prediction models, namely the baseline LSTM and the Transformer-based architecture. The evaluation focuses on quantitative performance metrics, training dynamics, inference efficiency, and qualitative behavior of predicted chess moves.

8.1 Evaluation Metrics

The models were evaluated using the following metrics:

- **Training Loss**: Average cross-entropy loss over the training set
- **Test Loss**: Cross-entropy loss computed on unseen test data
- **Top-1 Accuracy**: Percentage of cases where the predicted move matches the ground-truth next move
- **Top-5 Accuracy**: Percentage of cases where the correct move appears among the top five predicted moves
- **Inference Time**: Average time required to generate a prediction for a single input sequence

These metrics jointly capture predictive accuracy, generalization capability, and computational efficiency.

8.2 Training and Test Loss Analysis

Training and test loss curves provide insight into the convergence behavior and learning dynamics of both models.

8.2.1 LSTM Loss Analysis

Figure 8.1 shows the training loss as a function of training epochs for the LSTM model. The loss decreases steadily during training, indicating stable optimization behavior. However, the rate of convergence is relatively gradual, reflecting the sequential nature of recurrent architectures.

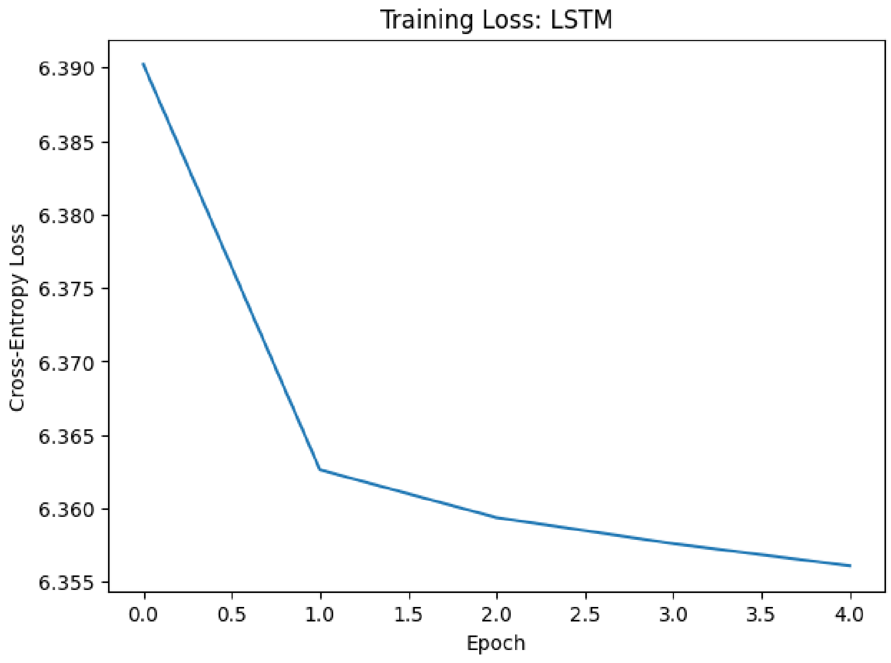

Figure 8.1: Training loss vs. epochs for the LSTM model

8.2.2 Transformer Loss Analysis

Figure 8.2 illustrates the training loss curve for the Transformer model. Compared to the LSTM, the Transformer demonstrates faster convergence in earlier epochs, attributable to parallel attention-based processing and improved gradient flow.

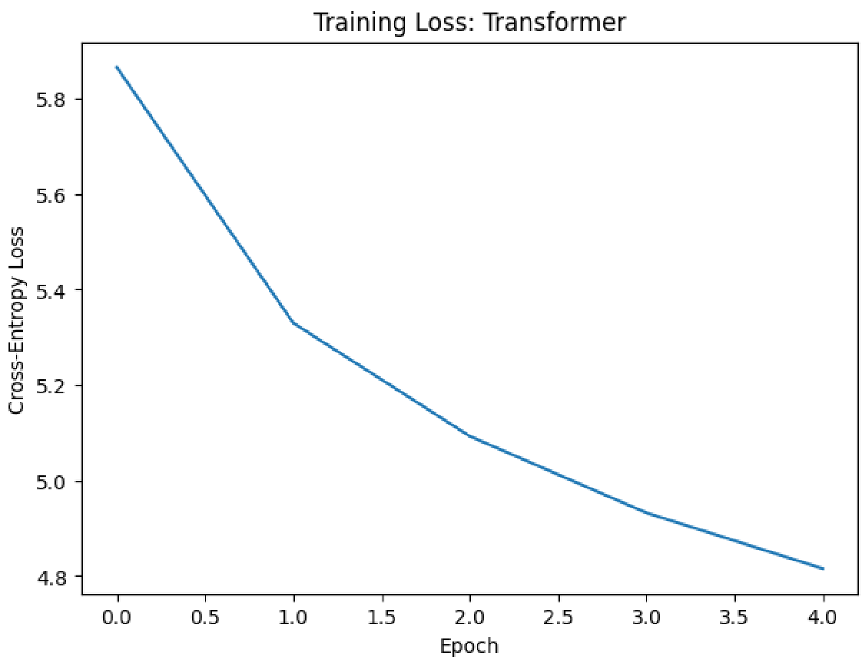

Figure 8.2: Training loss vs. epochs for the Transformer model

8.3 Accuracy Comparison

Table 8.1 summarizes the Top-1 and Top-5 accuracy obtained by both models on the test set.

Table 8.1: Accuracy comparison between LSTM and Transformer models

Metric	**LSTM**	**Transformer**
Top-1 Accuracy	1.9996 %	14.168 %
Top-5 Accuracy	7.156 %	31.985 %

While the absolute Top-1 accuracy values remain relatively low due to the large move vocabulary and inherent ambiguity in chess move prediction, the Transformer consistently outperforms the LSTM across both accuracy metrics.

8.4 Inference Time Analysis

Inference time measures the computational efficiency of the models during deployment. Table 8.2 reports the average inference time per input sequence.

Table 8.2: Inference time comparison

Model	Inference Time (seconds)
LSTM	0.003995
Transformer	0.002471

Despite its higher architectural complexity, the Transformer exhibits lower inference latency than the LSTM. This is primarily due to the Transformer's parallelized computation, whereas the LSTM processes tokens sequentially.

8.5 Quantitative Performance Summary

Table 8.3 provides a consolidated comparison of both models across key quantitative metrics.

Table 8.3: Quantitative performance comparison

Metric	LSTM	Transformer
Training Loss (after 5 epochs)	6.3561	4.8157
Test Loss	6.36879	4.85665
Top-1 Accuracy	1.9996 %	14.168 %
Top-5 Accuracy	7.156 %	31.985 %
Inference Time (s)	0.003995	0.002471
Training Time (5 epochs) (m)	approx. 15	approx. 30

8.6 Architectural Comparison

To contextualize performance differences, Table 8.4 compares key architectural characteristics of the two models.

Table 8.4: Architectural differences between LSTM and Transformer models

Component	LSTM	Transformer
Sequence Processing	Sequential	Parallel
Core Mechanism	Recurrent gates	Self-attention
Long-range Dependency Handling	Limited	Strong
Embedding Dimension	128	128
Number of Layers	1	2
Number of Parameters	approx. 1.81M	approx. 1.34M

8.7 Qualitative Evaluation

Beyond numerical metrics, qualitative analysis was conducted by decoding model predictions back into SAN notation and visualizing the predicted moves on a chessboard. Predicted moves were examined for:

- Legality with respect to the current board position
- Plausibility in the context of the game state
- Consistency across similar board configurations

While both models occasionally predicted illegal or suboptimal moves, the Transformer demonstrated a higher tendency to produce contextually reasonable predictions, particularly in mid-game positions. These observations align with the quantitative results and highlight the advantages of attention-based modeling for structured sequential decision-making tasks such as chess.

Chapter 9

Error Analysis

This chapter analyzes the failure modes of the proposed LSTM and Transformer models through both quantitative and qualitative perspectives. Given the large output vocabulary and the inherent ambiguity of chess move prediction, understanding the nature of prediction errors is essential for interpreting the reported performance metrics.

9.1 Quantitative Error Analysis

9.1.1 Impact of Large Output Vocabulary

The move prediction task involves selecting the next move from a vocabulary of 3,672 unique tokens. This results in a highly sparse target distribution, where multiple valid or reasonable moves may exist for a given board position, but only one is treated as the ground-truth label.

As a consequence, Top-1 accuracy values are relatively low, even when the model predicts a strategically valid alternative move. This limitation is intrinsic to single-label classification formulations of multi-modal decision problems such as chess.

9.1.2 Top-1 vs. Top-5 Accuracy Discrepancy

A consistent gap is observed between Top-1 and Top-5 accuracy for both models. This indicates that although the models frequently fail to predict the exact next move, the correct move often appears among the top-ranked predictions.

This behavior suggests that the learned representations capture meaningful positional patterns, but the final ranking among multiple plausible moves remains challenging.

9.1.3 Model-wise Error Distribution

The LSTM model exhibits higher error rates in positions requiring long-range dependencies, such as move planning involving earlier piece development or delayed tactical motifs.

This limitation stems from the sequential nature of recurrent processing.

The Transformer model, while achieving better overall accuracy, still produces errors in highly tactical or rare positions, particularly those underrepresented in the training data.

9.2 Qualitative Error Analysis

9.2.1 Illegal Move Predictions

Both models occasionally generate illegal moves that violate chess rules for the given position. These errors arise due to the absence of explicit rule-based constraints or legal move masking during training and inference.

Illegal move predictions are more frequent in the LSTM model, whereas the Transformer shows improved contextual awareness but does not fully eliminate such cases.

9.2.2 Plausible but Incorrect Moves

In many instances, the models predict moves that are legal and strategically reasonable but differ from the ground-truth move played in the dataset. Examples include alternative piece development moves, equivalent captures, or different checks in similar positions.

Such predictions are penalized as errors under strict accuracy metrics, despite being acceptable from a chess-playing perspective. This further explains the disparity between numerical accuracy and perceived move quality.

9.2.3 Position-Specific Failure Patterns

Both models struggle in:

- Highly tactical positions involving forced combinations
- Endgame scenarios requiring precise move sequences
- Rare openings or uncommon mid-game structures

These failures correlate with limited representation of such positions in the training data and the absence of explicit board evaluation features.

9.3 Comparison of Error Characteristics

Overall, the Transformer model demonstrates:

- Fewer illegal move predictions

- Better handling of long-range dependencies
- More consistent ranking of plausible moves

However, both models share fundamental limitations imposed by the sequence-only representation of chess games and the lack of explicit game-state supervision.

9.4 Limitations

Despite demonstrating the feasibility of sequence-based chess move prediction, the proposed models exhibit several inherent limitations arising from modeling choices, dataset characteristics, and evaluation constraints.

9.4.1 Sequence-Only Representation

Both the LSTM and Transformer models operate purely on sequences of historical moves, without explicit access to the underlying board state, piece locations, or spatial relationships. As a result, the models must implicitly infer game state information, which is error-prone, particularly in complex or tactically dense positions.

This limitation restricts the models' ability to reason about geometry-dependent concepts such as piece coordination, threats, pins, and discovered attacks.

9.4.2 Lack of Legal Move Constraints

The models are trained as unconstrained multi-class classifiers over a fixed vocabulary of moves. No legal move masking or rule-based validation is applied during training or inference.

Consequently, both models occasionally predict illegal moves that violate chess rules for the given position. While such predictions are informative for error analysis, they reduce practical usability and negatively affect accuracy metrics.

9.4.3 Single-Label Supervision

Each training example contains only a single ground-truth next move, even though multiple legal and strategically sound alternatives may exist. The models are therefore penalized for predicting plausible moves that differ from the recorded move in the dataset.

This formulation inherently limits Top-1 accuracy and does not fully capture the multi-modal nature of decision-making in chess.

9.4.4 Data Distribution Bias

The dataset consists of historical games, which may overrepresent common openings and mid-game structures while underrepresenting rare openings, sharp tactical positions, and complex endgames.

This imbalance leads to reduced model performance on less frequent or atypical positions, particularly in the later stages of the game.

9.4.5 Computational Constraints

Due to limited computational resources, model depth, embedding dimensions, and training duration were constrained. Larger models or longer training schedules may yield improved performance but were beyond the scope of this project.

Chapter 10

Future Work

The limitations identified in this project suggest several promising directions for future research and model improvement.

10.1 Incorporating Board-State Representations

A key extension would involve augmenting or replacing move-sequence inputs with explicit board-state representations, such as piece-square encodings, bitboards, or graph-based representations. This would allow models to reason directly about spatial relationships and tactical patterns.

Hybrid approaches combining move history with board-state embeddings may further improve performance.

10.2 Legal Move Masking

Integrating legal move constraints during training and inference would prevent illegal predictions and significantly improve practical applicability. Legal move masking can be implemented by restricting the output distribution to only moves valid in the current position.

This approach is commonly used in modern chess engines and reinforcement learning-based game models.

10.3 Multi-Label and Policy-Based Learning

Rather than treating move prediction as a single-label classification problem, future models could adopt multi-label objectives or policy-based learning, where multiple valid moves are considered acceptable outputs.

Such formulations better reflect the inherent ambiguity of chess decision-making and may improve both accuracy and interpretability.

10.4 Scaling Model Capacity

Increasing model depth, embedding dimensions, and attention heads - particularly for the Transformer - could enhance representational capacity. Training on larger datasets and for more epochs may further improve generalization.

10.5 Integration with Evaluation Engines

Future work could involve integrating chess engines or evaluation functions to assess the quality of predicted moves beyond exact match accuracy. Engine-based evaluation would provide a more nuanced assessment of strategic strength.

10.6 Interactive and Real-Time Demonstrations

The deployment of trained models in interactive environments, such as online chess analysis boards or real-time prediction systems, would enable qualitative evaluation and user-driven exploration of model behavior.

Chapter 11

Conclusion

This project investigated the task of next-move prediction in chess using sequence-based deep learning models. Two architectures were implemented and evaluated: a baseline Long Short-Term Memory (LSTM) network and a Transformer-based encoder model. Both models were trained on historical chess games represented as sequences of moves in Standard Algebraic Notation (SAN).

The primary objective of this work was to analyze whether modern sequence models, particularly Transformers, offer measurable advantages over recurrent architectures for structured decision-making problems such as chess move prediction.

11.1 Summary of Work

The project involved end-to-end development of the modeling pipeline, including dataset preprocessing, vocabulary construction, sequence padding, model implementation, training, evaluation, and qualitative analysis. Publicly available chess game data in PGN format was parsed to extract move sequences, which were then converted into fixed-length tokenized inputs suitable for neural network training.

A baseline LSTM model was implemented to capture sequential dependencies, while a Transformer was employed to leverage self-attention mechanisms for modeling long-range relationships between moves. Both models were trained under comparable conditions and evaluated using multiple quantitative and qualitative metrics.

11.2 Key Findings

Experimental results indicate that while both models are capable of learning meaningful patterns from move sequences, the Transformer consistently outperformed the LSTM across most evaluation metrics. The Transformer achieved lower training and test loss, higher Top-1 and Top-5 accuracy, and faster inference time per sample, despite a longer training duration.

The results suggest that self-attention mechanisms are better suited for capturing long-range dependencies in chess move sequences compared to recurrent architectures. However, absolute accuracy values remain modest, reflecting the inherent difficulty of the task and the presence of multiple valid moves in many positions.

11.3 Insights from Error Analysis

Qualitative analysis revealed that both models occasionally predict illegal moves, primarily due to the absence of explicit rule-based constraints during training and inference. Additionally, the use of single-label supervision penalizes predictions that may be strategically reasonable but differ from the recorded move in the dataset.

These observations highlight the limitations of treating chess move prediction as a pure sequence classification problem and motivate the incorporation of domain knowledge in future models.

11.4 Contributions

The key contributions of this project are summarized as follows:

- Implementation of a complete pipeline for chess move prediction using deep learning.
- Comparative analysis of LSTM and Transformer architectures on a real-world chess dataset.
- Empirical evaluation using multiple quantitative metrics and qualitative visualization.
- Identification of limitations associated with sequence-only modeling of chess.

11.5 Final Remarks

Overall, this project demonstrates that Transformer-based models provide tangible benefits over recurrent baselines for chess move prediction tasks, even under constrained computational settings. While the models developed here are not competitive with modern chess engines, they offer valuable insights into the strengths and limitations of sequence modeling approaches applied to complex, rule-governed domains.

The findings of this work serve as a foundation for future research integrating board-state representations, legal move constraints, and more expressive learning objectives to bridge the gap between data-driven sequence models and strategic game understanding.

Bibliography

[1] Niklas Fiekas. python-chess: A chess library for python, 2023.

[2] Arthur Guez et al. Learning to search with mcts. *ICML*, 2019.

[3] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[4] Lichess. Lichess open database, 2024.

[5] Reid McIlroy-Young et al. Aligning superhuman ai with human behavior: Chess as a model system. *KDD*, 2020.

[6] Adam Paszke, Sam Gross, Francisco Massa, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019.

[7] David Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 2018.

[8] Ashish Vaswani et al. Attention is all you need. *NeurIPS*, 2017.