NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

UDC 004.942

RESEARCH PROJECT PLAN

Analysis of Neural Models in Chess and Their Implementation

Submitted by the Student:

Group #234, 2nd year of study Polina Doronicheva

Approved by the Project Supervisor:

Daria Bashminova
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2025

Contents

Annotation

This project examines five modern neural engines for chess and selects two of them — Stockfish NNUE and Leela Chess Zero for implementation and detailed comparison. A total of 120 games were played across four configurations, including 30 head-to-head games between each model variant. The evaluation focused on CPU usage, move time, and positional evaluation.

Results showed that Lc0 achieved the highest average evaluation quality, with a gain of +3.2 Elo per game, but had the largest variability, 49.4 Elo, and highest CPU load. Stockfish NNUE, especially in its default configuration, demonstrated lower evaluation gains, +0.83 Elo, but superior efficiency: the lowest average CPU usage and a stable move time of around 2.3 seconds per move.

In conclusion, while Lc0 proved stronger in terms of position evaluation, NNUE emerged as the more practical and balanced engine. These findings can support developers of mobile chess applications and researchers working on CPU-efficient neural evaluation strategies.

Аннотация

В работе рассмотрены пять нейросетевых шахматных движков, из которых для реализации и тестирования выбраны два — Stockfish NNUE и Leela Chess Zero. Было сыграно 120 партий в четырёх конфигурациях. Исследование включало анализ загрузки CPU, времени на ход и качества оценок.

Lc0 показал наибольший средний прирост оценки - +3.2 Elo за партию, но при этом имел наибольшую вариативность, 49.4 Elo, и нагрузку на процессор. NNUE в дефолтной конфигурации оказался самым стабильным: среднее время хода - 2.3 секунды, минимальное использование CPU и устойчивые оценки.

Таким образом, несмотря на силу Lc0, NNUE является более сбалансированным и практичным решением для использования в условиях ограниченных вычислительных ресурсов.

Keywords

Neural Networks, AI in Chess, Move Analysis, Game Analysis, Strategy Optimization, Model Implementation

1 Introduction

Subject Area Description

Neural networks have changed how chess is played. They are used for various tasks, like analyzing moves, evaluating gameplay, optimizing strategies, and creating learning tools. By learning from large datasets, these networks can tackle complex positions and suggest the best moves.

Problem Statement

This project aims to compare neural network models used in chess to find the best one based on factors like architecture and training methods. I will pay attention to performance and efficiency when it comes to analyzing moves.

Relevance and Significance of the Task

Comparing neural network models in chess is important as AI is increasingly used in competitive and educational contexts. Even though many models have their strengths, there have not been a broad comparison among popular engines. This project's goal is to fill this gap, testing two of the best models thoroughly.

Main Results and Novelty of the Work

This paper presents a comparative analysis of five chess models: AlphaZero, Leela Chess Zero, DeepChess, Maia, and Stockfish-based NNUE. They were compared based on criteria such as code availability, training method, model type, amount of resources consumed, peak Elo and strengths and weaknesses of each model.

The results of the theoretical comparison showed that the best choice for further analysis would be models Lc0 and NNUE according to their superiority over others.

The test results showed that, while Lc0 demonstrated the highest average evaluation strength (+3.2 Elo per game), it also had the largest variability and required more CPU resources. By contrast, NNUE showed lower evaluation gains (+0.83 Elo) but offered better performance stability and efficiency, with minimal CPU load and consistent move times.

The novelty of this work is in its combination of theoretical comparisons with practical testing, providing concrete data for anyone choosing between these engines.

Structure of the Work

Section 2 describes the project timeline and key steps, from theoretical analysis to implementation and evaluation. Section 3 contains a literature review summarizing previous research on neural networks in chess and highlighting the gaps that are addressed in this project. Section 4 presents a detailed analysis and comparison of engines, including a comprehensive comparison table. Section 5 shows the implementation of Lc0 and NNUE, the experiments conducted, and the statistical evaluation of their performance. Finally, Section 6 summarizes the results and key conclusions of the study.

2 Work Plan

February: Analysis of Chess AI Models and Comparative Study

February will be fully dedicated to analyzing various chess AI models to understand their architecture, performance, and computational requirements. The main tasks include: Reviewing existing literature on chess AI models (AlphaZero, Leela Chess Zero, etc.); Examining how each model processes chess positions, evaluates moves, and executes search algorithms; Collecting key performance metrics (Elo rating, efficiency, accuracy, training requirements); Creating a structured comparison table highlighting the strengths and weaknesses of each model; Identifying the two most suitable models for implementation based on performance and feasibility. By the end of February, a clear comparison of models will be completed, and two models will be selected for implementation.

March: Implementation of Selected Models

March will focus on implementing and testing the two chosen chess AI models. The key objectives include: Setting up the programming environment and necessary libraries; Implementing the core architectures of the selected models; Running test games to evaluate move accuracy and computational efficiency; Comparing the results of both models based on defined performance metrics. By the end of March, the implemented models should be tested and their results analyzed.

April: Final Report and Documentation

April will be dedicated to documenting the findings and writing the final report. The tasks include: Summarizing key results from the model comparison and implementation; Structuring and formatting the research paper; Finalizing conclusions based on experimental findings; Ensuring proper

citation and bibliography formatting. By mid-April, the final report should be completed and ready for submission.

3 Literature Review

DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess

[1] This article proposes a fully neural approach to chess evaluation, in which the model studies the quality of a position directly based on data, without relying on traditional search algorithms. DeepChess demonstrates that only deep learning can achieve competitive results, challenging the dominance of classical engines.

However, the lack of integration with search engines limits its practical effectiveness. On the contrary, our research includes hybrid models such as Stockfish NNUE, which combine neural estimates with efficient search algorithms, which improves performance.

Neural Networks for Chess

[2] Klein provides a comprehensive overview of neural network applications in chess, from handcrafted evaluations to deep reinforcement learning. The paper serves as a foundational educational reference for understanding key architectural innovations.

However, it lacks experimental results or comparative analysis. Our project builds on this theoretical basis, and also shows the results of implemetation.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm

[4] This work introduces AlphaZero - a model capable of mastering chess from scratch using self-play and Monte Carlo Tree Search. It set a new standard for neural chess engines in terms of performance.

Despite its impact, the AlphaZero implementation remains closed-source. Our project focuses on open-source models like Lc0 and NNUE, enabling reproducibility and practical evaluation.

Monte Carlo Tree Search: A Review of Recent Modifications and Applications

[6] This review summarizes advancements in MCTS and its applications in games such as Go, Shogi, and Chess. It shows how search tree enhancements have contributed to breakthroughs in game-

playing AI.

Our project is a good addition to using MCTS by directly comparing MCTS-based engines like Lc0 with alpha-beta-based engines like NNUE in the chess domain.

Diversifying AI: Towards Creative Chess with AlphaZero

[7] This paper explores the stylistic and human-like aspects of AlphaZero's play. They analyze how neural agents can express individuality or "creativity" through decision-making, moving beyond pure strength.

Our project, by contrast, focuses on quantifiable performance metrics such as speed, efficiency, and hardware usage, providing possibility for developers to choose between engines.

Mastering Chess with a Transformer Model

[3] This article explores the use of transformer-based architecture in chess. Although it is promising for modeling sequential structure, it remains mostly theoretical and does not support implementation.

We address this issue by comparing transformer-based methods with established neural models, using standardized performance criteria and practical testing.

Stockfish: Engine Documentation

[5] This documentation details the architecture and implementation of Stockfish, including the integration of NNUE — a lightweight, efficiently updatable neural network evaluator. Its design allows powerful analysis even on CPU-limited devices.

Our study includes both default and tuned versions of Stockfish with NNUE and evaluates their performance alongside models like Lc0 to provide a balanced view across hardware constraints.

4 Neural Models in Chess: Analysis and Comparison

4.1 DeepChess Model

DeepChess model is trained on data from chess games.

Training Process of DeepChess Model

The model is trained based on a set of chess game data, and each stage is designed to extract the necessary knowledge to accurately assess the position and select the optimal move.

1. **Pretraining with Unsupervised Neural Networks**

Firstly, a deep, unsupervised neural network is used to extract about 2x1000 characteristics from chess positions. This network is trained on a large dataset consisting of 2.12 M positions where White won and 1.54 M where White lost. The model does not have prior knowledge of chess rules and does not have information about the advantages of certain positions. Instead, it analyzes the structure of positions, such as the placement of pieces, pawn structure, and other hidden dependencies.

2. **Supervised Learning for Comparing Positions**

Next, model learns to compare two positions and choose the one that is more advantageous for one side. For example, it will compare a position from a game where White won and another from a game where White lost. In this way, the model learns to associate winning positions with an advantage for the respective side, allowing it to more accurately evaluate new positions in real games.

3. **Integration into Search and Use of Alpha-Beta Pruning**

Alpha-Beta algorythm is used here as the result after comparison is the binary preference, not a numeric evaluation.

After training, the neural network is integrated into the chess engine for practical use. This improves the search and evaluation process. DeepChess compares more complex position characteristics thanks to alpha-beta search and uses this data to select optimal moves.

4. **Optimization Using Network Distillation**

At the end, the method of network distillation is applied to optimize the computational resources required for the model. Initially, DeepChess may be too slow for real-time use in chess engines, as it requires significant computational power. To address this issue, a smaller model is trained to mimic the behavior of the more complex and resource-intensive model.

Training metrics and results

Full network reaches 98% validation accuracy, however after distilliation accuracy decrease to 97%, that is still quite high level. During the game, distilled engine scores +63 Elo over 100 rapid-games and +96 Elo at a 2-hour classic game.

Also this engine has some limitations. The main are the closed-source, which affect on code supporting (architecture and training dataset have not been updated since 2016); engine works on GPU, so it is available for small amount of users. In addition DeepChess' pre-distillation speed roughly four times slower, than similar engines on CPU.

4.2 AlphaZero Model

The AlphaZero model trains using neural networks and reinforcement learning by playing against itself without relying on any human data or fixed strategies.

Initial Stage of AlphaZero Training

The engine starts with a deep neural network that has random initial weights and no understanding of chess, rely only on basic game rules. The input consists of an encoded chessboard formed as a multi-channel tensor, which represents features like castling rights (about 44 M positions, 5 billions of positions).

Using this setup, AlphaZero plays itself. It employs a Monte Carlo Tree Search algorithm that builds on the neural network's predictions, yielding a policy for possible moves and a value for the expected game outcome.

Training from Self-Play Data

Following several games, AlphaZero uses the resulting data to train its network, focusing on move probabilities and game results to refine its evaluations.

Training is performed in batches with using gradient descent. After each iteration, the updated version of the network replaces the previous one and is used in subsequent self-play games. This continuous cycle ensures improvement in playing strength without any external supervision or pre-labeled datasets.

Move Selection and Inference During Gameplay

After training, the neural network is used to guide decision-making during gameplay. On each move, AlphaZero applies the Monte Carlo Tree Search.

AlphaZero model focuses on a narrow and selective search tree. Instead of relying on brute-force search, it uses the predicted move probabilities to concentrate exploration on the most promising options. Simultaneously, the predicted position value is used to evaluate potential outcomes deeper in the tree.

After completing the search, AlphaZero selects the move, typically the most frequently explored or most confident move, without using specially developed rules or heuristics.

Engine results

After 4-hour training, AlphaZero surpassed Stockfish (the most popular engine), reached about 3550 Elo. In addition, during their 100-matches fight, AlphaZero won 28 times, had 72 draws and 0 loses, so this model could be considered the best neural chess model. However there are some limitations. Firstly, it is a closed-source model, so cannot be implemented and evaluated in real scenarios. Secondly, require a lot of CPU/GPU consumption. Finally, there is no incremental online learning, so if developers would need higher Elo, then they would start the entire expensive self-play cycle from scratch to add new data into engine.

4.3 Leela Chess Zero Model (Lc0)

Leela Chess Zero is a deep neural network–based chess engine that learns entirely from scratch through reinforcement learning and self-play. Inspired by the AlphaZero architecture, Lc0 uses Monte Carlo Tree Search to evaluate positions and select moves. Lc0 relies only on continuous self-play. The model is fully open source and was developed by a global community of contributors.

Lc0 data acquision, engine training and move selection

As was mentioned before, Lc0 has the same structure as AlphaZero, so this engine uses MCTS to evaluate positions and future neural network learning. However, there exists difference in numbers - Lc0 was trained on 45 million self-play games (2 billion positions). Also they have the similar neural training.

After training, Lc0 is used to make decisions during gameplay. On each move, the engine applies a MCTS, guided by the predictions of the trained network.

Lc0 plays a dual role during search: 1. It guides MCTS toward the most relevant branches by assigning higher prior probabilities to promising moves; 2. It evaluates leaf positions to inform

backpropagation within the search tree.

Once the search is complete, Lc0 selects the move with the greatest statistical confidence — typically the most visited move or the one with the highest predicted value. This strategy allows the engine to combine deep strategic planning with accurate positional evaluation, all without relying on handcrafted rules or domain-specific heuristics.

Comparison to AlphaZero Architecture

The overall architecture of Leela Chess Zero closely follows the AlphaZero framework: it uses a deep convolutional neural network with policy and value outputs and relies on Monte Carlo Tree Search for move selection and training data generation. Both systems are trained entirely through self-play without relying on human games, handcrafted features, or external evaluation.

However, several key differences distinguish Lc0 from AlphaZero:

1. Unlike AlphaZero, which is proprietary, Lc0 is fully open-source and developed collaboratively by volunteers around the world. 2. Lc0 uses a 112-channel input tensor, compared to AlphaZero's 119, with slightly different encoding of historical board states and auxiliary features.

3. While AlphaZero uses ResNet blocks, Lc0 implements Squeeze-and-Excitation blocks to enhance channel-wise feature recalibration.

4. Some versions of Lc0 include auxiliary outputs, such as a "moves left" head, which predicts how many moves remain in the game — a feature not present in AlphaZero.

5. Lc0 uses regular internal matches to determine whether a newly trained network outperforms the current best model, replacing it only upon success, however AlphaZero does not support this function.

Lc0 results and drawbacks

In the first 24 hours, the engine increased Elo from 2600 to 3590. In addition, at TCEC S20 (2021), Lc0 defeated Stockfish in the final with a score of 23.5-22.5. It is important to note that Lc0 needs 6–20 k MCTS simulations per move, so without GPU this engine becomes 50x slower, than with GPU support.

4.4 NNUE Model

The NNUE model is a compact neural network for position evaluation, optimized for fast execution on standard CPUs. Unlike models that learn through self-play, NNUE is trained on pre-existing games played by strong chess engines, where good moves and evaluations are already known. The input is a handcrafted feature representation that encodes the positions of pieces relative to the kings, allowing the network to perform efficient incremental updates without recomputing the full evaluation. This design enables integration with traditional brute-force search methods such as alpha-beta pruning, combining neural evaluation with selective search.

Position evaluation

The input to NNUE is a specially prepared sparse representation of the position, called HalfKP encoding. It consists of:

1. the position of the king of the side to move; 2. type, color, and coordinates of each piece on the board.

This allows for more accurate consideration of aspects such as king safety and potential threats.

The input data is encoded as a large binary vector, enabling incremental updates. After each move, only the elements that have changed are recomputed, speeding up the evaluation process.

During operation, NNUE:

1. receives the updated position from the engine,
2. computes a numerical evaluation of the position in centipawns (from the white's perspective),
3. passes this evaluation back to the engine for use in the alpha-beta search.

Precision and performance

In comparison with old versions, the latest one, based on Stockfish 12, NNUE works on 50% faster, spending on 10-20% less on each move evaluation. Also, on the latest version Elo increased on 150 points. Although the engine is currently considered a strong contender, this model has an important drawback - it does not learn from self-games, which limits its potential, unlike the models discussed before.

4.5 Maia Model

The Maia model uses neural networks that are trained on large datasets collected from real chess games played by individuals with different skill levels, sourced from the Lichess platform. Maia predicts

moves and evaluates positions based on probabilities derived from the neural network, making its playing style more human-like and similar to the play of individuals with different skill levels.

Model Training

The architecture of the neural network is similar to AlphaZero. It consists of several hidden layers that analyze the current position on the chessboard. The network is trained through supervised learning, where each move in the training data is labeled with the move that the human player chose in a real game. Approximately 12 million games played between 2016-2018 are used for each rating range. There were built 12 separate networks (from 1100 Elo to 1900 Elo).

The chessboard positions are encoded, each square on the board is represented with an encoding that indicates the presence of a piece, its type, and which side controls it. Additional information such as castling rights or en passant availability is also considered, using spatially encoded representation and multi-channel tensor formats.

When the neural network weights are randomly initialized, the model makes random moves that are not based on strategic analysis. Instead, the move is selected based on probabilities generated by the randomly initialized network. This leads to the system making errors, but these mistakes, along with self-play, allow the generation of training data for further improvement of the model. Since the network is trained solely on data from real games, it learns to predict moves characteristic of players with different skill levels step-by-step.

Improvement During Gameplay

During the game, the model continues to evaluate positions on the board. For each move, it uses two key elements, policy (probability of which move to make based on the current position) and value (evaluation of the position from the current player's perspective).

These two aspects help Maia make more accurate predictions, allowing it to improve its play. Mistakes made during the early stages become the foundation for further learning, enabling the network to adjust its weights and improve its strategy over time, using gradient descent and backpropagation.

In total, engine learns about 35 hours.

Progress Results, Strengths and Weaknesses

In human games, the top-1 move-prediction accuracy of Maia peaks at 53% for its target band, in comparison 42% for a Leela default engine and 25% for Stockfish.

However, this model has some limitations. Maia performs no search, its practical Elo ceiling is 2000, so modern engines surpass it. Also, as was mentioned before, the last training dataset was built in 2018, so engine is outdated.

4.6 Comprehensive Comparison of Chess Models

After evaluating the architecture, learning approaches, and capabilities of all examined models, two engines were selected for implementation: Leela Chess Zero and NNUE.

Lc0 was chosen as the most complex and representative neural model. Unlike outdated engines like Maia or simplified engines like DeepChess, lc0 demonstrates deep strategic understanding through reinforcement learning and MCTS. In comparison with AlphaZero, Lc0 has the same parametrs, but was chosen as its open source engine.

NNUE was chosen as it is an interesting, compact and efficient neural module optimized for integration with traditional engines such as Stockfish. It provides a clear balance between classical logic and modern learning-based assessment, This hybrid is an useful tool for comparison.

On the following page, a comprehensive comparison table summarizes the characteristics, strengths, and limitations of all reviewed chess AI models.

Model	**Code Type**	**Learning Method / Operation**	**Resources**	**Peak Elo**	**Advantages**	**Disadvantages**
DeepChess	Closed	Unsupervised and supervised	Very high, GPU required	2700	Learns without heuristics, can discover new strategies, interprets positions	Requires large dataset, long training time, binary output, project not updated since 2017
AlphaZero	Closed	Reinforcement self-play	Very high, TPU	3550	Fully autonomous, deep strategy, transferable to other games	Closed source, needs heavy hardware, expensive full update
Leela Chess Zero	Open	Distributed RL self-play	High, single modern GPU (CPU available)	3580	Open-source, community-driven, weekly new nets, strong in complex positions	Requires GPU, slow on CPU, complex setup
NNUE (Stockfish 12+)	Open	Supervised	Low, CPU-only	3900	Fast incremental eval, CPU, +150 Elo vs classic eval, integrates with existing engine	Cannot self-improve, relies on classical search for tactics
Maia	Open	Supervised on 12M human games, no search	Low	2000	Mimics human move choices, useful for training	Limited tactical depth, no self-play, ceiling 2000 Elo, dataset from 2018

Table 4.1: Comparison of Chess AI Models

5 Model Implementation and Evaluation

Experimental Setup and Computational Resources

Firstly, let us denote my computer configurations on which engines were implemented:

CPU: Intel(R) Core(TM) i7-10700F CPU @ 2.90GHz

OS: Windows 10

Engines: Lc0 v0.31.2, Stockfish NNUE v17.1

Both engines were implemented on the CPU due to the lack of GPU hardware.

It took approximately 17-18 hours to thoroughly test the two engines:
1 game took approximately 9 minutes, then 30 games took 270 minutes, 4 sets of 30 games (Lc0 and NNUE separately, Lc0 vs NNUE and NNUE tunned) took 1080 minutes = 18 hours continuous computer operation

Initial Findings

The measurements started by comparing the CPU and the time spent by the engines on each turn in milliseconds over 30 games (i.e. 30 games were played by Lc0 and then 30 games were played by NNUE)

Comparison on figure 5.1 shows that sometimes both engines demonstrated irregular spikes, however, overall NNUE illustrated a more consistent and moderate CPU usage.

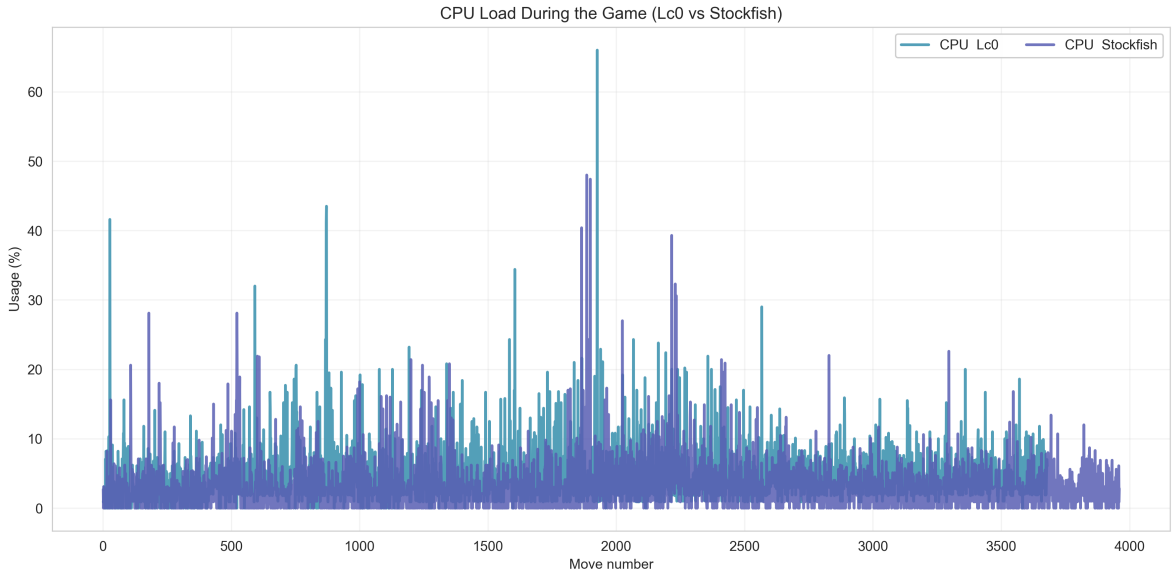

Figure 5.1: Comparison of CPU usage between Lc0 and Stockfish NNUE over the course of the game

Also, comparing the time for each turn, it is noticeable that NNUE spends an average of 2.3-2.32 seconds, which shows that the engine is quite stable (figure 5.2). Lc0, in turn, spends from 1.7 to 2.9 seconds, which shows its instability, but this also means that in some moments the engine copes faster than its rival, it remains to check whether the effect is not lost with a decrease in time.

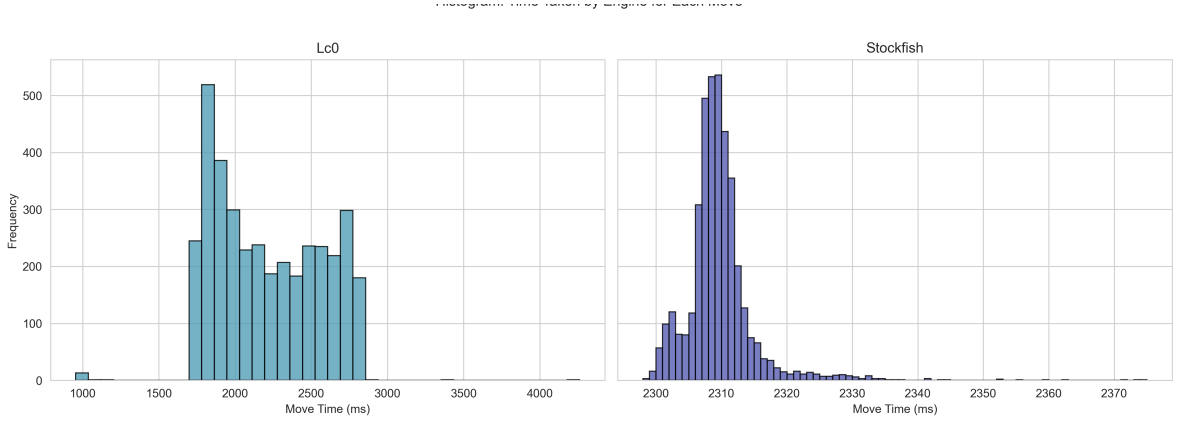

Figure 5.2: Histogram of Move Times for Each Engine

Starting to evaluate the effectiveness of each engine, I will run Lc0 against Stockfish to see what results will be obtained. As result, it turned out that both engines won 15 games each - that is, a draw. Therefore, it seems that their effectiveness is identical. However, the pie chart 5.3 shows that Lc0 turned out to be 2% more effective than NNUE, but this did not produce results in our sample. It is also interesting that the graph 5.3 shows that, on average, the odds are in different directions (above zero means that Lc0 in the lead; below zero means that NNUE is in the lead) match, except for one step of NNUE, where he was apparently able to checkmate quickly. This means that the efficiencies of the two engines are very similar; it remains to test this using hypotheses, since we had a sample of only 30 batches.

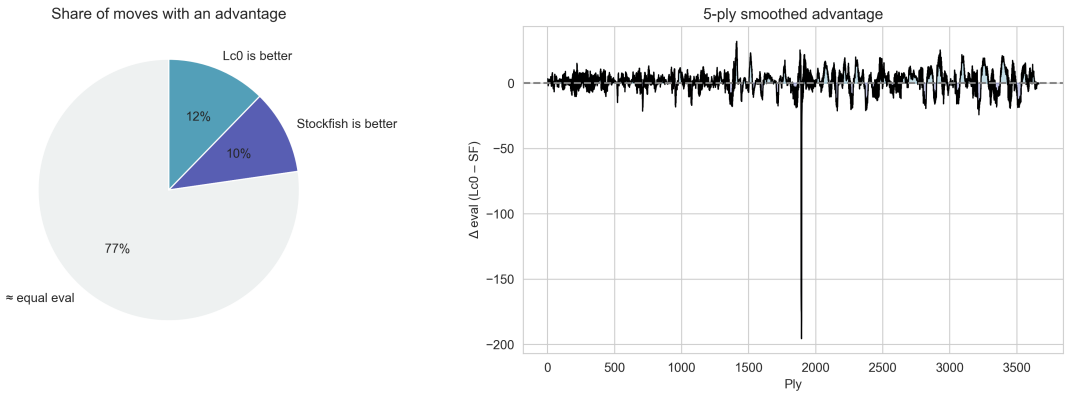

Figure 5.3: Move Evaluation - Lc0 vs NNUE

Statistical Hypothesis Testing

Statistical Tests Used:

- **Kolmogorov–Smirnov (K-S) Test:**

  This non-parametric test checks whether two samples come from the same distribution. It compares the cumulative distributions of the samples and looks for the largest difference between them:

  $$D = \sup_x |F_1(x) - F_2(x)|$$

  where $F_1(x)$ and $F_2(x)$ are the cumulative distribution functions. If the p-value is small, we conclude that the distributions are significantly different.

- **Levene's Test for Equal Variances:**

  This test checks whether two groups have the same variance. It does not assume normality of the data and is more robust than other variance tests. The values are transformed using:

  $$Z_{ij} = |X_{ij} - \bar{X}_i|$$

  where $\bar{X}_i$ is the group mean. If the p-value is small, we conclude that the variability differs significantly between the two samples.

- **Welch's t-Test:**

  This test compares the mean values of two independent samples. The formula for the test statistic is:

  $$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

  If the p-value is small, we conclude that the difference between the means is statistically significant.

Hypothesis 1 - Move time distributions are the same for Lc0 and default NNUE

To check it I used the two-sample Kolmogorov-Smirnov test.

H_0: The distribution of move times for LCZero is the same as that for Stockfish.

H_1: The distributions of move times are different between LCZero and Stockfish.

Test results:

- KS Statistic: 0.584
- p-value: 0.000

Since the p-value is less than 0.05, the null hypothesis is rejected. Thus, there is a statistically significant difference in move-time distributions between Lc0 and NNUE.

The graph 5.4 shows that NNUE really spends around 2.3 seconds, and is a stable engine when Lc0 does not have such stability. However, this is because the Lc0 neural network reacts differently to each position, while NNUE does not have such a function.

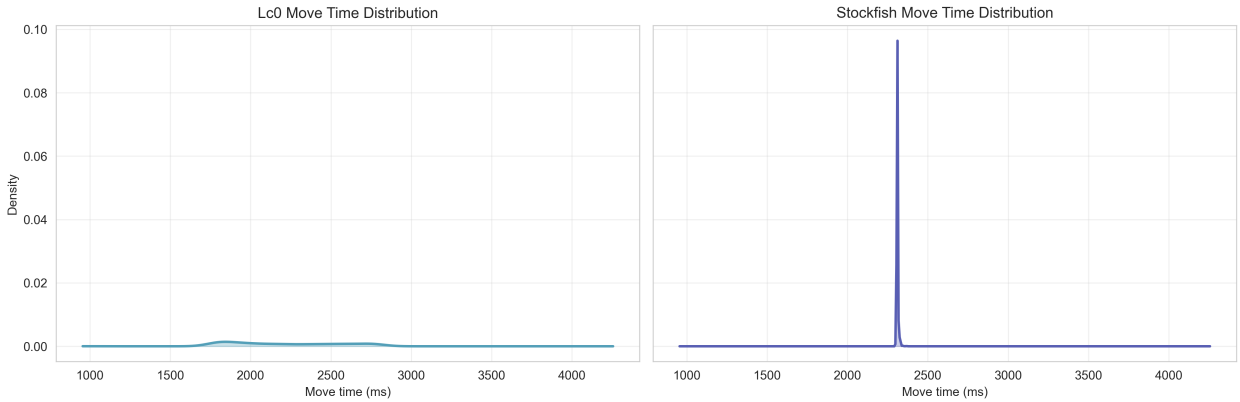

Figure 5.4: Move Times distribution: Lc0 vs Stockfish

Hypothesis 2 - CPU usage variability is the same for Lc0 and Stockfish.

To check the variability of the CPU consumption, I will use Levene's test

H_0: The variance of CPU usage per move is equal for LCZero and Stockfish.

H_1: The variance of CPU usage per move is significantly different between LCZero and Stockfish.

Test results:

- Levene's Statistic: 6.08
- p-value: 0.014

Since the p-value is less than 0.05, I reject null hypothesis. This indicates that the variance of CPU usage is different between the engines.

Figure 5.5 illustrates that CPU load during Lc0 matches has a larger spread than when while playing NNUE. In addition, it can be seen that the load during NNUE game is on average less (since

it is closer to the y axis), which can be explained by the fact that NNUE is initially optimized for CPU operation - it uses a fixed search depth and a lightweight form of neural network estimation. Lc0 was run on a CPU in testing, however, this engine was originally designed to work on GPU and performs a more complex evaluation, which leads to a less stable load on the processor.

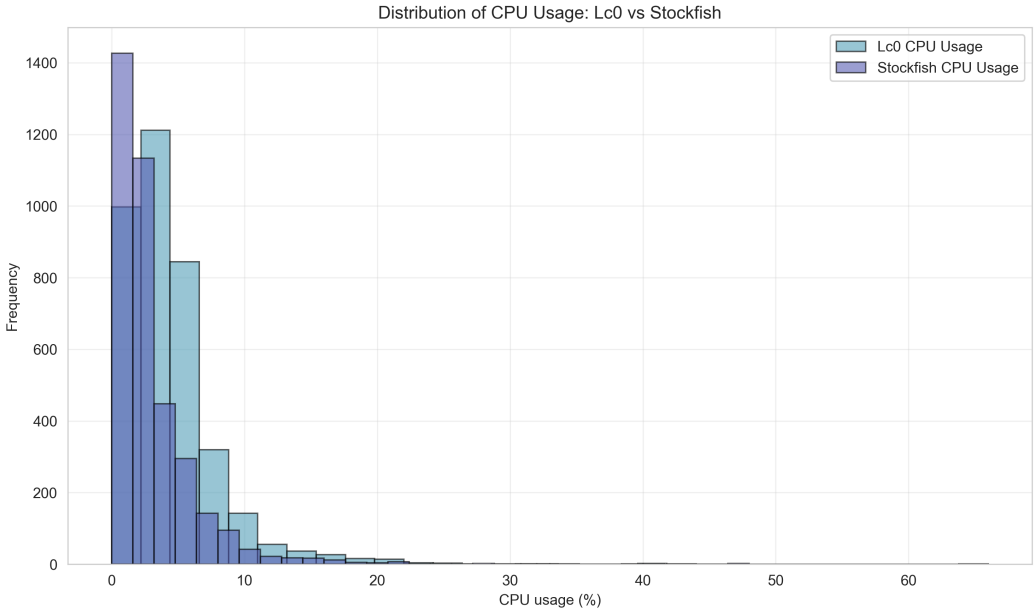

Figure 5.5: Distribution of CPU Usage: Lc0 vs Stockfish

Hypothesis 3 - The mean evaluation scores produced by the default and tuned versions of Stockfish are equal

The next step was to try to improve NNUE so that the 2% efficiency difference with LC0 was eliminated.

To do this, first we had to change the parameters of NNUE and compare them with the previous version (it tweaked parameters such as RAM, the number of CPU threads to speed up; applied an aggressive style of play to avoid a draw; selected the maximum skill level; and slightly reduced the time to think about difficult moves, otherwise the game would have been too long)

The Welch's t-test will be used to compare the scores of the basic NNUE engine and the tuned one.

H_0: The average evaluation score from Default Stockfish is equal to that from Tuned Stockfish.

H_1: The average evaluation scores are different.

Test results:

- t-statistic: -0.202
- p-value: 0.84

Given that the p-value is greater than 0.05, then we fail to reject the null hypothesis. This means there is no statistically significant difference between the average evaluation scores of default and tuned NNUE.

In addition to the above results, here are provided graphs 5.6 of the scores obtained during the games and the distribution. It is noticeable that the score evaluations for the tuned engine are more variable, but in general they are similar.

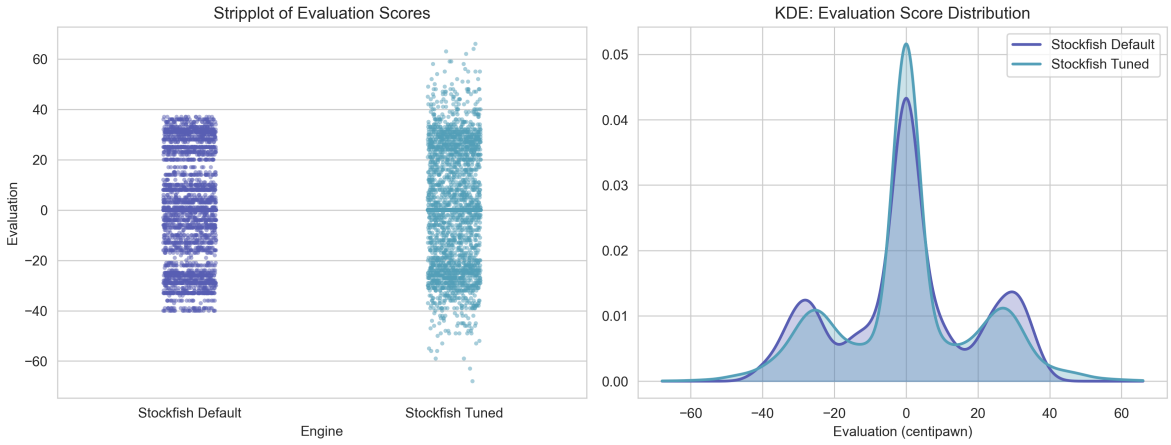

Figure 5.6: Comparison of Evaluation Scores: Default vs Tuned Stockfish

Hypothesis 4 - Move time distributions are the same

Now it's worth checking if the running time has improved without loss of efficiency. To do this, we will use the Kolmogorov-Smirnov test.

H_0: The distributions of move computation times for Default and Tuned Stockfish are the same.

H_1: The distributions of move times differ significantly between the two versions.

Test results:

- KS Statistic: 0.999
- p-value: 0.000

Since the p-value is less than 0.05, we must reject the null hypothesis. This confirms that setting up the engine has a significant impact on the time needed to calculate the moves.

Contrary to expectations, the graph 5.7 shows that the tuned parameters led to an increase in the time for calculating moves, despite a decrease in the "Slow Mover" parameter, which is responsible for the calculation time of the engine. This could be influenced by configurations such as skill improvement, but if we combine the results of hypotheses 3 and 4, we can conclude that the optimized version of the engine is still losing Lc0.

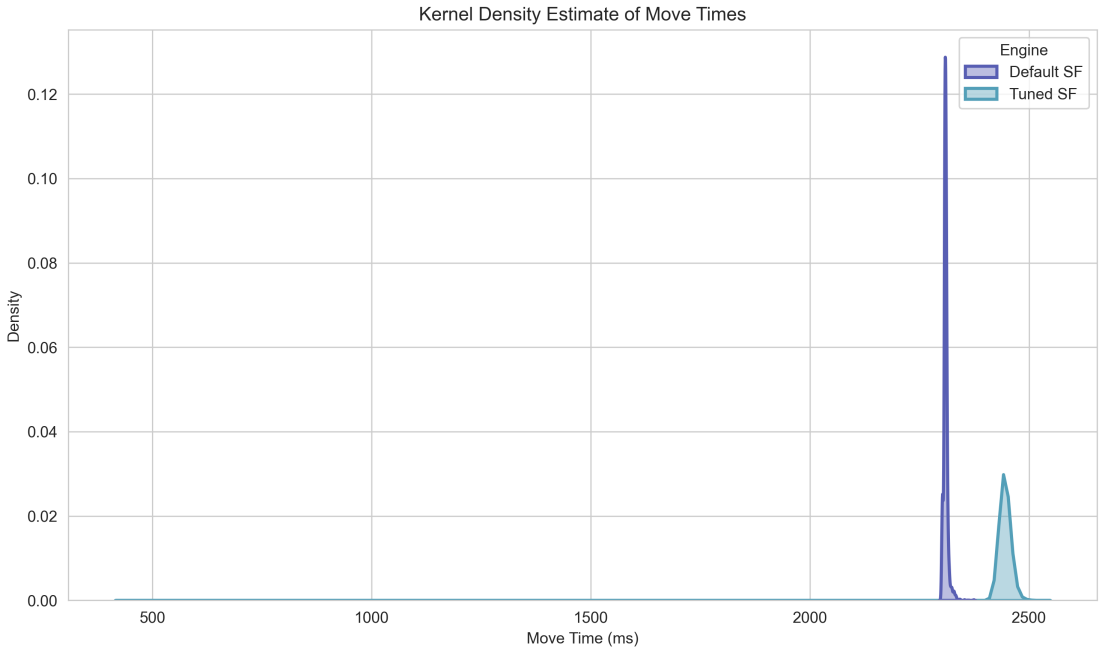

Figure 5.7: Estimate of Move Times: Default vs Tuned Stockfish

Summary of Engine Metrics

Now let's go to the final table 5.8 to summarize the calculations.

First, the worst time engine is the tuned version of NNUE, which showed an average result of 2.4 seconds, while the best is Lc0, which took about 2.2 seconds to complete, but despite this advantage, this engine had a strong variation, in around 0.3 seconds, which made this engine the least stable. According to the same two indicators, NNUE ranked 2nd.

Second, as the results above have already shown, Lc0 consumed more CPU than both versions of NNUE. In this category, the basic NNUE was the best.

Third, moving on to the scores, it can be continued to consider the results in centipawns, which may not be so clear. Therefore, the results in the table are converted to Elo. 1 Elo is approximately 1.6 - 2 centipawns. After the calculations, it is clear that Lc0 had the largest increase on average, +3.2 Elo per game. NNUE performed worse several times: the basic version showed an increase of only +0.83 Elo, and the tuned version +1.01 Elo. However, Lc0 had the greatest variability - as much as 49.41 Elo, while NNUE had 10 fewer points, around 38 Elo.

To summarize, Lc0 played stronger than its opponent, despite being less stable. However, despite such results, this model cannot be called the best, because it lost to NNUE in terms of parameters such as CPU load and turn selection time. Therefore, to summarize, the best model would be the default NNUE - less ambitious, but more stable and providing an increase, albeit not as pronounced.

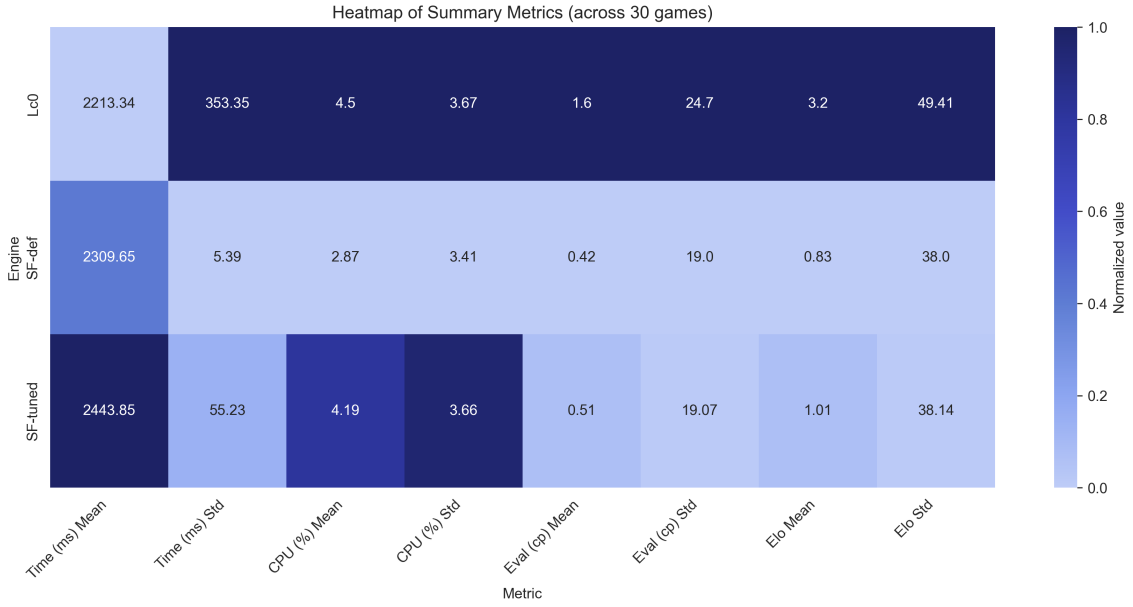

Figure 5.8: Summary Metrics

6 Conclusion

This project combined a theoretical and practical study of five modern neural chess engines.

The theoretical part involved a comparative review of five leading models, assessing their architectures, training methods, and resource requirements. Based on this, Lc0 and NNUE were selected for practical evaluation.

The code part included the implementation and testing of both engines across 30 games. During the gameplay, there were stored move time, CPU usage and evaluation quality for the further analysis with testing hypotheses.

The results showed that while Lc0 provides stronger positional evaluations, NNUE is more stable. Overall, NNUE is recommended as the more practical engine for general use, especially in limited-resource environments.

References

[1] Eli David, Nathan S. Netanyahu, and Lior Wolf. "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess". In: *arXiv* 1711.09667 (2017). URL: https://arxiv.org/abs/1711.09667.

[2] Dominik Klein. "Neural Networks for Chess". In: *arXiv* 2209.01506 (2022). URL: https://arxiv.org/abs/2209.01506.

[3] Daniel Monroe and Philip A. Chalmers. "Mastering Chess with a Transformer Model". In: *arXiv* 2409.12272 (2024). URL: https://arxiv.org/abs/2409.12272.

[4] David Silver, Thomas Hubert, Julian Schrittwieser, et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *arXiv* 1712.01815 (2017). URL: https://arxiv.org/abs/1712.01815.

[5] Stockfish Developers. "Stockfish: Engine Documentation". In: *stockfishchess.org* Documentation (2024). URL: https://stockfishchess.org/docs/.

[6] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. "Monte Carlo Tree Search: A Review of Recent Modifications and Applications". In: *arXiv* 2103.04931 (2022). URL: https://arxiv.org/abs/2103.04931.

[7] Tom Zahavy, Vivek Veeriah, Shaobo Hou, et al. "Diversifying AI: Towards Creative Chess with AlphaZero". In: *arXiv* 2308.09175 (2024). URL: https://arxiv.org/abs/2308.09175.

A Appendix

Multi-channel Tensor - a multidimensional data array used to represent a chess position. Each "channel" can encode a separate type of information (for ex, shapes, castling rights, move history, etc.), which allows the neural network to perceive a position as an image with several layers of features.

Distillation - a model compression technique in which a smaller neural network is trained to repeat the behavior of a larger and more accurate model. It is used to speed up work while maintaining high accuracy.

Gradient Descent - An optimization method used for training neural networks. It updates the model parameters in the direction of the greatest reduction in error.