2024 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT) | 979-8-3503-8592-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CONECCT62155.2024.10677087

Implementing the Chess Engine using NNUE with Nega-Max Algorithm

Angad Makarand Chitale[1], Aaron Mano Cherian[2], Amitoz Singh[3], Prakasam P[4*]
Vellore Institute of Technology, Vellore, India
[1]angadmakarand.chitale2021@vitstudent.ac.in, [2]aaronmano.cherian2021@vitstudent.ac.in, [3]amitoz.singh2021@vitstudent.ac.in, [4*]prakasamp@gmail.com

***Abstract*—The game of chess has been long used for benchmarking algorithms. It is because of the unique combination of intuition, intelligence and memory required to win a game of chess. Therefore, chess has been used to judge the "smartness" of machines. Stockfish, the strongest chess engine available currently, utilises the Min-Max algorithm with Efficiently Updateable Neural Network (NNUE). NNUE being added to the Stockfish repo in 2020, since then NNUE has largely increased the capacity and accuracy of the engine. However, Min-Max algorithm has a peculiar disadvantage with respect to Nega-Max algorithm. That is while Min-Max algorithm computes the feasibility of a particular move only in positive numbers, thus leading to a requirement of large number of comparisons. The Nega-Max algorithm utilises both negative and positive numbers. Thus, Nega-Max simplifies the comparison of feasibility of any two nodes in question. This reduces the overall computational overhead and makes the chess engine faster and more accurate at least in theory. To prove the theoretical efficiency of Nega-Max, we compared "Stockdory", a chess engine implementing NNUE along with Nega-Max, to "Stockfish". Results show that Stockfish outperformed Stockdory while analysing both the unorthodox psychological playing style of Lasker and the more positional play of Alekhine.**

Keywords—Stockfish, Alpha Beta pruning, Nega-Max, Chess, Artificial Intelligence, Efficiently Updateable Neural Network (NNUE)

I. Introduction

Zero sum games provide a viable option to compare and benchmark the algorithms in finding optimal solutions and the computational power of the computers. AI algorithms in essence are self-looping algorithms that give themselves feedback and utilise them to improve their answers.

Chess, a zero-sum game, is popular for being used as the game to benchmark the algorithms because of large number of variations that can be produced. A game of chess involves creatively utilising the already played plethora of moves from a data set. Therefore, it is used to judge the 'smartness' of machines [8].

A. Motivation

This research aims to advance computer chess and artificial intelligence (AI) by integrating Neural Network with Updated Evaluation (NNUE) and the Nega-Max algorithm. Chess, as a zero-sum game, provides a suitable platform for evaluating AI systems due to its inherent complexity. Traditional chess engines, proficient in storage and searching, lack the ability to replicate human-like ingenuity crucial for strategic decision-making. The research aims to address this challenge by combining NNUE, which dynamically refines piece values through neural networks, with the depth-first search strategy of the Nega-Max algorithm[6]. The envisioned outcome is a technologically sophisticated chess engine that excels in computational efficiency and strategic decision-making, surpassing the capabilities of most conventional engines.

B. Research Contributionns

Computers specialise in storage and effective searching but lack the human ingenuity and intuition. However, the AI models with neural networks are closing these gaps fast. Alpha Zero, a chess engine, for instance creates an abstract human brain with the help of neural network to develop some kind of intuition [5]. Chess is a game that encompasses both tactics and strategy. Tactical moves involve calculating potential sequences of moves to determine forced advantages or disadvantages. On the other hand, strategic decisions involve evaluating positions without extensive calculation. Chess engines typically utilize Alpha-Beta pruning or Monte Carlo search algorithms to generate move sequences while optimizing them through neural networks.

To achieve Alpha Beta pruning, the Min-Max algorithm is widely used in chess engines [4]. However, a more optimised version of the Min-Max exists which is known as the Nega-Max search. The introduction of Efficiently Updateable Neural Network (NNUE) has heralded a paradigm shift in chess engines, notably impacting the renowned Stockfish engine [1][2].

NNUE represents a groundbreaking methodology in evaluating chess positions, employing neural networks to refine piece values dynamically based on position and context. The architecture of StockFish NNUE HalfKP is illustrated in Fig 1.

The StockFish Chess Engine is widely considered the most powerful chess engine available for public usage [2] owing to its implementation of NNUE and Alpha Beta pruning. StockFish also has an active open-source community that contributes regularly to the optimization of its evaluation, search and pruning functions which solidify it as a powerful engine of choice to utilize as a comparison metric.

This research combines Neural Network with Updated Evaluation (NNUE) and the Nega-Max algorithm to enhance chess engine performance. The neural network efficiently evaluates chess positions using NNUE principles, while the Nega-Max algorithm's depth-first search aids in strategic decision-making. The integration improves computational efficiency, overcoming traditional limitations in chess engines. This approach results in a more sophisticated and competitive gameplay experience, contributing to the evolution of artificial intelligence in chess and strategic decision-making domains.

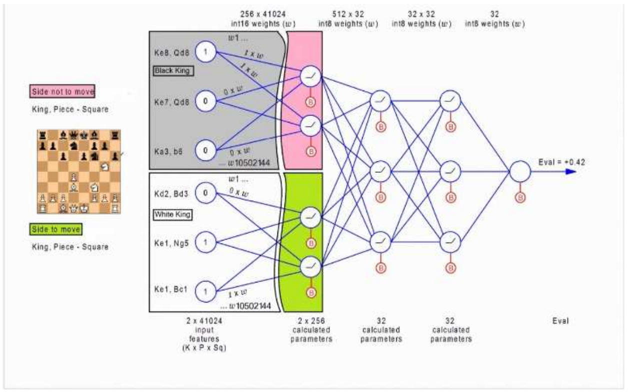

Fig 1.StockFish NNUE HalfKP Architecture [3]

Beyond chess, this research has broader implications for AI algorithms, offering pragmatic approaches to strategic decision-making in various domains. The amalgamation of NNUE and Nega-Max aims to contribute to the evolution of AI, the optimization of existing models and fostering a more sophisticated and competitive gameplay experience. The research signifies a step forward in expanding the horizons of what AI can achieve in diverse strategic decision-making contexts.

The remaining portion of this manuscript is structured as follows. The background related to Chess game is presented in section II. Section III deals with the materials and methods related to the proposed research. Experimental results and discussion are explained in section IV. Section V concluded the proposed research work.

II. Background

Simply put, a chess engine is a computer program that can play chess by analysing the position of the pieces on the board and generating moves that are most likely to lead to a favourable outcome. Understandably, a chess engine consists of several sophisticated components that work together to perform this task. As a rule of thumb, the main components include the following.

A. Board representation

As the name suggests, this is how the chess engine stores and manipulates the information about the current state of the game, such as the location and type of each piece, the castling rights, the en passant square, the half-move clock, and the side to move. There are different ways to represent the board, such as using bitboards, arrays, or lists. The choice of board representation affects the speed and efficiency of the chess engine.

B. Move generation

This powers the process of generating all the legal moves for a given position. The move generator has to take into account the basic rules of chess, such as check, capture, promotion, castling, and en passant, while also maintaining a fast and accurate response, as it is called repeatedly by the search algorithm.

C. Evaluation function

This crucial function assigns a numerical score to a given position, indicating how favourable it is for one side. The evaluation function is used by the search algorithm to compare and select the best moves [9]. A good evaluation function should be able to capture the various aspects of chess strategy and tactics, such as material balance, piece activity, pawn structure, king safety, and so on.

D. Search algorithm

This consists of the algorithm that explores the possible moves and outcomes of a given position and selects the best move according to the evaluation function [7]. The search algorithm is responsible for finding the optimal move in a reasonable amount of time, while avoiding the pitfalls of the combinatorial explosion of the chess tree.

E. Transposition table

It is a data structure that stores the results of previous evaluations and searches, in order to avoid repeating the same calculations. The transposition table can improve the speed and quality of the chess engine, most commonly by using hash keys to identify positions and retrieving the stored information.

F. Opening book

It is a database of pre-computed moves for the initial stages of the game, based on the theory and practice of chess openings. The opening book can help the chess engine to play faster and better in the opening, by avoiding unnecessary calculations and following the best lines.

G. Endgame table base

This is a database of pre-computed moves for the final stages of the game, when there are few pieces left on the board. The endgame table base can help the chess engine to play perfectly in the endgame, by knowing the exact outcome and the optimal moves for any position.

III. Materials and Methods

A. Working of a Neural Network

Neural networks, integral to high-level language models, employ interconnected nodes governed by specific functions. The intricate process of designing algorithms for these networks is exemplified in object recognition. In a simplified neural network for object recognition, the initial layer receives a complete image. Subsequent layers break down the image into pixels and employ filters to detect patterns like edges or textures. These patterns progress through layers, forming a hierarchical representation. For instance, geometric shapes like rectangles or triangles are derived from combinations of detected elements. Illustrating this concept further using a house recognition example, the network identifies elementary components like angles and lines. These elements are pooled in subsequent layers to form meaningful shapes – rectangles, windows, or roof representations. The network integrates these elements in successive layers to achieve a holistic house representation despite variations in orientation or size. In chess, neural networks play a crucial role in evaluating board positions. Similar hierarchical feature extraction and pattern recognition principles used in object recognition aid in analysing complex chess configurations. This enables engines to make strategic decisions and identify optimal. The StockFish NNUE HalfKAv2 Architecture is shown in Fig 2.

B. *Negamax Algorithm*

The Nega-Max algorithm, derived from the minimax algorithm, is a fundamental component in game theory, particularly renowned for its applications in developing game-playing artificial intelligence (ai). this algorithm plays a crucial role in optimizing decision-making processes, especially in zero-sum games like chess, where the gain of one player directly corresponds to the loss of the other.

The core principle of the Nega-Max algorithm aligns closely with the Minimax strategy, aiming to identify the optimal move in a two-player, zero-sum game setting. In zero-sum games, such as chess, the total utility is constant, meaning one player's success directly corresponds to the other player's failure. The algorithm additionally employs a recursive tree search mechanism, like Minimax. It traverses the game tree, considering both players making the best possible moves at each level. The goal is to assign scores to different positions within the game tree, reflecting the desirability of those states.

1) *Recursive Tree Search*

As aforementioned, Nega-Max employs a recursive tree search, exploring potential moves in the game tree by assuming both players are making optimal decisions. At each level of the tree, the algorithm alternates between the player maximizing their utility and the opponent minimizing it. The algorithm assigns scores to the leaf nodes based on a heuristic evaluation function. This function estimates the desirability of a given game state, providing a quantitative measure for the algorithm to make decisions. As the algorithm backtracks from the leaf nodes towards the root, it selects moves with the highest scores, assuming both players will play optimally.

2) *Alpha-Beta Pruning*

One of the notable enhancements to Nega-Max is the incorporation of the Alpha-Beta pruning technique. This optimization method efficiently reduces the number of nodes evaluated in the search tree, eliminating branches that won't impact the final decision. By discarding irrelevant branches, Alpha-Beta pruning significantly enhances the algorithm's computational efficiency.

3) *Optimizing Search Space*

Nega-Max aims to optimize its search space by evaluating and prioritizing moves based on their potential outcomes. The algorithm focuses on the most promising paths in the game tree, allowing for a more streamlined and efficient decision-making process. This optimization becomes particularly crucial in complex games like chess, where the branching factor can be exceedingly high.

C. *Efficiently Updateable Neural Networks (NNUE)*

The NNUE architecture for chess evaluation features three hidden layers, dividing input bits into two halves representing the player's and opponent's pieces relative to their kings. This departure from traditional neural networks emphasizes computational efficiency.

Weight sharing optimizes the first layer by using shared weights for mirrored piece-square relations. SIMD extensions enhance computation speed, especially in the second layer, prioritizing integer operations. Training involves pairs of positions and scores from existing or self-played games, with iterative reinforcement learning refining evaluation through position re-analysis.

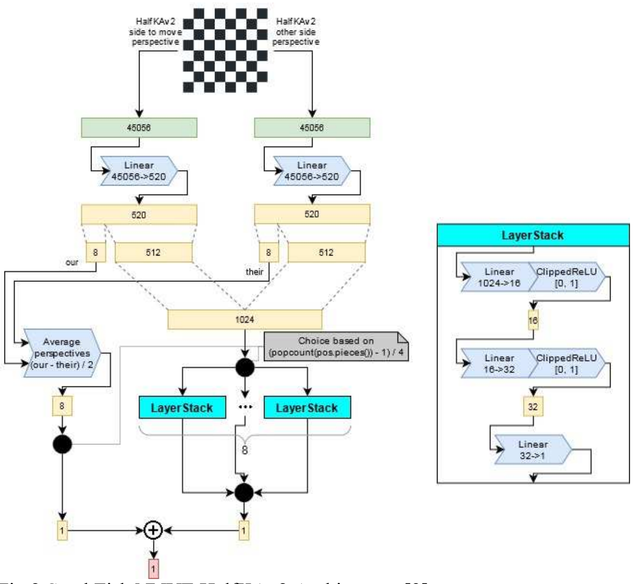

Fig 2.StockFish NNUE HalfKAv2 Architecture [3]

Inputs are split into player and opponent pieces, with the final layer providing evaluation values using clipped ReLU activation. Collaboration between engines, like Stockfish and Leela Chess Zero, underscores a communal ethos in chess engine development, fostering knowledge exchange.

IV. Results and Discussion

Testing is a crucial aspect of software development, particularly in the realm of complex systems like chess engines. As we have developed chess engine utilizing NNUE and Negamax algorithms, it becomes imperative to ensure its quality, reliability, and functionality through rigorous testing procedures. Traditional testing approaches that involve manual input selection and output observation may prove inadequate for such intricate systems.

One of the primary challenges we face in testing our chess engine is the oracle problem – determining the correctness of the observed outputs. While human experts can assess the engine's moves against expected results, this approach has limitations, especially when evaluating against a highly advanced engine like Stockfish.

Model-Based Testing (MBT) has emerged as a promising research area, offering a more systematic and automated approach to testing complex systems like chess engines. The Mean Error indicates that on average, the engine disagreed with the moves played. The Mean Absolute Error and Mean Squared Error give a sense of the average magnitude of these "errors", with the latter penalizing larger errors more heavily.

Note that these statistics are based on a simple +1/-1 scoring scheme for matching/non-matching moves. For the Purposes of drawing a competent and even comparative analysis between the Stockfish engine and our model, we have utilized two iconic chess games. The results of the same have been tabulated in table I and II. The games were downloaded in PGN form and fed to the respective engines for analysis[10]. For an accurate evaluation of the chess engine, it is necessary for us to compare statistically different types of chess games. For this reason, the chess games played between Bogoljubov and Alekhine in 1922 and between Lasker and Bauer in 1889 have been chosen.

Emanuel Lasker and Alexander Alekhine were two of the greatest chess players of all time, but their playing styles differed significantly. Lasker's game against Bauer in 1889 exemplified his "psychological chess" approach, playing counterintuitive moves like 1.f4 to unsettle opponents, sacrificing material for brilliant combinations that shattered the opponent's position. On the other hand, Alekhine's game against Bogoljubov in 1922 showcased his exceptional combinational vision, meticulously building an overwhelming position through precise calculation and deep positional understanding.

A. *Analyzing a 1922 chess game between Bogoljubov and Alekhine*

Based on the tabulated data which is shown on Table I, here are some key observations and conclusions.

1) *Observations from the StockDory Analysis*

- The engine's analysis matches 45.28% of the actual game moves, indicating a moderate level of agreement between the engine's top choices and the moves played by the players.
- The negative Mean Error (-0.094) suggests that, on average, the engine disagreed with the moves played in the game.

TABLE I. Analysis of 1922 chess game

Parameter	**StockDory Analysis**	**StockFish Analysis**
Game	Efim Bogoljubov vs Alexander Alekhine, Hastings 1922, Round 10	Efim Bogoljubov vs Alexander Alekhine, Hastings 1922, Round 10
Result Identified	0-1 (Alekhine won)	0-1 (Black wins)
Opening Identified	Dutch Defense (A84)	A84 (Dutch Defense)
Total Moves	106	106
Matching Moves (Engine vs Game)	48	85
Accuracy (Matching Moves / Total Moves)	45.28%	0.8019 (80.19%)
Engine Evaluation	The engine analysis suggests that Alekhine's strategic plan and combination were brilliant, leading to a decisive advantage and eventual win. Key moves like 19...d5!, 29...b4!, and 30...bxc3! were highlighted as remarkable.	The engine evaluates the moves played in the game and provides alternative moves it would consider.
Engine Suggestions	The engine suggests some alternative moves for White, such as 5. Nd2 instead of 5. Bd2, 10. d5 instead of 10. Qc3, and 16. b4 instead of 16. f3, which could have provided better chances.	The engine suggests several improvements for both sides at various points in the game.
Time Spent (Depth)	4 seconds per move, minimum search depth of 10	4 Seconds per move
Evaluation Time	9 minutes and 3 seconds	6 minutes and 50 Seconds
Sum of Errors (Matching = +1, Non-Matching = -1)	-10	-21
Mean Error	-0.094	-0.1981 (-19.81%)
Mean Absolute Error	0.547	0.1981 (19.81%)
Mean Squared Error	30.66	0.0392 (3.92%)

- The Mean Absolute Error (0.547) and Mean Squared Error (30.66) quantify the average magnitude of these disagreements, with the Mean Squared Error being higher due to its higher penalty for larger "errors" (non-matching moves).

- The engine's textual analysis highlights Alekhine's strategic plan and brilliant combination as decisive factors in the game, particularly praising moves like 19...d5!, 29...b4!, and 30...bxc3!.
- The engine suggests alternative moves for White (e.g., 5. Nd2, 10. d5, and 16. b4) that could have provided better chances, indicating potential improvements over the actual moves played.
- While the engine's top moves disagreed with the players' moves in a significant portion of the game (54.72%), the players' moves were often part of a deep strategic plan and brilliant combination, as highlighted by the engine's analysis.
- The relatively high Mean Absolute Error and Mean Squared Error suggest that the engine's top moves and the actual moves played sometimes differed substantially, potentially due to the depth of the players' strategic vision and the engine's evaluation limitations.
- The engine's analysis provides valuable insights into the strategic implications of the moves played, identifying key moments and potential improvements, which can contribute to a deeper understanding of the game and inform chess research and study.

2) Observations from the StockFish Analysis

- Despite the players' high caliber, the engine's analysis suggests that there were still opportunities for improvement, particularly in certain critical positions where the engine's evaluation differed significantly from the moves played.
- The Sum of Errors (-21) indicates that the engine disagreed with more moves than it agreed with, suggesting potential areas for improvement in the players' decision-making.
- The relatively high accuracy (80.19%) and low Mean Squared Error (3.92%) indicate that while the players made some inaccuracies, their overall play was of a high quality and generally aligned with the engine's evaluation.
- The analysis highlights the value of using modern chess engines in dissecting and studying historical games, as they can provide valuable insights and identify potential areas for improvement, even in games played by the greatest players of the past.
- The Dutch Defense (A84) proved to be a successful choice for Alekhine in this game, demonstrating its potential for creating complex and dynamic positions, particularly against well-prepared opponents.
- The game's length and complexity underscore the importance of deep calculation, accurate evaluation, and resilience in high-level chess competitions, as even small inaccuracies can lead to decisive consequences.
- Alekhine's victory in this game further solidified his reputation as a brilliant tactician and a formidable attacking player, capable of finding complex and deep combinational ideas.

B. Analyzing a 1889 Lasker vs Bauer chess game

Based on the tabulated data which is shown on Table II, here are some key observations and conclusions.

1) Observations from the StockDory Analysis

- The game followed the Bird's Opening (ECO code A03), which is considered an offbeat opening, likely chosen by Lasker to throw off his lesser-known opponent..
- The engine was able to match 39 out of the 75 moves played in the game, resulting in an accuracy of 52%.
- The analysis engine spent 4 seconds per move and had a minimum search depth of 10.

TABLE II. ANALYSIS OF 1899 CHESS GAME

Parameter	**StockDory Analysis**	**StockFish Analysis**
Event	Amsterdam	Amsterdam
Round	1	1
Result Identified	1-0 (White won)	1-0 (White won)
Opening Identified	Bird's Opening (A03)	Bird's Opening
Total Moves	75	75
Matching Moves (Engine vs Game)	39	36
Accuracy (Matching Moves / Total Moves)	52%	36/75 = 0.48 (48%)
Engine Evaluation	Not provided	Not provided
Engine Suggestions	Not provided	Some alternative moves suggested by the engine are mentioned
Time Spent (Depth)	4 Seconds per move, Minimum search depth=10	4 seconds per move, minimum search depth of 10
Evaluation Time	00:06:28	7 minutes 5 seconds (Total time)
Sum of Errors (Matching = +1, Non-Matching = -1)	-36	+36 - 39 = -3
Mean Error	-0.48	-3/75 = -0.04
Mean Absolute Error	0.48	0.04
Mean Squared Error	0.23	0.02

- The total evaluation time for the game was around 6.5 minutes.
- The sum of errors (matching moves = +1, non-matching moves = -1) was -36, indicating that the engine disagreed with more moves than it agreed with.

- The mean error was -0.48, suggesting that the engine tended to disagree with the moves played in the game on average.
- The mean absolute error and mean squared error provide additional insights into the magnitude of the errors.
- It's important to note that the analysis provided by the engine should be taken with a grain of salt, as it may not fully capture the psychological and strategic considerations that influenced the players' decisions during the game. Additionally, the engine's evaluation may be limited by its knowledge base and search capabilities

2) Observations from the Stockfish Analysis

- The game was played in the opening round (Round 1) of the Amsterdam event in 1889.
- Emanuel Lasker (White) defeated Johann Hermann Bauer (Black) with a score of 1-0.
- The opening played was the Lasker's Defense (A03), which was an offbeat opening chosen by Lasker against a lesser-known opponent.
- The game lasted for 75 moves.
- The engine (Stockfish) matched 36 out of 75 moves played in the game, resulting in an accuracy of 48%.
- The engine provided some alternative move suggestions at various points in the game, but no explicit evaluation of the positions was given.
- The engine analysis was performed with a depth of at least 10 and a time control of 4 seconds per move.
- The total evaluation time for the game was 7 minutes and 5 seconds.
- The sum of errors (matching moves scored as +1, non-matching moves as -1) was -3, indicating that the engine disagreed with more moves than it agreed with.
- The mean error was -0.04, suggesting a slight overall disagreement between the engine and the moves played in the game.

From the above evaluation criteria on the 1889 Lasker vs Bauer chess match, we can identify that the StockDory engine performed a better analysis as compared to the Stockfish engine (as evidenced by the Matching Moves, Accuracy and Evaluation Time criteria).

While Lasker prioritized unsettling opponents psychologically, even at the cost of objective accuracy, Alekhine focused on finding the objectively best moves through his grasp of dynamics and variations. Lasker's psychological approach often led to lower engine accuracy scores compared to Alekhine's more direct, positional style driven by precise calculation.

V. Conclusion

The analysis comparing Stockfish and Stockdory engines demonstrates that Stockfish currently holds an edge in overall performance when evaluating direct, positional style chess games justifying its status as one of the top chess engines available. However, Stockdory's performance should not be overlooked, as it displays impressive capabilities despite being a relatively new engine. Its ability to match Stockfish in certain areas, such as evaluation times and specific error metrics, highlights its potential for future growth and innovation.

References

[1] "StockDory GitHub Repo," TheBlackPlague, [Online]. Available: https://github.com/TheBlackPlague/StockDory. [Accessed: 08-Jan-2024].

[2] "StockFish," [Online]. Available: https://stockfishchess.org/. [Accessed: 07-Aug-2023].

[3] "Stockfish NNUE Images," Chess Programming, [Online]. Available: https://www.chessprogramming.org/Stockfish_NNUE. [Accessed: 27-Mar-2024].

[4] A. M. Abdelbar, "Alpha-Beta Pruning and Althofer's Pathology-Free Negamax Algorithm," Department of Computer Science & Engineering, American University in Cairo, Cairo, Egypt, 2012.

[5] S. Maharaj, N. Polson, and A. Turk, "Chess AI: Competing Paradigms for Machine Intelligence," ChessEd, Booth School of Business University of Chicago, Phillips Academy, 2021.

[6] D. Klein, "Neural Networks for Chess: The magic of deep and reinforcement learning revealed," 2022.

[7] M. Barthelemy, "Statistical analysis of chess games: space control and tipping points," Preprint, Apr. 20231. [Online]. Available: https://www.researchgate.net/publication/3702269732. [Accessed: 07-Oct-2023]

[8] W. Knight, "Instead of practicing, this AI mastered chess by reading about it," MIT Technology Review, July 31, 20191. [Online]. Available: https://www.technologyreview.com/2019/07/31/238821/instead-of-practicing-this-ai-mastered-chess-by-reading-about-it/. [Accessed: 11-Sep-2023]

[9] M. Sojka, "Performance comparison of selected chess engines," JCSI, vol. 24, pp. 228–235, 2022

[10] "Standard: Portable Game Notation Specification and Implementation Guide," Thechessdrum.net. [Online]. Available: http://www.thechessdrum.net/PGN_Reference.txt. [Accessed: 21-Sep-2023]