Three Decades of Computer Chess: A Historical and Comparative Analysis of Chess Engine Development (1987–2026)

Evarista Onokpasa
University of Jos,
Jos, Nigeria

ABSTRACT
In (E. 2025) the history of computer chess algorithms and heuristics was traced from 1947-1986. This paper rigorously examined the breakthroughs achieved during this period, with particular emphasis on the dominant performance and cutting-edge innovations demonstrated by leading chess engines in major competitive tournaments. Chess engines in this era, were designed with techniques such as game tree evaluations, min-max algorithm, alpha-beta pruning, forward pruning, transposition and refutation tables and the use of specialized hardware. This paper brings this history to date by focusing on algorithms of engines of computer chess champions from 1987-2026. It covers innovations such as Reinforced and Deep Learning, the MonteCarlo tree search, NNUE (Efficient Updatable Neural Network) and Distributed training and carries out a comparative analysis of their performances.

Keywords
Chess, Strategies, Game tree, NNUE, Neural Network, Alpha-beta, Monte Carlo Tree Search

1. INTRODUCTION
Major developments in computer chess engines, were witnessed in the late 1980s. Some of these developments include the use of bitboards, transposition tables, quiescence search, alpha-beta search with iterative deepening and parallel processing tree search. The 1990s saw the emergence of null move pruning, endgame table bases, principal variation search, brute force on specialized hardware, super-computing brute-force with heuristics. From the year 2000 onward, computer chess advanced through the development of hyper-optimized alpha–beta pruning, the introduction of Monte Carlo Tree Search (MCTS), significant enhancements to traditional alpha–beta search, and the growing integration of neural networks into evaluation and decision-making processes. These Milestone developments are discussed under Sections 4 and 5.

2. DEFINITION OF TERMS
To provide clarity, some key computer chess terminologies are defined in this section.

Evaluation Function
A mathematical formula estimating how favorable a chess position is. Core of decision-making — balances material, mobility, king safety, etc.

Heuristic
A method that guides the search process toward good moves. e.g Killer heuristics, quiesence search etc.

Neural Network (NN)
A machine learning model that mimics how humans learn patterns. here, it evaluates positions or moves.

Self Play
An AI training technique where the program plays against itself to learn.

Transposition Table
A memory table that stores previously evaluated positions (board states) to avoid duplicate calcuation, speed up search, improves search depth and move ordering.

Elo Rating
Elo rating is a system used to measure the relative skill level of players in competitive games, especially in chess and online multiplayer games. It was developed by the Hungarian-American physicist Arpad Elo (Elo 1978).

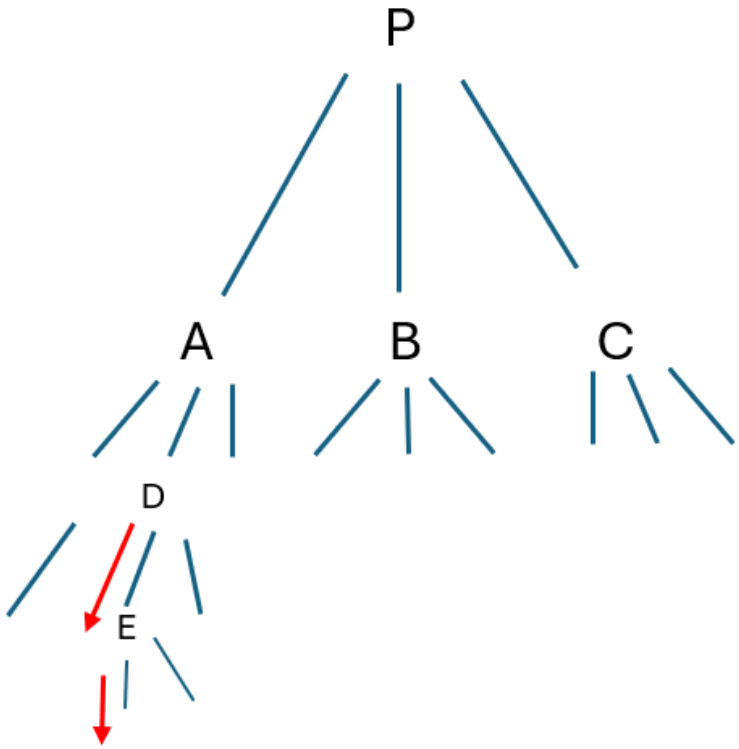
3. RELATED WORK
A considerable amount of research has been conducted on the analysis of chess engine algorithms, covering the period from 1986 to 2022. In 2021, Klein covered a study of chess engines algorithms alpha-beta tree search, Monte Carlo tree search, and NNUE was considered without an explicit performance comparison of the different chess engine (D. 2022). Dailey et al did a move comparison and analysis of Rybka, fruit, and other chess engines which participated in the World Chess Championship in 2007 (D. et al. 2014). In 2018, Acher and Esnault, carried out a massive analysis of 5 million chess games, played by chess engines competing at international chess tournaments, with the aim of answering questions related to engineering/scalabality issues in computing billions of chess moves(M. and F. 2016). Sojka did a performance comparison of chess engines, covering 2020-2022 (M. 2022). Using Plaskett's puzzle Maharaj et al. compared the strengths of Stockfish and Leela Chess Zero (S. et al. 2022). This paper covers a comprehensive summary of the algorithms of **dominant/top performing**(in world chess tournaments) chess engines covering the years 1987-2026 (with more focus on state-of-the-art engines) and a comparison of their performances.

4. CHESS ADVANCEMENTS FROM 1987-2005
4.1 Specialized Hardware Chess Engines
4.1.1 Deep Thought
Deep thought is the successor to the chess machine ChipTest. ChipTest's hardware(processor) was built in 1986 by Feng Hsu when he was a graduate student at Carnegie Mellon University. He designed the processor, using VLSI technology. When tested with a Belle type move-generator, it amazingly, had the ability to process 2,000,000 moves per second(G. 1998)! Thomas Anantharam, Feng Hsu's colleague at Carnegie Mellon, tested his simple brute force chess program on the custom hardware and achieved a search speed of 50,000 nodes per second. With the help of 2 more colleagues the software was fine tuned and the machine was entered into the NACCC

(North American Computer Chess Championship) of 1986. It lost two games and won two, an even outcome.

By observing the play of the machines during the tournament, Hsu had the idea to improve the chess program using **singular extension**. Brute-force programs examine the full breadth of the game tree—narrowed by $\alpha-\beta$ pruning, while operating with a limited search depth, n. Thus, at depth n, if there are tactical forcing moves, which could lead to better evaluation, these are not explored. The concept of singular extension is aimed at exploring just the tactical forcing moves at depth n to produce better evaluation. This is illustrated in Figure 1. So, the machine can go to a depth of $n+35$ without massively expanding the branches of the game tree, this implies an improvement in search depth without significantly compromising search speed. By the time for the 1987 NACCC tournament came ChipTest was in very good shape, with the singular extension algorithm embedded in its software and an improvement to the microcode for the hardware, ChipTest came top in the tournament!

Following the success of ChipTest, the team developed its successor, Deep Thought. Deep Thought was endowed with 2 customized VLSI chips for move generation, 250 integrated ciruit chips for search and position evaluation. This improved its search speed over ChipTests by about 100%(F. et al. 1990). Its software had 120 evaluation function parameters, with each one's weight automatically tuned. Some of the parameters included thought won the 1989 World Computer Chess Championship. Deep thought's processor was designed by Feng Hsu, during his PhD research work at Carnegie Mellon university. The processor exploited the VLSI technology to achieve a compact processor which had the ability to search between 400,000 to 500,000 chess positions per second. Deep thought ran on a software designed by Thomas Anantharam(F. et al. 1990) who was a graduate student at the time at Carnegie Mellon University. The core of the software is the game tree search, which is designed using the singular extension method. This method focuses on critical tactical positions (e.g. long forcing combinations, sacrifices, forced mates etc) by searching such positions 1 or more plies deeper to make better move choices. This is illustrated in Figure 1. It is important to mention the extended book (about 700,000 grandmaster chess opening moves) included in Deep thought's design (Campbell 1999), which enabled deep thought make good opening moves in the absence of an opening book information. The hightest USCF rating obtained by Deep thought was 2552 (J. 1989) placing it at Grand-master level.



This minimax game tree, has a singular extension from tactical position D to E, to give a better evaluation at P

4.1.2 Deep Blue

Deep blue was produced as an improvement to Deep thought. It had a specialized hardware which was 1000 times faster in search speed and had a better positional evaluation than its predecessor (F. 2002). There were bugs which had appeared in matches played by Deep thought (e.g the bug which the author called a phantom queen i.e an additional queen which appeared after an en passant move), bugs like these were caught in the software for Deep blue. Deep blue also used improved chess knowledge of tactical moves in its evaluation function (F. 2002, 195–96). Having a search speed maximum of 200 million positions per second, an improved evaluation function, deep blue was in good position to match the world human chess champion Gary Kasparov in 1997. Deep Blue made history by coming off victorious and became the first computer to beat a human chess grandmaster! Deep blue had no official USCF rating but it has an estimated USCF rating of above 2800 as it beat chess champion Gary Kasparov whose USCF rating was 2800 at the time. Deep Blue started the movement of the development of chess engines with superhuman strength of play.

4.2 Optimized Alpha-Beta Engines

4.2.1 Fritz 12 and Deep Fritz

Fritz chess engine (1-13) was originally designed by Frans Mosch and Mathias Feist. Its algorithm was based on null move pruning/null move heuristic(F. 1989; G. and S. 1990). This is a search optimization technique that relies on the idea that in most positions, choosing to do nothing (making a "null move") is usually worse than making some legal move. This is achieved by assuming a turn can be skipped and evaluating the opponents moves. The best of the opponents move ρ, is compared with the current lower and upper bound. if rho is greater than the current upper bound the branch is cut-off. If ρ is less than the current lower bound, it is ignored and search continues. if it is greater than the current lower bound it replaces it as the new lower bound. This pruning technique was a major improvement to game tree search. Prior to null-move pruning, computer chess grandmasters such as Blitz, Hitech, and Deep Thought relied on traversing the full-width game tree with $\alpha - \beta$ pruning to impressive depths in order to achieve strong chess performance. Deep Fritz essentially had same search algorihtm as Fritz but was a multi-core version and thus had greater search speed.

4.2.2 Fruit

The chess engine Fruit, was first released in 2004 by Fabian Letouzey. From its open source documentation (F. and Team, n.d.), one can read on the features of Fruit's evaluation function and tree pruning algorithms. It combines the classic min-max and alpha-beta pruning along with 4 prunning techniques(null, history, delta and futitlity). It does a verification search to solve some Zugzwang problems, verification reduction which searches Zugzwang situations to a maximum depth of 6 plies and improved move ordering and move generation compared to earlier engines. Its evaluation function has classical features of material advantage, king safety, piece mobility, pawn structure, passed pawns etc. It also employs iterative deepening in its game tree search, this can be seen in its source code published on (F. 2005). Fruit employed basic chess engine features; however, it gained advantage over similar chess engines in the use of additional pruning techniques, and its move ordering and generation. It attained an ELO rating of 2680–2750 (CCRL Team 2005).

5. CHESS ADVANCEMENTS FROM 2006-2026

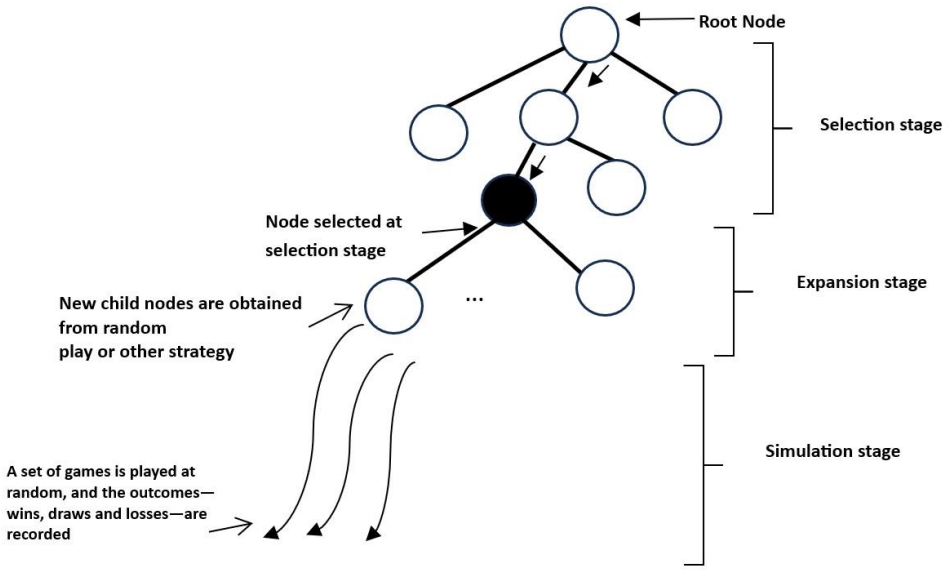
5.1 Chess Engine Built on Neural Network

5.1.1 AlphaZero (Deep Neural network and Monte-Carlo Tree Search)

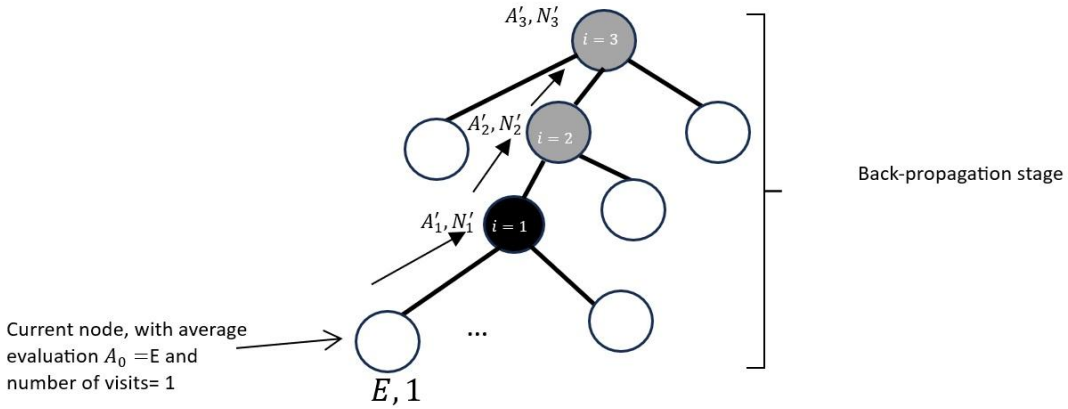
AlphaZero algorithm was built on deep neural network and the Monte-Carlo tree search(MCT). Prior to the application of neural **Monte-Carlo Tree** search relies on gathering statistics of winning moves through random play of valid legal moves. The algorithm defines 4 steps: Selection, Expansion, Simulation and Backpropagation. The **Selection** step, starts from a child node, usually the root/current node - after some iterations (this could be random play), a sequence of moves are selected to reach a leaf node (this is not a terminal node but a node which hasn't been explored for child nodes). Next the selected node is explored by adding child node(s) to the tree - **Expansion** step. At the **Simulation** step random games are played from one of the child/current node to game end (i.e. a win, loss or draw). The first 3 steps are described in Fig 2. During simulation, all winning statistics are stored and **Back-propagated** to update parent nodes with average evaluation and number of visits. A simple formula (D. 2021) can be used for the update values in the back-propagation step.

$$A_i' = (A_i * N_i + E)/N_i + 1 \qquad N_i' = N_i + 1$$

where A_i and A_i' stand for the average evaluation and updated average evaluation respectively for a parent node marked/identified with the index i. N_i is the number of times the node is visited and N_i' the updated number of times the node is visited. E is the evaluation obtained from the Simulation step. E can be defined based on the encoding for win, loss and draw. Figure 3 shows the parent nodes trailed and updated in the back-propagation step. The process loops through all the 4 steps until a predefined number of iterations is reached. When the MCT is completed, a move with the highest average evaluation can be selected or using some other heuristics like the Upper Confidence bound applied to Trees (UCT) (D. 2022).

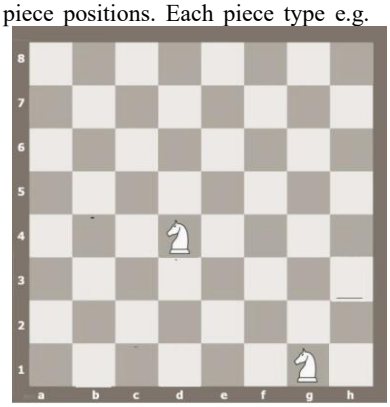


Selection, Expansion, Simulation in MonteCarlo tree search illustrated

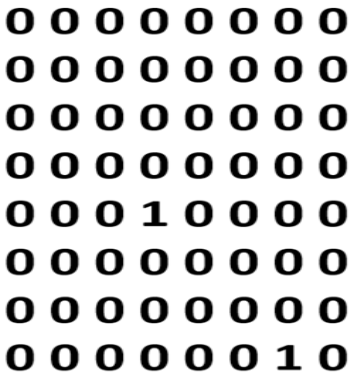


Back-propagation step, with the parent nodes shaded and indexed. A_i and number of visits N_i for each parent node are calculated using E from the simulation step.

The AlphaZero engine uses **MCTS** along with reinforced learning by training its **Neural Network** through self play. The network uses twelve 8 × 8 planes (a plane is an 8 × 8 binary array) to encode chess piece positions. Each piece type e.g. black knight, white knight etc is encoded in a plane as shown in Figure 6.



Position of White Knights in a Sample Game



The Plane for the White Knights in 4

White Knight Positions and the Plane Reprsentation

Additional planes are used to record number of repetitions of a position, to encode which player to move, which player(s) still has castling rights etc. The details of the planes designed to encode the chess game information is found in (D. 2022). The network takes as input the current chess game state which consists of 119 8×8 planes that encode information for the current game. The network output consists of the probabilities of the possible play and a value ranging from -1 to 1 for the current game state, this value indicates the likelyhood of a win or loss at the current state. The network is designed to run through 3 main steps repeatedly.

- 1 or 2 convolutional layer(s) with 256 filters, a 3 × 3 kernel and a unit stride
- batch normalization and
- rectifier nonlinearity (i.e. Rectified Linear Unit (ReLU)).

Each repetition of the 3 steps is known as a residual block. After running through the residual blocks, the output is subsequently directed into two separate heads: the **policy head** and the **value head**. In the policy head, a sequence of steps similar to the steps in the residual block is applied: 1 convolutional layer with 2 filters, 1×1 kernel and unit stride with batch normalization and ReLU to produce values between 0 and 1. These correspond to the probabilities for each possible move. The value head applies a 1×1 convolution (single filter, stride 1), followed by batch normalization and ReLU activation. The result is passed through a fully connected layer with 256 units and ReLU, and finally mapped to a single tanh-activated output via a last fully connected layer. The value produced ranges between -1 to 1. This is a brief coverage of the training process.

It is important to explain the process of obtaining training data. The training data used in Alphazero is obtained from self-play using the MonteCarlo tree search for move choices. Before training begins the network is initialized with random weights. Training data are generated by using Monte Carlo Tree Search (MCTS) with the current neural network to conduct self-play games. At each step of the game, MCTS is used to select moves by identifying promising moves and generating corresponding move probabilities. These MCTS-derived move probabilities should not be confused with the probabilities subsequently produced by the neural network during the training or prediction process. Obviously at the start of the training MCTs probabilities are definitely better than that of the neural network. A game is played from start to finish using MCTS to select moves for both players, and each position is added as training data for the neural network. The current network is then further

trained through additional self-play, producing an updated network. Games are subsequently played between the current and updated networks with MCTS support, and these games also serve as training data. If the updated network performs significantly better, it replaces the current network and becomes the new basis for further training through continued self-play.

The AlphaZero chess engines thus has a simplified learning algorithm, eliminating the use of huge training data required for supervised learning. Alphazero

5.2 NNUE (Efficient Updatable Neural Network) Chess Engines

5.2.1 Stockfish NNUE

Early releases of Stockfish (1.0 - 12.0) were built on classical search engine techniques such as alpha-beta tree search, iterative deepening, hand-crafted evaluation, transposition tables and well-thought out heuristics(e.g. Killer move, history etc) see Table [Chess Engines summary] for details on heuristics used. Stockfish 13 onwards employed Efficient Updatable Neural Network(NNUE) NNUE was first created for the game of Shogi by Yu Nasu, and was later adapted for chess in the Stockfish engine by Hisayori Noda. In the NNUE documentation (Stockfish Developers 2025) it gives details of the layers implemented in the network:

- Linear layer, implements the function

$$\begin{equation} y=Ax+b \label{eq:linear} \end{equation}$$

where A is the weight matix, x is the input column vector, b is the bias column vector and y is the output

- Clipped ReLU is an activation function defined by

$$y = min(max(x, 0),1)$$

- Sigmoid layer uses the Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

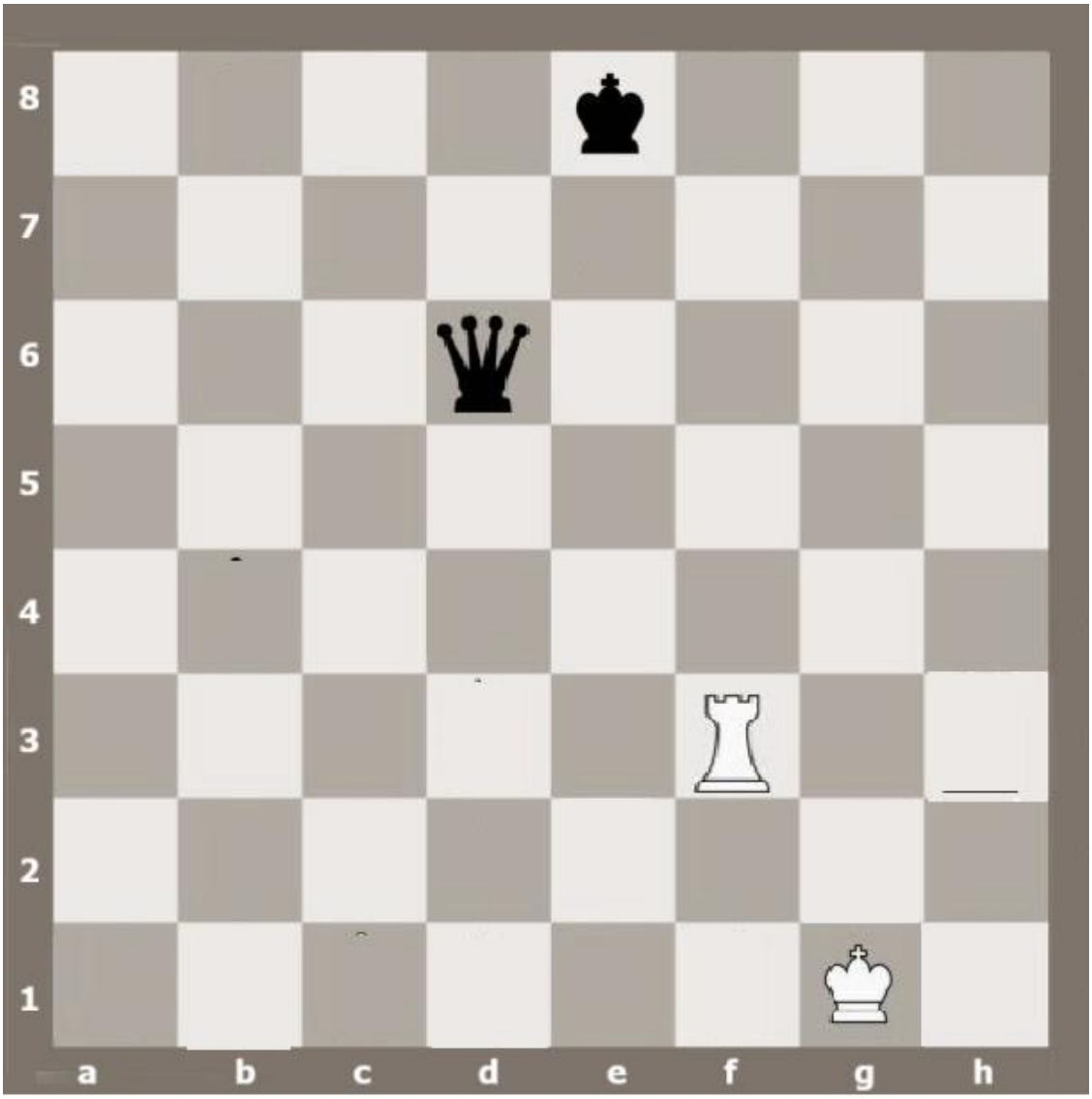
To improve computational speed it replaces the $sigmoid(4x)$ with an approximation function $quantmoid(4x)$.

The feature for the the network input is encoded as follows: each square on the board is represented as an ordered set (square, pieceType, color), where occupied squares are set to 1 and 0 otherwise. The input features can be shown with the active features i.e. squares with values set to 1. The active board position in Figure 7, is: (D4, Knight, Black) (D6, Queen, Black) (E6, Pawn, Black) (E8, King, Black) (D3,Knight, White) (F1, Bishop, White) (G1, King, White) (H3, Rook, White).

The network can be fed with both active and inactive input features depending on the feature set that is chosen. In the hidden layers of the network, the input is multiplied by a sparse matrix which reduces the size of the internal neurons. One version of the stockfish implementation starts with an input of size 512 neurons to 32 neurons to 32 neurons to 1 output.

Two important features in Stockfish are the HalfKP and HalfKA. HalfKP feature consists of the ordered sequence (ourKingSquare, pieceSquare, pieceType, pieceColor) where pieceType can be any piece except the king while HalfKA admits the king as pieceType. Note that the reference for the rank of the square in ourKingSquare and pieceSquare is counted from 1 to 8 from the perspective of the player(e.g. for the chess game in Figure 7, the HalfKP for the active features from the White perspective is: (G1, F3, rook, white), (G1, D6, queen. black) and from the black perspective (E1, D6, rook, white), (E1, D2, queen, black). These features aid the network to be trained on the connection between the king and other pieces. HalfKP are a total of 40960 combinations i.e 64 (possible location of the king) × 64(location of other piece)×5(piece types)×2(piece colors). Thus, the input starts with 40960 neurons.

The input is fed in the network via 2 accumulators, 1 accumulator for each side white/black, this is where training starts. The accumulator keeps a record of the changes made to neurons in the hidden layers without carrying out a recomputation with each move. The output from the accumulator is passed to the **linear layer**, where the weights and biases are applied using the function in Equation [eq:linear]. This produces an output of 256 neurons per accumulator. The 2 accumulators are combined either by concatenating the features or interleaving them. Next, the ClippedReLU function (Equation [eq:CReLU]) is applied. It may include several hidden layers where each neuron applies a linear transformation followed by a clipped ReLU activation. In the final layer a linear function is a applied to produce an out put in centipawns (a pawns is assigned a unit value, thus a centipawn is one-hundredth of the value of a pawn). Using the sigmoid function, the centipawn evaluation is converted to Win-Draw-Loss evaluation, where 0 = loss, 0.5 = draw and 1 = win. Stockfish has a training data generator which generates training data through self-play. The engine plays against itself while recording the positions and its evaluations (Stockfish Developers 2023). The Stockfish neural network can also be trained using training data of self-played games from other chess engines e.g. Leela Chess Zero, thanks to open-source tuning. The Stockfish neural network architecture has been refined over a dozen times from its first release. Each refinement has resulted in an improvement in the strength of the Stockfish chess engine. Stockfish is about the world's best playing chess engine, winning all events in the Top Chess Engine Championship (TCEC) and attaining an Elo rating of 3774 in the Blitz category (CCRL Team 2005).



Chess Game State to Illustrate NNUE

5.2.2 Leela Chess Zero (LcO)

This chess engine is the open source version of AlphaZero. It was made available to the public in 2018 (Leela Chess Zero Project 2019). Leela Chess Zero has the advantage of incremental improvement by global developers. At the time of release, Lc0 had essentially the same network architecture as AlphaZero. It employed a Convolution Residual Network with an output consisting of both a **policy head** and **value head**, very similar to AlphaZero. In 2022, Lc0 was moved to a Transformer model network (D. and A. 2024). This transformer model was specially adapted to the game of Chess and was termed Chessformer (Leela Chess Zero Project 2019). It has a transformer feature known as **self-attention**, which compares any square on the chess board with any other square in order to quickly focus on the squares which have most relevant relationships. Positions are represented by modifying the representation from (P. et al. 2018), which clearly shows the arrangement of the chessboard.

Lc0 was trained with static dataset from self-played games obtained from AlphaZero. Several models of Lc0 were trained, with the largest model CF-240M, trained with a dataset of about 500 milllion long games (i.e. an average of 200 positions per game. The most recent release of Lc0 with finer tuned network weights attained Elo rating of 3704 (CCRL Team 2005) in the Blitz category of major computer chess championships.

5.2.3 Komodo and Komodo dragon

Komodo was originally produced as a commercial chess engine by Larry Kaufman and Don Dailey in 2010, . This version applied the classical alpha-beta tree search, with null move pruning (ChessBase 2026a) as well as Late Move Reduction pruning. It also employs transposition tables and Syzgy end game table bases used for selecting best moves at end-game situations (ChessBase 2026b). Komodo was acquired by chess.com in 2018 and renamed to Komodo chess. In 2020 Komodo dragon was released, this engine implements alpha-beta search and Efficient Updatable Neural Network and an optional Monte Carlo Tree search mode. As stated on the Komodo Dragon official website, it can be set to one of 3 modes - Standard, Monte Carlo Tree Search and Armageddon (Komodo Chess 2026). Dragon's NNUE was trained through reinforcement learning using billions of positions and evaluations (TalkChess Forum 2020). Komodo's NNUE is CPU - efficient and thus not GPU dependent. Komodo ranks among the top 3 engines. Its most recent version has an Elo rating of 3726 (CCRL Team 2005) in the Blitz category.

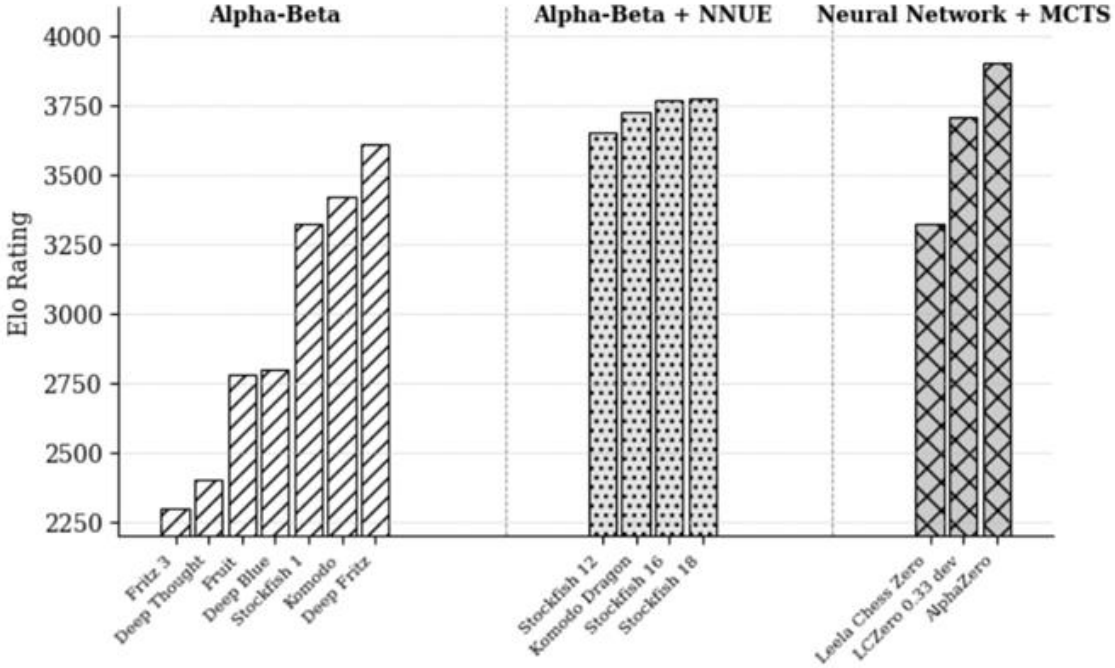
6. DISCUSSION

A study of computer chess engines in (E. 2025) which covered the period of computer chess inception to 1985, showed the dominance of chess engines built with Shannon's Type A strategy i.e. full-breadth alpha-beta search (e.g. ITEP, TECH, KAISSA, CrayBlitz, Hitech) over chess engines built with human-like move choice i.e. select one move and explore deeply (e.g. TURK and Ostrich).This evidence demonstrated that a human style of play was ineffective in the design of powerful chess engines. Specialized hardware machines Deep thought and Deep Blue reigned in the years spanning 1986-1997 and made history by defeating the Human grand master Gary Kasparov. However, these were brute force machines with chess knowledge embedded in them, with moderate improvement to the alpha-beta tree search(e.g. the singular extension in Deep Blue). The introduction of null-move pruning in alpha–beta search trees, as implemented in Fritz and Deep Fritz, was a major improvement in chess game-tree search thus Fritz could search deeper down the game tree as it had fewer branches to traverse. Fruit further included history, delta and futility pruning to its alph-beta search engine, thus having an even better advantage of fewer branches and more depth in its tree exploration, thus providing an improved Shannon Type A strategy. The introduction of neural networks transformed chess engine design by replacing many handcrafted evaluation heuristics and alpha-beta tree search with data-driven learning methods. Chess engine performance has moved beyond the debate over whether Shannon Type A or Shannon Type B strategies are superior, and is now focused on how effectively a chess engine can learn in order to improve its playing strength. Which learning model acquires stronger chess playing skills-Deep, Reinforcement, transformer or supervised? Or would a hybrid combination produce better results

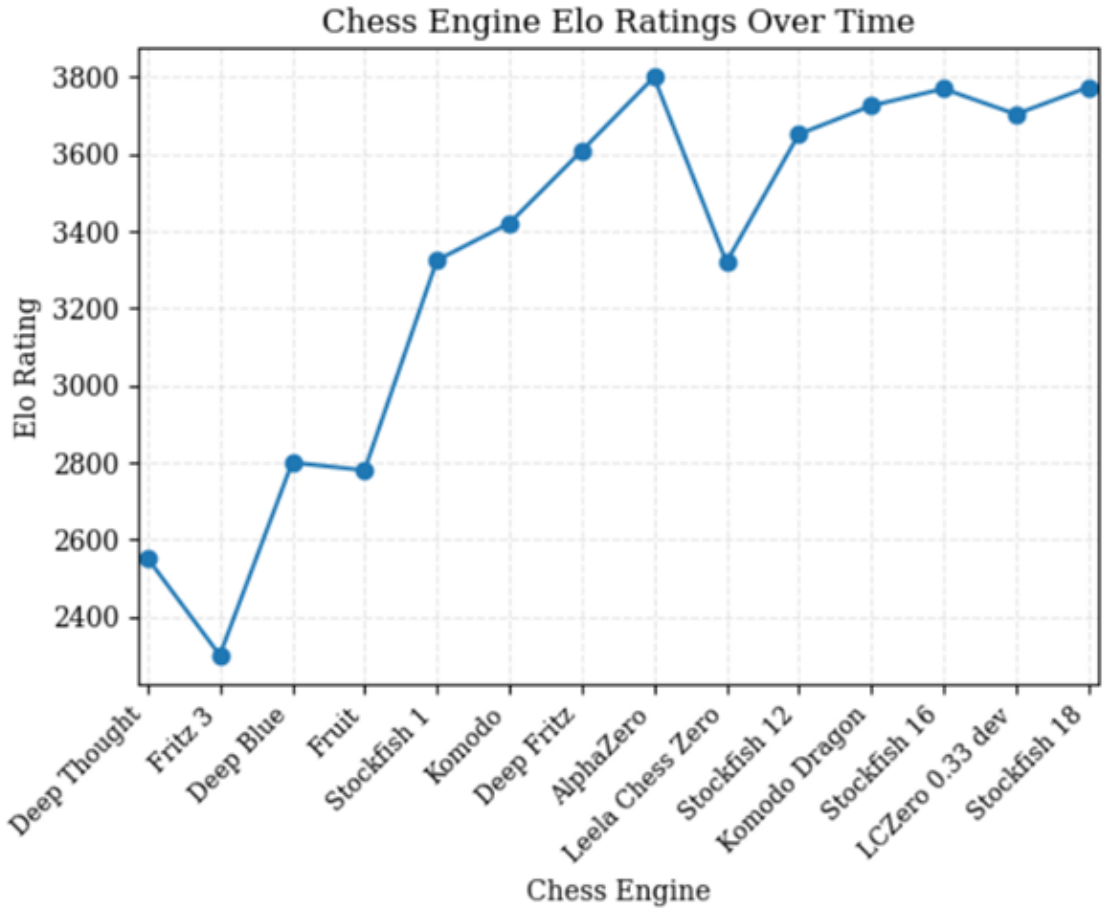
7. PERFORMANCE COMPARISON

In this section, a comparative analysis of the performance (Elo rating) of the dominant chess engines covered in this study is carried out. Table [tab:Chess Engines summary] gives a summary of the features (Implemented algorithm, year of release, Hardware requirement, Search speed and Elo rating according to (CCRL Team 2005)) of these engines.

In Figure 9, a steady improvement in the Elo ratings of dominant chess engines can be observed over the years, with the exception of performance declines observed for Fritz and Leela Chess Zero.. Fritz used a similar algorithm as Deep thought and Deep blue but was not endowed with massive specialized hardware which its contemporaies had. As such Fritz couldn't have explored the alpha-beta tree to the extent as did Deep thought and Deep Blue. It is important to mention that Fritz did win the WCCC (World Computer Chess) championship in 1995 in which a Deep Blue prototypeparticipated. This championship result thus proves that the strength in the design of Deep Blue machine lies heavily on its large-scale, massively parallel custom hardware architecture. Leela Chess Zero was launched as the open source version of Alphazero. At the time of launch it had no pretrained chess knowledge and attained relatively lower ratings to its predecessor AlphaZero. However, after reinforced learning from self play games it gained strength of play and its currently rated at 3704 (Elo). Comparing the hardware requirements of different chess engines in this era presents a challenge, as hardware technologies have advanced significantly within this period. A comparison can be done though, for engines that employed similar technologies, Fritz3 which was released in 1995 ran on a single core pentium CPU Figure 8 brings to fore the performances of chess engines by their algorithmic paradigms. The Hybrid Engines which combined Alpha-Beta and NNUE performed better on average than engines built on the other paradigms.This may be an indication that Alpha-beta and NNUE algorithms are more robust and efficient across implementations than other algorithms,



Chess engines grouped by algorithmic paradigm (e.g., Alpha–beta pruning, Monte Carlo Tree Search, neural network–based evaluation, or hybrid architectures) plotted against measured Elo rating performance.



Comparison of Elo ratings of major chess engines over time, ordered by year of release. Illustrates the progress of computer chess strength from early classical alpha–beta search systems to modern neural-network-based and hybrid engines, showing a steady increase in playing strength across successive generations.

8. CONCLUSION AND FUTURE WORK

The last 40 years saw an impressive development of chess engines which rose beyond human capabilities in chess playing skills. From specialized brute-force machines – Deep Thought and Deep Blue, to cleverly optimized pruning techniques used in Fruit, machines have placed humans at the rear in chess tournaments. With the application of Neural networks through Deep, Reinforced and Supervised learning as well as Efficient Updatable Neural Networks, the improvement in strength of chess engines may know no bounds. As years pass and increasingly efficient learning methods are employed in chess engine design, a continuously updated history is necessary to track these developments.

9. REFERENCES

[1] Campbell, M. 1999. “Knowledge Discovery in Deep Blue.” *Communications of the ACM* 42 (11): 65–67.

[2] CCRL Team. 2005. *Computer Chess Rating Lists (CCRL)*. https://computerchess.org/.

[3] ChessBase. 2026b. “Engine Parameters.” https://help.chessbase.com/Komodo/15/Eng/000038.htm.

[4] ChessBase. 2026a. *Engine Parameters*. Https://help.chessbase.com/Komodo/15/Eng/000038.htm.

[5] D., Dailey, Hair A., and Watkins M. 2014. “Move Similarity in Chess.” *Journal of Game Theory* 3 (2): 45–52. https://doi.org/10.5923/j.jgt.20140302.02.

[6] D., Klein. 2021. *Neural Networks for Chess: The Magic of Deep and Reinforcement Learning Revealed*. Independently published.

[7] D., Klein. 2022. *Neural Networks for Chess*. https://arxiv.org/abs/2209.01506.

[8] D., Monroe, and Chalmers P. A. 2024. “Mastering Chess with a Transformer Model.” *arXiv Preprint arXiv:2409.12272*. https://arxiv.org/abs/2409.12272.

[9] E., Onokpasa. 2025. “History and Milestone Developments in Computer Chess Algorithms from 1947 -1986.” *International Journal of Computer Applications* (New York, USA) 187 (51): 7–16. https://doi.org/ 10.5120/ijca2025925838 .

[10] Elo, Arpad E. 1978. *The Rating of Chessplayers, Past and Present*. Arco Publishing.

[11] F., Beal D. 1989. “Experiments with the Null-move. Advances in Computer Chess 5, (Ed. D.F. Beal).” *Elsevier Science Publishers, Amsterdam, The Netherlands. ISBN 0-444-87-159-4*.

[12] F., Hsu. 2002. *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion*. Princeton University Press.

[13] F., Hsu, Anantharaman T., Campbell M., and Nowatzyk A. 1990. “A Grandmaster Chess Machine.” *Scientific American*, 44–50.

[14] F., Letouzey. 2005. *Fruit 2.1 Chess Engine - Search_full.cpp*. https://github.com/Warpten/Fruit-2.1/blob/master/src/search_full.cpp.

[15] F., Letouzey, and Fruit Chess Team. n.d. *Fruit Chess Engine - Purchase Page*. http://www.fruitchess.com/purchase.htm.

[16] G., Atkinson. 1998. *Chess and Machine Intuition*. Intellect Books.

[17] G., Goetsch, and Campbell M. S. 1990. "Experiments with the Null-move Heuristic. Computers, Chess, and Cognition (Eds. T.A. Marsland and J. Schaeffer), pp. 159–168." *Springer-Verlag, New York, N.Y., ISBN 0-387-97415- 6/3-540-97415-6.*

[18] J., Berliner H. 1989. "Deep Thought Wins the $10,000 Fredkin Prize." *AI Magazine* 10 (2). https://www.aaai.org/ojs/index.php/aimagazine/article/view/735.

[19] Komodo Chess. 2026. *Dragon by Komodo Chess – World Champion Chess Engine*. Https://komodochess.com/dragon.htm.

[20] Leela Chess Zero Project. 2019. *Leela Chess Zero: Full Elo Graph*. Https://lczero.org.

[21] M., Acher, and Esnault F. 2016. *Large-Scale Analysis of Chess Games with Chess Engines: A Preliminary Report*. https://arxiv.org/abs/1607.04186.

[22] M., Sojka. 2022. "Performance Comparison Between Selected Chess Engines." *Journal of Computer Sciences Institute* 24: 228–35. https://doi.org/10.35784/jcsi.2975.

[23] P., Shaw, Uszkoreit J., and Vaswani A. 2018. "Self-Attention with Relative Position Representations." *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 464–68. https://doi.org/10.18653/v1/N18-2074.

[24] S., Maharj, Polson N., and Turk A. 2022. "Chess AI: Competing Paradigms for Machine Intelligence." *Entropy* 24 (4). https://doi.org/10.3390/e24040550.

[25] Stockfish Developers. 2023. *Training Datasets - Nnue-Pytorch Wiki*. https://github.com/official-stockfish/nnue-pytorch/wiki/Training-datasets.

[26] Stockfish Developers. 2025. *NNUE Documentation*. https://github.com/official-stockfish/nnue-pytorch/blob/master/docs/nnue.md#what-is-nnue.

[27] TalkChess Forum. 2020. "TalkChess Forum Thread." https://www.talkchess.com/forum3/viewtopic.php?f=2&t=75748.