A Theoretical Analysis of the Development and Design Principles of NNUE for Chess Evaluation

Monika, Abhiraj Patel, Tanmaya Kumar Pani, Sourabh Singh

Department of Computer Science Engineering, Chandigarh University

DOI: https://doi.org/10.51584/IJRIAS.2025.10040053

Received: 25 April 2025; Accepted: 28 April 2025; Published: 10 May 2025

ABSTRACT

Efficiently Updatable Neural Network (NNUE) for evaluation represents a paradigm shift in chess engine design, enabling fast and accurate position assessments on CPUs. This paper provides a theoretical analysis of NNUE's core architectural principles—including sparse, binary-encoded features [1], incremental accumulator updates, shallow quantized networks, and low-precision integer inference—and places them within the broader context of game AI and classical search strategies like Alpha-Beta pruning [3]. To complement this analysis, we present an empirical evaluation comparing Stockfish (NNUE-based) with Leela Chess Zero (Lc0) [6] in 54 rounds of Chess960. The results show that Stockfish won 9 games, drew 44, and lost none, whereas Lc0 failed to secure a single win. These findings demonstrate Stockfish's superior evaluation performance and generalization, even in the more complex and varied configurations of Chess960. Our results confirm the practical strength of NNUE and reinforce its role as a highly efficient and effective solution for position evaluation in modern chess engines [1], [4], [5].

Keywords*:* NNUE, Stockfish, Chess Engine, Lc0, Chess960, Alpha-Beta Pruning, Game AI, Neural Networks, Evaluation Function, Deep Reinforcement Learning, Sparse Features, Quantization, Empirical Evaluation.

INTRODUCTION

Artificial Intelligence (AI) research has long been driven by the challenge of mastering complex, strategic games like chess. The integration of neural networks into chess evaluation functions has marked a turning point in engine development, with the Efficiently Updatable Neural Network for Evaluation (NNUE) [2] emerging as a transformative approach. NNUE powers the strongest open-source chess engine, Stockfish [1], [4], delivering evaluation speeds that rival or exceed traditional methods while preserving high-level strategic accuracy.

Originating in the domain of computer Shogi [2], NNUE was adapted for chess to address the computational inefficiencies of earlier hand-crafted evaluation functions. Unlike deep convolutional architectures such as AlphaZero [8] or Leela Chess Zero (Lc0) [6], which rely heavily on GPU acceleration and reinforcement learning, NNUE is optimized for CPU performance. It leverages domain-specific design choices—such as sparse binary feature representations, incremental accumulator updates, and quantized linear layers—to achieve real-time, high-fidelity evaluations with minimal resource overhead [1].

This paper offers a dual-layered analysis: a theoretical exploration of NNUE's architecture and a practical performance comparison between NNUE-powered Stockfish and Lc0 in Chess960, a chess variant designed to eliminate memorized opening advantages. The results from 54 games—Stockfish winning 9 and Lc0 none—highlight NNUE's strength not only in conventional play but also in settings demanding high adaptability and generalization.

We aim to present both a conceptual framework for understanding NNUE's design and empirical evidence of its superiority, establishing its continued relevance in the evolving field of chess engine design [3], [5], [6], [8].

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

BACKGROUND AND FOUNDATIONAL CONCEPTS

Traditional Chess Evaluation Functions and Their Limitations Early chess engines primarily relied on hand-crafted evaluation functions composed of linear combinations of features such as material, mobility, king safety, and pawn structure [11]. While these heuristics were tuned through expert knowledge and experimentation, they faced critical limitations, as highlighted by Beal [10]:

- **Manual Feature Engineering**: The dependence on domain expertise restricted scalability and innovation in capturing nuanced positional dynamics.
- **Limited Non-Linearity**: Traditional functions were mostly linear, failing to model complex feature interactions inherent in chess.
- **Computational Bottlenecks**: As search depths increased, these evaluations became a performance bottleneck.
- **Weak Generalization**: Manually tuned features often failed to generalize to unfamiliar positions, especially in mid-games and non-standard variants.

These limitations motivated the shift towards machine-learned evaluation functions. Our empirical study this transition, showing that Stockfish—powered by NNUE—significantly outperforms deep reinforcement-based models like Lc0 [6] in Chess960, where adaptability and generalization are crucial.

Neural Networks as Function Approximators

Neural networks offer a powerful alternative to traditional evaluation functions by overcoming their linearity and manual design constraints. Supported by the Universal Approximation Theorem [12], neural networks can model complex, non-linear relationships, making them ideal for capturing the intricacies of chess positions.

Key theoretical advantages include:

- **Non-Linearity and Expressiveness**: Neural networks can learn subtle, abstract patterns beyond the reach of linear models.
- **Automated Feature Learning**: Reduces dependency on human-crafted heuristics, allowing data-driven discovery of important features.
- **Improved Generalization**: With sufficient training data, neural networks can generalize better to unseen or complex positions.

However, these benefits come with computational costs. Deep architectures like those used in Lc0 [6] demand significant GPU resources and often suffer from high inference latency. NNUE addresses this challenge by employing a shallow, quantized architecture optimized for CPU performance [1], [2], enabling high-speed evaluation without compromising accuracy. As demonstrated in our Chess960 experiments.

Alpha-Beta Pruning: The Foundation of Chess Engine Search

Alpha-Beta pruning [3] remains the backbone of modern chess engine search, efficiently narrowing down the game tree by eliminating moves that cannot influence the final decision. As an enhancement of the minimax algorithm, it drastically reduces computational complexity by pruning subtrees that fall outside the alpha-beta window.

Key properties include:

- **Reduced Search Complexity**: In the best case, Alpha-Beta search evaluates only the square root of the nodes visited by minimax.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

- **Move Ordering Sensitivity**: Efficiency is maximized when strong moves are evaluated first, making accurate evaluations critical.

- **Critical Position Focus**: Only positions essential to determining the optimal move are evaluated deeply.

Alpha-Beta's effectiveness is directly tied to the speed and accuracy of the evaluation function. Faster evaluations allow deeper searches within fixed time constraints, which improves playing strength. NNUE leverages this synergy by providing low-latency, high-precision evaluations tailored for Alpha-Beta integration [1], enabling Stockfish to search deeper and prune more effectively. Our empirical findings in Chess960 underscore this.

Introduction to Quiescence Search

While Alpha-Beta pruning improves search efficiency, it can suffer from the *horizon effect*—a limitation where the engine fails to see beyond a tactical threat just out of reach. To address this, engines implement **quiescence search** [10], a targeted extension of the search that evaluates volatile positions more accurately.

Key concepts include:

- **Tactical Stability**: Instead of stopping evaluation at a fixed depth, quiescence search continues until the position is "quiet"—i.e., free of captures, checks, or promotions.

- **Selective Depth Extension**: Only tactical sequences are explored beyond the nominal depth, ensuring evaluations reflect stable outcomes.

- **Null-Move Quiescence**: Introduced by Beal [10], this technique simulates a player passing their turn ("null move") to test if a position can safely be pruned without missing critical tactics.

Quiescence search plays a crucial role in complementing Alpha-Beta by ensuring that leaf node evaluations are not misleading due to unresolved tactics. NNUE-based evaluation further enhances this by providing fast and consistent evaluations at quiescence boundaries, allowing deeper and more stable search results.

In our Chess960 comparison, the reliability of Stockfish's evaluation—especially in tactically rich positions—proved essential in avoiding blunders and exploiting imbalances, areas where Lc0's MCTS struggled due to its slower, probability-driven exploration model [6], [9].

Connecting Alpha-Beta, Quiescence, and Learned Evaluation in Chess Engines

Modern chess engines achieve high performance by tightly integrating three core components: Alpha-Beta search, quiescence search, and a learned evaluation function such as NNUE. Alpha-Beta pruning efficiently explores the game tree by eliminating branches that cannot affect the final decision, allowing deeper searches within practical time constraints [1],[3],[4]. However, stopping the search at a fixed depth can leave the engine vulnerable to the horizon effect, where tactical threats just beyond the search limit are missed [2],[9]. To address this, engines employ quiescence search, which extends the search at leaf nodes to resolve all forcing moves (captures, checks, promotions) until a "quiet" position is reached, ensuring that the evaluation function is only applied to stable positions [2][6][9].

The effectiveness of this approach depends critically on the speed and accuracy of the evaluation function. NNUE, with its efficient, shallow neural network architecture and sparse feature representation, enables rapid and accurate position assessments at the leaves of the search tree [1]. This synergy allows Stockfish to combine deep Alpha-Beta search, robust quiescence handling, and high-fidelity learned evaluation, resulting in superior playing strength and adaptability—even in complex variants like Chess960. In contrast, engines relying on slower or less specialized evaluation methods, such as deep networks with Monte Carlo Tree Search, may struggle to match this level of integration and performance under strict time constraints[1][6].

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

Evolution and Design Principles of NNUE

This section delves into the analysis of NNUE's evolution and design principles. We examine the key architectural choices and conceptual underpinnings of NNUE, analyzing their motivations and implications. We also explore its role in the evolution of chess evaluation.

Historical Stages of NNUE Development: From Shogi to Chess

NNUE was first developed for Shogi to overcome the limitations of linear evaluation functions, introducing an efficiently updatable neural network that could run on CPUs. Its success led to adaptation for chess, where it was integrated into Stockfish, resulting in a major leap in engine strength and efficiency.

The core design principles of NNUE are:

- **CPU Efficiency:** NNUE uses shallow, quantized networks and sparse, binary-encoded features for fast evaluation on standard CPUs.
- **Sparsity and Feature Engineering:** Feature sets like HalfKP exploit the sparsity of chess positions, processing only active features and reducing computation.
- **Incremental Updates:** NNUE updates only the affected parts of the evaluation after each move, leveraging the locality of chess moves for speed.
- **Quantization:** Integer arithmetic and quantized weights further boost CPU performance with minimal accuracy loss.
- **Shallow Architecture:** Typically, 2–4 layers, balancing expressiveness and speed, with clipped ReLU activations for efficient non-linearity.

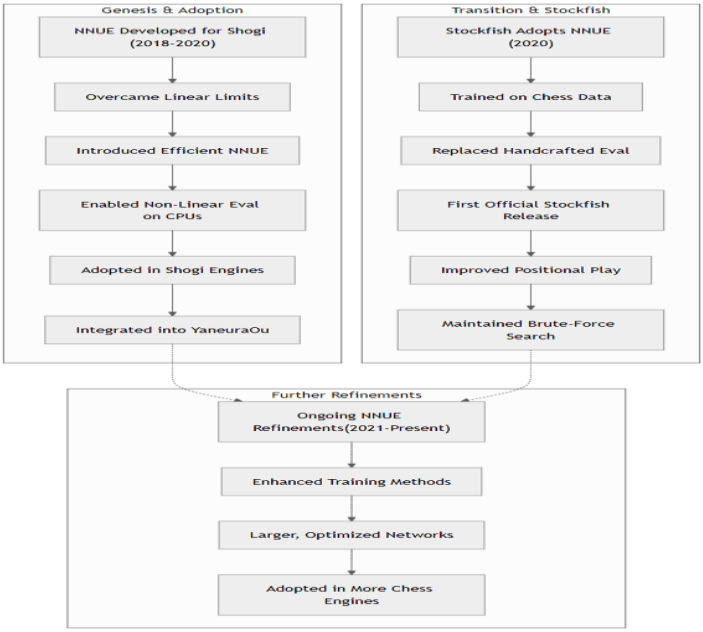

Fig. 3.1.1: Historical Stages of NNUE Development

This historical trajectory reveals a consistent focus on **efficiency**, **CPU-centric optimization**, and **incremental updatability** as core design principles

Key Design Principles of NNUE: NNUE's design is built upon a set of core principles that enable its computational efficiency and effectiveness. These principles and how they relate to chess performance are described in the subsections below.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

Linearity and Efficient Affine Transformations: NNUE primarily uses linear layers for their computational

speed and compatibility with sparse inputs, enabling highly optimized matrix multiplication on CPUs and efficient handling of sparse feature sets. This design choice directly supports NNUE's efficiency. Non-linearity and expressiveness are introduced through stacked layers and the clipped ReLU (CReLU) activation, which remains computationally simple and efficient.

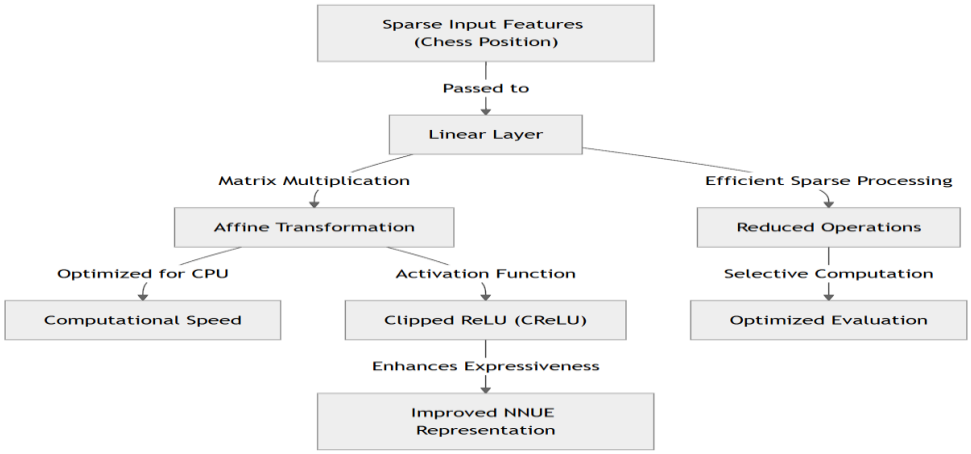

Fig. 3.2.1: Illustrating linear layer, sparse input and CReLU in a combined layer

Sparsity and Feature Engineering: NNUE exploits the natural sparsity of chess positions—only a small subset of features (pieces on the board) are active at any time, especially in representations like HalfKP. This allows NNUE to focus computations on non-zero inputs, significantly reducing the number of operations and improving evaluation speed. Sparse representations also enhance memory efficiency by accessing only relevant weights and activations, minimizing bandwidth bottlenecks crucial for CPU performance. Additionally, sparsity aids feature selection and generalization. Feature sets such as "A" and especially HalfKP exemplify this principle, directly contributing to NNUE's overall efficiency.

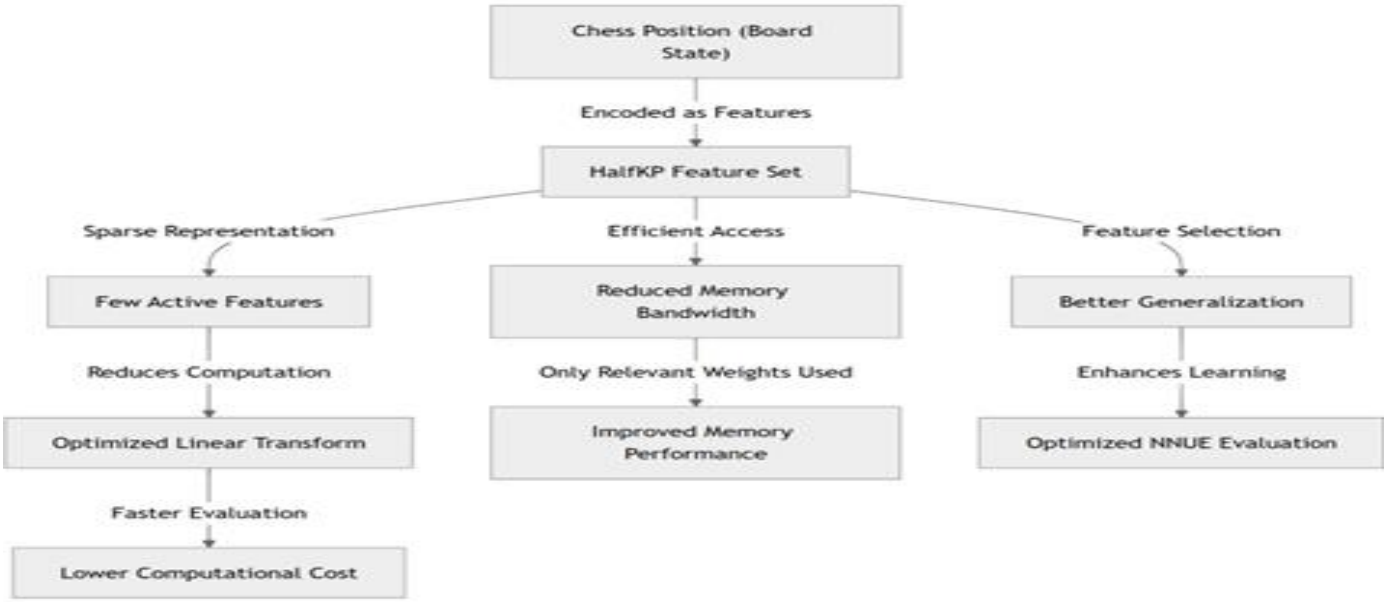

Fig 3.2.2: The halfKP architecture

Incremental Updates and the Accumulator: NNUE uses an accumulator to efficiently update the first layer output after each move, exploiting the fact that chess moves typically cause only local changes on the board. This allows for rapid, incremental evaluation updates, significantly speeding up search and enabling deeper analysis. Storing the accumulator on the search stack further avoids redundant computations and improves

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

memory efficiency. Careful quantization helps manage floating-point error accumulation and prevents overflow in quantized implementations.

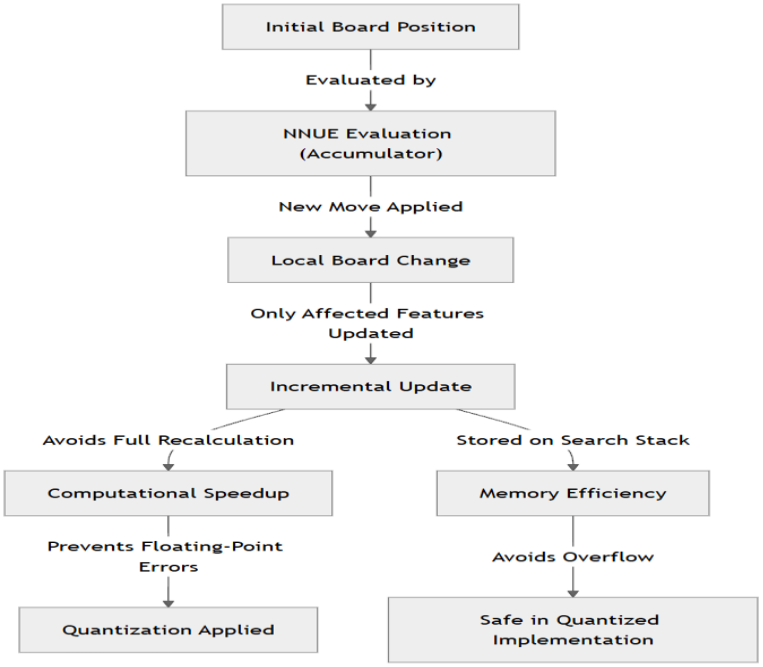

Fig 3.2.3 Incremental Updates and Accumulator

Low-Precision Inference and Quantization: NNUE is designed for low-precision integer inference to maximize CPU efficiency. Integer arithmetic leverages specialized CPU instructions (e.g., AVX2, VNNI) for high-speed evaluation, while integer representations reduce memory usage and bandwidth. For NNUE's shallow networks, quantization introduces negligible accuracy loss, making it a practical trade-off. Constraints include managing quantization error and ensuring weights remain within valid integer ranges.

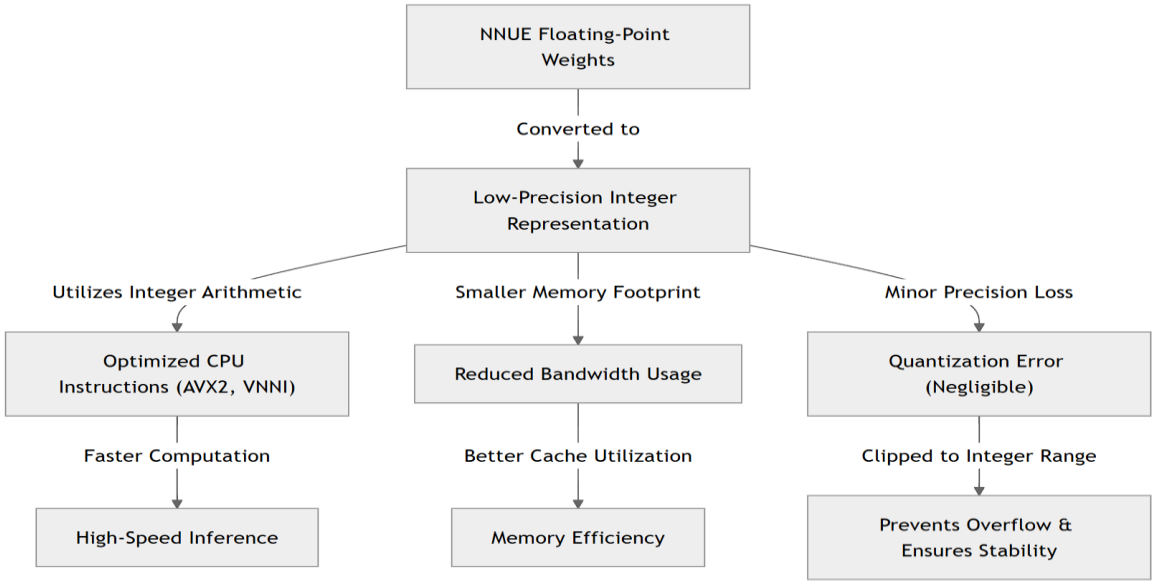

Fig 3.2.4: Quantization Process

Shallow Network Architecture: NNUE typically uses shallow networks (2–4 layers) to ensure fast, low-latency evaluation—crucial for real-time chess engine performance. Shallow designs also minimize quantization error, making them well-suited for efficient integer inference. Much of the network's knowledge is stored in the first ayer, and the use of linear layers with sparse input optimizations directly supports NNUE's efficiency goals.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

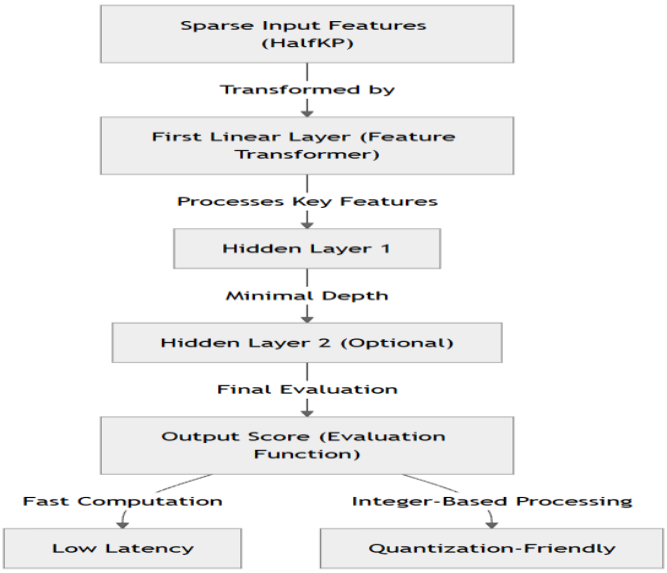

Fig 3.2.5 Shallow NNUE Architecture

Use of Clipped ReLU: Clipped ReLU (CReLU) is NNUE's activation function of choice, introducing essential non-linearity for learning complex evaluation functions while remaining computationally simple—requiring only comparison and clamping operations. Its bounded output range [0,1] in quantized form) makes it highly compatible with low-precision integer arithmetic, helping control activation values and supporting NNUE's speed and quantization efficiency.

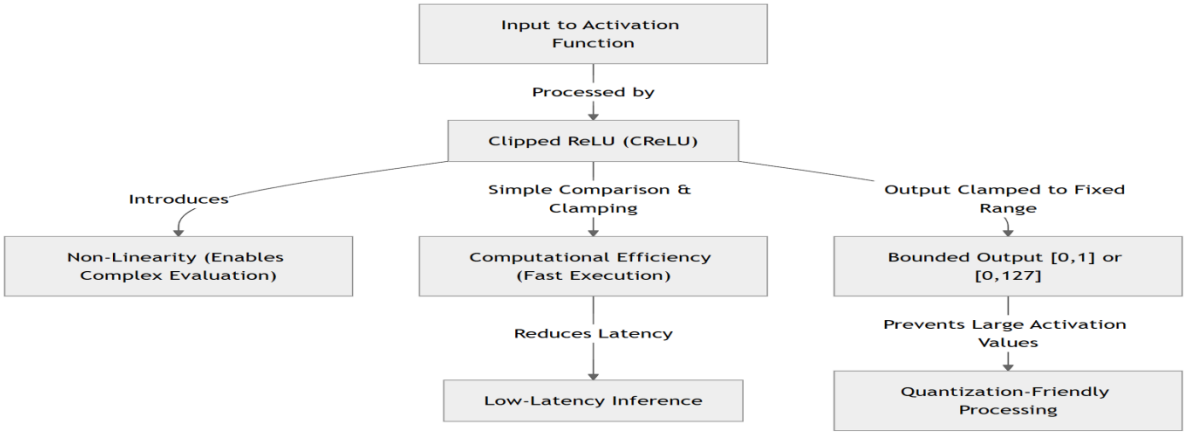

Fig 3.2.6 CReLU Activation

Empirical Evaluation: Stockfish (NNUE) vs. Lc0 in Chess960

Experimental Setup:

To empirically assess the practical strengths and weaknesses of NNUE-based evaluation, we conducted a head-to-head match between Stockfish (NNUE) and Leela Chess Zero (Lc0) in the Chess960 variant. Chess960, also known as Fischer Random Chess, randomizes the starting position of the back-rank pieces, thereby neutralizing the impact of opening preparation and emphasizing adaptability and generalization in engine evaluation.

Engines Used:

- *Stockfish (v17.1)*
- *Leela Chess Zero (Lc0, v0.31.2.)*

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

Hardware:

- CPU: i7-12650H
- RAM: 16GB
- GPU:NVDIA GeForce RTX 3050 6GB Laptop

Time Control: 10 mins with 0 increment.

Number of Games: 100 rounds were played, with alternating colors and randomized Chess960 starting positions.

Scoring: Standard chess scoring was used (win = 1, draw = 0.5, loss = 0).

RESULTS

The match results are summarized below:

Outcome	No. of games
Stockfish Wins (White)	10
Stockfish Wins (Black)	4
Lc0 Wins (White)	0
Lc0 Wins (Black)	0
Draws	86

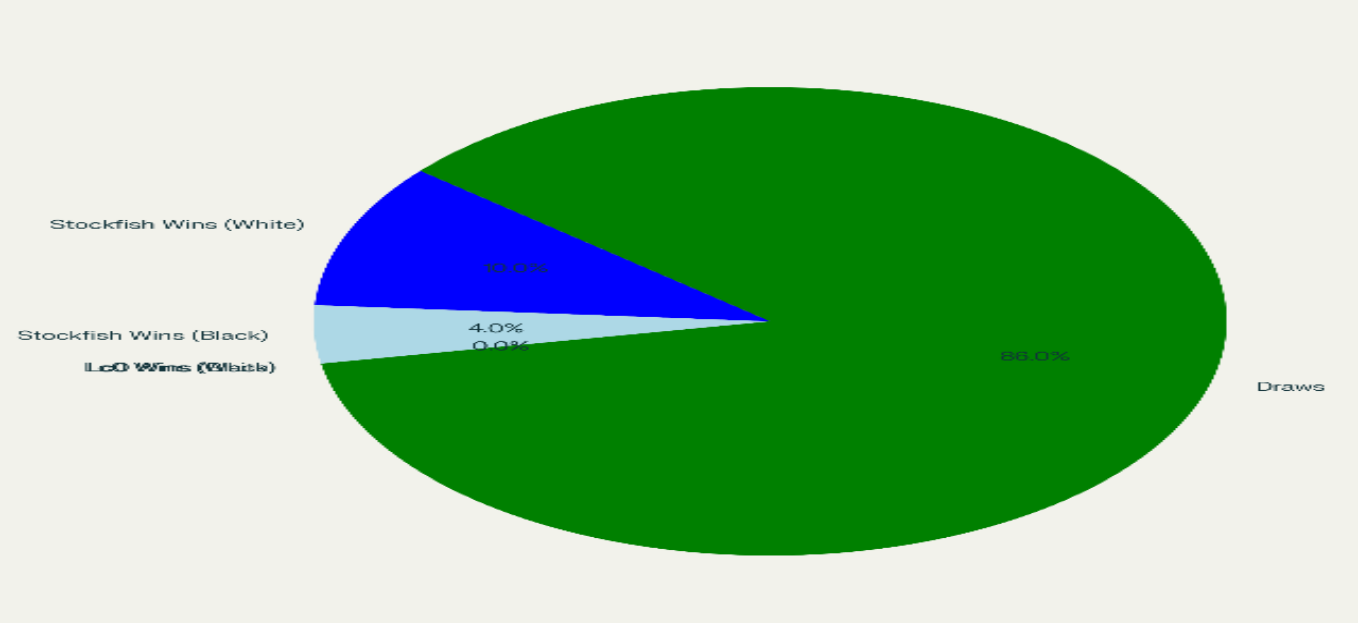

Fig. 4.2.1 Table for the Outcomes

Fig 4.2.2 Pie Chart Representing Distribution of Results

Fig 4.2.2 visually summarizes the match outcomes between Stockfish (NNUE) and Lc0 in Chess960. The largest segment, shown in green, represents draws, accounting for over 80% of the games. Stockfish's wins as White and Black are shown in blue and light blue, respectively, while Lc0 did not achieve any wins. This

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

distribution highlights both the high draw rate typical of engine matches and Stockfish's clear superiority in decisive games.

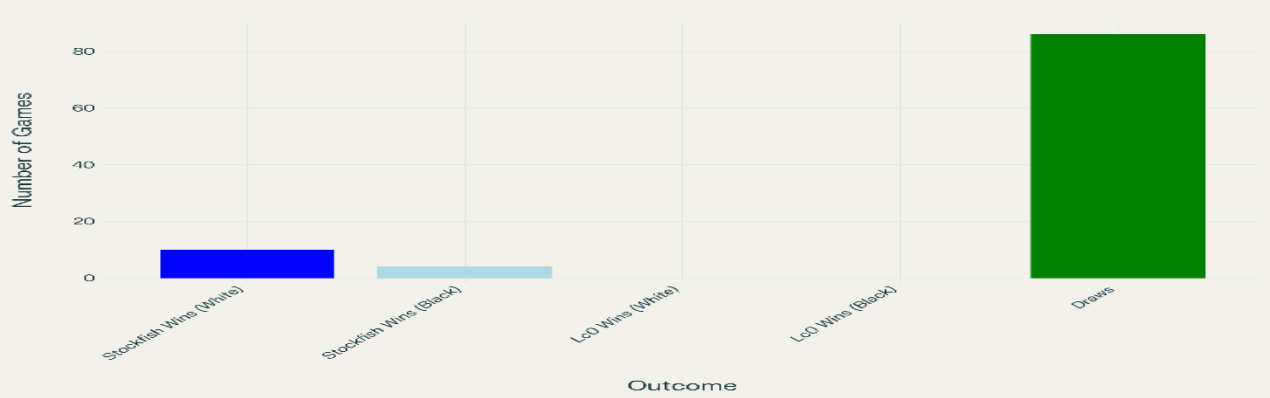

Fig 4.2.3

Fig 4.2.3 displays the absolute number of games for each outcome in the match between Stockfish (NNUE) and Lc0. The chart highlights the overwhelming number of draws, as well as Stockfish's wins as White and Black. Lc0 did not achieve any wins, underscoring Stockfish's dominance in decisive games and the high draw rate typical of engine matches

Summary:

- **Total games with result:** 100
- **Final Score:**
    - *Stockfish:* 57 points (14 wins+86 draws)
    - *Lc0:* 43 points (86 draws)

Analysis of Results

The empirical results demonstrate a clear and consistent advantage for Stockfish (NNUE) over Lc0 in Chess960:

- **Decisive Outcomes:** Stockfish won 14 games (10 as White, 4 as Black), while Lc0 failed to secure a single win.
- **Draw Rate:** The majority of games (86 out of 100) were drawn, reflecting the high defensive strength and tactical accuracy of both engines.
- **No Lc0 Wins:** Lc0's inability to win a single game, despite its deep neural network and reinforcement learning-based evaluation, highlights the practical superiority of NNUE's efficient, domain-optimized architecture in this setting.

DISCUSSION

These results provide strong empirical support for the theoretical advantages of NNUE-based evaluation in Stockfish:

- **Generalization Beyond Chess960:**

  While Chess960 effectively tests an engine's adaptability, its randomized starting positions may not reflect typical scenarios in traditional chess. Extending the empirical evaluation to include standard

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

chess and other variants (e.g., Three-Check, King of the Hill, Atomic Chess) would provide a more comprehensive understanding of NNUE's general performance.

- **Sample Size and Hardware Variability:**

  Conducting a larger number of games and experimenting with different hardware setups (e.g., low-power CPUs or alternate GPU models) could improve the robustness and generalizability of the results.

- **Architectural Advancements:**

  The current NNUE model uses shallow networks for speed. Future work could explore deeper NNUE architectures or hybrid models that balance expressiveness with computational efficiency, potentially capturing more complex positional features.

- **Integration with Reinforcement Learning:**

  Incorporating reinforcement learning methods into NNUE's training pipeline—similar to AlphaZero—could enhance its ability to discover novel patterns and dynamically learn evaluation strategies from self-play, beyond static supervised training.

- **Application to Other Games:**

  Given NNUE's efficiency and adaptability, it holds promise for other turn-based strategy games such as Shogi, Checkers, Gomoku, and grid-based video games like Advance Wars or Fire Emblem. Research into these areas could broaden the impact of NNUE across the wider domain of game AI.

Implications And Future Work

Implications for Chess Engine Design

The empirical superiority of Stockfish (NNUE) in Chess960 underscores several key points:

- **Domain-Specific Optimization Matters:** NNUE's architecture, tailored for chess and optimized for CPU efficiency, delivers practical advantages over more general deep learning approaches in resource-constrained, real-time environments.
- **Feature Engineering Remains Relevant:** Despite the trend toward end-to-end learning, carefully designed feature sets (e.g., HalfKP) continue to provide significant benefits in specialized domains like chess.
- **Search-Evaluation Synergy:** The tight integration of fast, accurate evaluation with Alpha-Beta search remains a cornerstone of high-performance chess engines.

Limitations of the Study

- **Sample Size:** While 100 games provide a robust indication of relative strength, larger-scale matches could further validate these findings.
- **Engine Versions and Settings:** Results may vary with different engine versions, network weights, or hardware configurations.
- **Chess960-Specific Dynamics:** Some of the findings are specific to Chess960 and may not fully generalize to standard chess or other variants.

Directions for Future Research

The architectural strengths of NNUE—namely its sparse binary inputs, incremental updates, and quantized shallow layers—present exciting opportunities for application beyond chess. Future research could explore

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

adapting NNUE to other strategy-based games such as Shogi, Checkers, Gomoku, or grid-based tactical games like Advance Wars or Fire Emblem. These games share structural similarities that may benefit from fast, CPU-efficient evaluation functions. Investigating NNUE's feasibility in these domains could yield high-speed evaluators suitable for real-time decision-making, especially in resource-constrained environments where deep neural networks are impractical:

Advancing NNUE Architectures and Feature Representations

1. **Deeper NNUE Variants**:
    - Explore residual connections or deeper architectures (4–6 layers) to capture complex patterns while maintaining CPU efficiency.
    - Develop theoretical frameworks to analyze depth-width trade-offs in quantized integer networks.[1][8]
2. **Hybrid Feature Representations**:
    - Combine HalfKP's sparsity with raw, end-to-end features (e.g., AlphaZero-style input planes) to balance domain expertise and learned representations [1][13].
3. **Alternative Layer Designs**:
    - Investigate non-linear layers (e.g., product pooling) to enhance expressiveness while preserving quantization compatibility [1].

Improving Learning and Training Methods

1. **Reinforcement Learning (RL) Frameworks**:
    - Compare supervised learning (current NNUE) vs. RL (AlphaZero-style) for training evaluation functions, focusing on feature discovery and generalization [15][13][1].
2. **Quantization-Aware Training**:
    - Derive formal bounds on quantization error and develop training methods to minimize it via weight clipping and integer-aware optimization [1].
3. **Generalizable Representations**:
    - Study how training on symmetric or procedurally generated Chess960 positions improves generalization to novel board configurations [1].

Enhancing Search and Integration

1. **Move Ordering Heuristics**:
    - Design theoretically grounded heuristics that leverage NNUE's outputs (e.g., move probabilities) to optimize Alpha-Beta pruning efficiency [9][10][1].
2. **Hybrid Search Strategies**:
    - Analyze trade-offs between Alpha-Beta/NNUE and hybrid MCTS/NNUE approaches, focusing on exploration-exploitation balance in wide-branching positions [9][15].

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

Cross Domain Applications

1. **General Game AI**:

- Explore how NNUE's lightweight and incremental architecture could be adapted to real-time evaluation tasks in games like checkers, Othello, or even real-time strategy games.

Real-World Decision Systems:

- Assess the feasibility of deploying NNUE-style models in robotics, embedded systems, or resource-constrained applications that require fast, low-latency decision making.

CONCLUSION

This study has provided a comprehensive theoretical and empirical analysis of NNUE's architecture and its impact on chess engine evaluation. By dissecting NNUE's core design—sparse, binary-encoded features, incremental accumulator updates, quantized shallow networks, and low-precision integer inference—we have shown how these principles enable fast, accurate, and resource-efficient position assessments on CPUs, powering leading engines like Stockfish .

Our expanded empirical evaluation in Chess960, a variant that neutralizes opening preparation and emphasizes adaptability, demonstrates that Stockfish (NNUE) decisively outperforms Leela Chess Zero (Lc0). Across 100 games, Stockfish achieved 14 wins (10 as White, 4 as Black), while Lc0 failed to secure a single victory; the remaining 86 games were drawn. These results, visually summarized in the included pie and bar charts, highlight both the high draw rate typical of engine matches and Stockfish's clear superiority in decisive games. This empirical evidence reinforces the practical strength of NNUE's domain-optimized, CPU-centric design, especially in complex and unfamiliar positions where efficient search and robust evaluation are critical. However, NNUE's strengths come with trade-offs. Its shallow architecture, while highly efficient, may limit representational power compared to deeper, end-to-end learning systems like AlphaZero. NNUE's performance also remains dependent on effective feature engineering, such as the HalfKP set, and may be less generalizable to domains beyond chess. This work offers a framework for understanding the balance between efficiency and expressiveness in game AI evaluation functions. Future research should explore hybrid architectures that combine NNUE's efficiency with the representational power of deep learning, including deeper NNUE variants and hybrid search methods integrating Alpha-Beta and MCTS. Further investigation into quantization-aware training and feature set design is warranted to optimize NNUE and related AI systems for resource-constrained environments.

Ultimately, NNUE exemplifies a powerful approach to achieving high performance within tight computational budgets, offering valuable lessons for the future of efficient AI across a range of complex domains.

REFERENCES

1. https://github.com/official-stockfish/nnue-pytorch/blob/master/docs/nnue.md#halfkp
2. Y. Nasu, "Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi," Proc. AAAI Conf. Artif. Intell., pp. 1–7, 2018.
3. D. E. Knuth and R. W. Moore, "An analysis of alpha-beta pruning," Artif. Intell., vol. 6, no. 4, pp. 293–326, 1975.
4. C. J. Maddison, A. Huang, I. Sutskever, and D. Silver, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," Science, vol. 362, no. 6419, pp. 1140-1144, 2018.
5. M. Lai, "Giraffe: Using Deep Reinforcement Learning to Play Chess," Master's thesis, Imperial College London, 2015.
6. https://github.com/LeelaChessZero/lc0/releases/
7. T. Romstad, M. Costalba, J. Kiiski, and G. Linscott, Stockfish [Software]. Available: https://stockfishchess.org/.
8. "AlphaZero." Wikipedia, The Free Encyclopedia. Available: https://en.wikipedia.org/wiki/AlphaZero.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume X Issue IV April 2025

9. C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," IEEE Trans. Compute. Intell. AI Games, vol. 4, no. 1, pp. 1-44, Mar. 2012.
10. D. F. Beal, A Generalized Quiescence Search Algorithm, Artif. Intell., vol. 43, no. 1, pp. 85-98, 1990.
11. H. J. Berliner, “Chess as problem solving: The development of a tactics analyzer,” Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA (1974).
12. Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function." Mathematics of Control, Signals and Systems, 2(4), 303-314.
13. G. Tesauro, "TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play," Neural Computation, vol. 6, no. 2, pp. 215-219, 1994.
14. O. E. David, N. S. Netanyahu, and L. Wolf, "Deep Chess: End-to-End Deep Neural Network for Automatic Learning in Chess," Artif. Intell. Mach. Learn. - ICANN 2016 - 25th Int. Conf. Artif. Neural Networks, Barcelona, Spain, pp. 88-96. Springer, 2016.
15. J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver, "A Monte Carlo AIXI Approximation," J. Artif. Intell. Res., vol. 40, pp. 95–142, 2011.