Efficiency vs Generality: Analyzing the Trade-offs Between NNUE and AlphaZero-like Deep Networks

Name: Swoyam Pokharel
Student Number: 2431342
Tutor: Jeshmi Rajak

Submitted On: January 15, 2026

Abstract

This review aims to examine the fundamental trade-offs between NNUE and AlphaZero-style architectures in chess AI, analyzing their performance characteristics, computational requirements, and accessibility barriers. Based on the analysis of recent papers in this domain, this paper demonstrates how domain-optimized efficiency differs with general-purpose learning, and aims to inform architectural decisions for practical chess engine development targeting consumer hardware deployment.

Contents

1 · Introduction

Context

Chess has long served as a benchmark for the field of artificial intelligence, from Shannon's early works establishing chess as a computational problem to Deep Blue's victory in 1997. The use of artificial intelligence in the game of chess has come a long way, and after 2017 we began to see a significant paradigm shift when AlphaZero demonstrated that neural networks trained with just self-play could defeat classical engines like Stockfish. However, with the integration of NNUE (Efficiently Updatable Neural Networks) in Stockfish since 2020, traditional exhaustive search-based engines reclaimed dominance while still maintaining CPU efficiency. Thus, today's chess AI is defined by two competing philosophies: domain-optimized efficiency versus general-purpose learning.

The Efficiency vs. Generality Debate

NNUE-based architectures achieve fast evaluation through small networks and incremental updates, allowing them to run efficiently on consumer hardware. In contrast, AlphaZero-styled systems use deep neural networks with Monte Carlo Tree Search (MCTS), inherently requiring massive computational resources. However, their architectural differences come with different behaviors. Stockfish (NNUE) can search millions of positions per second on a CPU, while LCZero (LeelaChessZero; an open-sourced AlphaZero-like engine) evaluates mere thousands but with deeper "understanding."

NNUE optimizes specifically for chess through handcrafted input features and efficient update mechanisms, while AlphaZero/LCZero-style systems learn entirely from self-play through domain-agnostic neural architectures. This choice bleeds into trade-offs on playing strength and computational accessibility, and generality that extend beyond chess. Are systems better off being tailored to specific problems with domain knowledge and optimizations, or should they be agnostic, learning via general principles from scratch that transfer across domains?

Aims & Objectives

This review analyzes the trade-offs between NNUE and AlphaZero-like architectures across the three sectors: performance, computational efficiency, and flexibility using recent comparative studies and implementation papers. This topic was chosen to inform the architectural decisions for a custom chess engine built in Rust, currently under development as a final year project.

2 · Literature Review

2.1 · Neural Networks for Chess (Klein, 2022)

Methodology

Klein provides analysis and review of chess AI evolution, combining technical explanations with a practical implementation of a simplified "HexapawnZero" engine (Klein, 2022, p.18). The book examines multiple architectures including Stockfish NNUE, AlphaZero, Leela Chess Zero, and Maia, transitioning from classical alpha-beta search with handcrafted evaluation functions to neural network based systems.

Findings

Klein documents that AlphaZero, using 19-39 residual blocks and trained on 48 TPUs, reached a peak rating of 5185 Elo by learning entirely through self-play reinforcement learning (Klein, 2022, p.162). Whereas, NNUE's integration into Stockfish 12 added approximately 80 Elo points while maintaining CPU efficiency through 8-bit integer mathematics and SIMD instructions like `VPADDW` and `VPSUBW` (Klein, 2022, p.193, p.194, p.209). The HalfKP input representation uses 81,920 bits to enable incremental updates, recomputing only the difference when pieces move rather than the entire network (Klein, 2022, p.197, p.209). This allows NNUE to maintain search speeds of 4.6 million nodes per second compared to Stockfish 8′s 7.5 million nps, while AlphaZero operates at only 80,000 nps (Klein, 2022, p.212, p.193).

Klein also discusses Maia, a human centric engine trained on 12 million Lichess games that achieves over 50% move-matching accuracy for 1100-rated players compared to Stockfish's 35%, demonstrating that supervised learning from human games excels at mimicking human play while reinforcement learning achieves superhuman strength (Klein, 2022, p.222). The book emphasizes that neural networks perform poorly on data completely out of their training domain, highlighting a key limitation of learned systems. (Klein, 2022, p.47)

Takeaway

Klein's analysis shows that the choice between NNUE and AlphaZero boils down to what you optimize for. NNUE is built for efficiency, using chess-specific tricks to run fast on regular hardware. AlphaZero goes for generality, using the same approach for Chess, Shogi, and Go, but needs massive resources (48 TPUs) to train (Klein, 2022, p.160).

2.2 · Chess AI: Competing Paradigms for Machine Intelligence (Maharaj, Polson and Turk, 2022)

Methodology

Maharaj et al. conducted an experimental comparison between Stockfish 14 (alpha-beta search) and Leela Chess Zero network 609950 (neural MCTS) using Plaskett's Puzzle (Maharaj, Polson and Turk, 2022, p. 2), a famous endgame study requiring 15-move depth and a counterintuitive underpromotion. The engines were tested over five trials to account for multithreading randomness, with results averaged to ensure reliability (Maharaj, Polson and Turk, 2022, p. 6).

Findings

Their findings show that Stockfish successfully solved the corrected puzzle at a depth of 40, correctly identifying a forced mate in 29 half-moves (Maharaj, Polson and Turk, 2022, p. 8). However, LCZero failed to find the solution even after analyzing 60 million nodes (Maharaj, Polson and Turk, 2022, p. 8). They found that a surprising 92.4% of LCZero's search nodes followed an inferior move because its policy head assigned it 15.75% probability compared to only 7.40% for the winning move (Maharaj, Polson and Turk, 2022, pp. 8-9). Their experiment gave LCZero 34 times more computational power than Stockfish relative to standard tournament ratios, yet it still failed the puzzle (Maharaj, Polson and Turk, 2022, p. 9).

Stockfish reached 1.5×10^8 nodes per second while LCZero reached only 1.4×10^5 nps (Maharaj, Polson and Turk, 2022, p. 9). However, when given the first move, LCZero found the mate after 5.5 million nodes while Stockfish required 500 million nodes, showing LCZero's efficiency when its intuition is correct (Maharaj, Polson and Turk, 2022, p. 9). This paper effectively highlights a fundamental trade-off in chess AI design: NNUE-based engines like Stockfish are better at tactical calculation, making them effective for edge cases. AlphaZero-style systems like LCZero rely on pattern matching, which is more generalizable across other games like shogi and Go, but fail at edge cases when the policy network misjudges position complexity (Maharaj, Polson and Turk, 2022, p. 10).

Takeaway

Ultimately, Maharaj et al. conclude that Stockfish solves the puzzle with much greater efficiency than LCZero (Maharaj, Polson and Turk, 2022, p. 1).

2.3 · Stockfish or Leela Chess Zero? A Comparison Against Endgame Tablebases (Sadmine, Husna and Muller, 2023)

Methodology

Sadmine et al. conducted an analysis comparing Stockfish 15.1 and LC0 0.29.0 (both rated at 2850 Elo) against perfect play defined by Syzygy endgame tablebases (Sadmine, Husna and Muller, 2023, p.2; 2023, p.4) . The study evaluated engine moves across 3-piece, 4-piece, and 5-piece endgames, measuring errors as game-theoretic outcome changes (win to draw, draw to loss, etc.) (Sadmine, Husna and Muller, 2023, p.4). Tests were primarily conducted on raw policy networks without search to isolate learning ability, though performance with a small search budget of 400 nodes was also examined (Sadmine, Husna and Muller, 2023, p.7).

Finding

Their findings reveal Stockfish's policy is superior in 3-piece endgames, demonstrating stronger tactical calculation in simpler positions (Sadmine, Husna and Muller, 2023, p.9). However, LCZero produced fewer mistakes in most 4-piece endgames, with LC0 making 1.32% errors in winning positions compared to Stockfish's 1.47% (Sadmine, Husna and Muller, 2023, p.4).

The study found that predicting wins is easier for Stockfish, whereas predicting draws is easier for LC0, with LC0 performing better in drawing positions across nearly all 4-piece tablebases (Sadmine, Husna and Muller, 2023, p.9). They also found that, when LC0 makes a mistake, its evaluation of the position remains more accurate (further from zero in centipawns) than Stockfish's, suggesting a better understanding of positional nuance even when choosing non optimal moves (Sadmine, Husna and Muller, 2023, p.5). The authors also analyzed specific scenarios such as when the opponent's last pawn is under attack, finding that LCZero made fewer mistakes than Stockfish in these complex tactical-positional situations (Sadmine, Husna and Muller, 2023, p.9).

Takeaway

This paper demonstrates that the efficiency versus generality trade-off is more pronounced in certain game phases: NNUE-based engines excel at simpler tactical endgames through exhaustive calculation, while AlphaZero-style systems show superior positional "feel" in complex, endgames.

2.4 · Chess Engine Inspired by AlphaZero (Krakovský and Liberda, 2025)

Methodology

Krakovský and Liberda attempted to build a self-play reinforcement learning chess engine using Python, python-chess, and PyTorch (Krakovský and Liberda, 2025, p.4). Their implementation used a 2-layer residual blocks; which, for reference, AlphaZero used 19-blocks (Krakovský and Liberda, 2025, p.4). Their study aimed to replicate the AlphaZero methodology at a smaller scale to understand the practical challenges of implementing such systems.

Findings

Their findings reveal the computational gap between theory and practice. The engine only played 1,200 games over 1 GPU-hour, compared to AlphaZero's 44 million games over 41 TPU-years (Krakovský and Liberda, 2025, p.7). Their implementation faced big performance bottlenecks, with 40-50% of runtime spent in MCTS calculations due to Python object conversions (Krakovský and Liberda, 2025, p.7). GPU speedup was only 2x over CPU because of a lack of inference batching, and reaching AlphaZero's scale in this Python environment would take approximately 510 days (Krakovský and Liberda, 2025, p.8, p.9).

The engine reliably learned to draw but failed to learn how to win, with the model preferring draws by repetition or the 50-move rule (Krakovský and Liberda, 2025, p.10). The authors note that insufficient compute resources and the choice of Python as an implementation language severely limited progress, suggesting that C++ or Rust would be necessary for efficient search and simulation (Krakovský and Liberda, 2025, p.10).

Takeaway

This paper demonstrates the extreme accessibility barrier of the AlphaZero paradigm. While the approach allows generality, the computational requirements place it far beyond the reach of individual developers or small teams. The 510 day training estimate highlights that the benefit of generality comes at a big cost, making AlphaZero-style systems practical only for organizations with massive computational resources.

2.5 · Implementing the Chess Engine using NNUE with Nega-Max Algorithm (Chitale et al., 2024)

Methodology

Chitale et al. conducted an implementation and comparison using a custom engine called "Stockdory," which integrated NNUE with the Nega-Max algorithm (Chitale et al., 2024, p.2). The study compared Stockdory against Stockfish by analyzing two historical games: Bogoljubov vs Alekhine (1922) and Lasker vs Bauer (1889) (Chitale et al., 2024, p.4).

Findings

Their findings reveal performance gaps between the custom implementation and mature engines. In the 1922 positional game, Stockfish achieved 80.19% accuracy in matching game moves, while Stockdory achieved only 45.28% (Chitale et al., 2024, p.4). Stockfish also evaluated this game faster, completing analysis in 6 minutes and 50 seconds compared to Stockdory's 9 minutes and 3 seconds (Chitale et al., 2024, p.4). However, in the 1889 psychological game featuring Lasker's counterintuitive style, Stockdory slightly outperformed Stockfish, matching 52% of moves compared to Stockfish's 48%, and completed evaluation in 6 minutes and 28 seconds versus Stockfish's 7 minutes and 5 seconds (Chitale et al., 2024, p.5).

Takeaway

This paper illustrates the accessibility advantage of NNUE-based architectures. While Stockdory didn't reach the same level of performance as Stockfish, the authors did not mention computational barriers or scalability issues during implementation. This implies the relative ease and accessibility of building a functional NNUE based engine, and also proves how domain-optimized efficiency translates into allowing individuals and small teams to use NNUE.

2.6 · Monte Carlo Tree Search: A review of recent modifications and applications (Świechowski et al., 2023)

Methodology

Świechowski et al. conducted a literature review of 240 papers published between 2012 and 2022, examining the evolution of MCTS game-playing to general optimization (Świechowski et al., 2023, p.2, p.3). They analyzed papers from leading journals and conferences by filtering them from keywords like "UCT," "Bandit-Based," and "General Game Playing," and focusing on modifications addressing computational constraints in practical applications (Świechowski et al., 2023, p.2, p.4).

Findings

Their review finds MCTS to be an aheuristic algorithm requiring only environment rules rather than domain knowledge, with four main phases: Selection, Expansion, Simulation, and Backpropagation (Świechowski et al., 2023, p.2, p.8). They also highlight AlphaGo's victory over Lee Sedol as "second major breakthrough," that demonstrates that MCTS combined with deep reinforcement learning can infact achieve great performance (Świechowski et al., 2023, p. 26). However, they also state that vanilla MCTS fails in tactical traps and high branching factor games like StarCraft with 10^{50} possible actions (Świechowski et al., 2023, p.7, p.16, p.55).

The survey highlights three parallelization strategies: Leaf parallelization for simultaneous playouts, Root parallelization using multiple independent trees, and Tree parallelization with shared structures using "Virtual Loss" mechanisms, while stating global locks can reduce efficiency (Świechowski et al., 2023, p.48, p.49). In General Video Game AI, enhancements improved average win rates from 31.0% to 48.4% across 60 games (Świechowski et al., 2023, p.24).

Takeaway

This survey demonstrates MCTS has evolved into a flexible optimization framework, but achieving practical performance in complex domains beyond chess but requires domain-specific modifications, highlighting the balance between algorithmic generality and computational efficiency (Świechowski et al., 2023, p.45)

2.7 · Unveiling Concepts Learned by a World-Class Chess-Playing Agent (Pálsson and Björnsson, 2023)

Methodology

Pálsson and Björnsson used model probing techniques including linear surrogate models and Shapley value sampling to analyze Stockfish 14.1′s internal representations (Pálsson and Björnsson, 2023, p.1, p.4). This study compared NNUE's learned evaluation against the classical hand-crafted model by probing 100,000 positions from Leela Chess Zero's training dataset (Pálsson and Björnsson, 2023, p.4).

Findings

Their research finds that NNUE can statically detect complex threats including forks, mating attacks, and promotion potential without requiring search, which traditional engines based on alpha-beta with hand-crafted evaluation could only achieved through lookahead (Pálsson and Björnsson, 2023, p.7). They also found that linear combination of classical hand-crafted concepts explained less than 50% of NNUE's evaluation, which indicates that the network discovered fundamentally different position assessment logic (Pálsson and Björnsson, 2023, p.5). They also found that NNUE places significantly less weight on material than the classical model while emphasizing dynamic concepts like passed pawns (Pálsson and Björnsson, 2023, p.8).

The same study also found that probing accuracy for almost all concepts increased after the first linear layer, which proves that the network comes up with high-level ideas (Pálsson and Björnsson, 2023, p.6). King safety showed differences, with the classical formula having a Shapley value of 0.086 compared to NNUE's 0.019, suggesting NNUE discovered a more effective way to assess king safety (Pálsson and Björnsson, 2023, p.5, p.7).

Takeaway

This paper demonstrates that NNUE's domain-optimized efficiency enables it to learn "tactical intuition" statically without policy learning, but this performance gain comes at the cost of interpretability. The network's highly compressed, non-linear representations make it nearly impossible to fully understand its reasoning through classical analysis methods (Pálsson and Björnsson, 2023, p.2).

3 · Synthesis

The sources reveal a fundamental architectural separation in chess AI that captures other aspects beyond chess itself. The question of whether to embed domain knowledge for efficiency using NNUE (Klein, 2022, p.191; Maharaj, Polson and Turk, 2022, p.5; Pálsson and Björnsson, 2023, p.3) or chase generality at computational cost with MCTS (Klein, 2022, p.16; Maharaj, Polson and Turk, 2022, p.6; Krakovský and Liberda, 2025, p.1) is the big divide between the main architectural paradigms. This trade-off manifests across three dimensions that ossicilates favoring opposite architectures.

NNUE's domain optimization creates a "calculation engine" (Maharaj, Polson and Turk, 2022, p.10) that excels at tactical depth through exhaustive search, evidenced by Stockfish solving Plaskett's Puzzle at depth 40 (Maharaj, Polson and Turk, 2022, p.1) while processing 1,000 times more nodes per second than LCZero (Maharaj, Polson and Turk, 2022, p.9). However, this efficiency comes from chess-specific design choices, such as the HalfKP representation or different iterations on HalfKA in modern versions, and incremental update mechanisms (Klein, 2022, p.197) that sacrifice transferability to other domains. The architecture essentially learns to compress evaluation functions into neural weights, maintaining search speeds of 4.6 million nodes per second (Klein, 2022, p.212) through 8-bit integer mathematics and SIMD instructions (Klein, 2022, p.193). Despite this domain specificity, NNUE discovers fundamentally different position assessment logic (Pálsson and Björnsson, 2023, p.5), with linear combinations of classical concepts explaining less than 50% of its evaluation (Pálsson and Björnsson, 2023, p.5).

Alternatively, AlphaZero-style systems represent "intuition engines" (Maharaj, Polson and Turk, 2022, p.10) that discover positional understanding through pure self-play (Krakovský and Liberda, 2025, p.1), demonstrated by LCZero's superior performance in complex 4-piece endgames (Sadmine, Husna and Muller, 2023, p.9) and drawing positions (Sadmine, Husna and Muller, 2023, p.9). This domain-agnostic approach enables the same engine to master multiple games (Maharaj, Polson and Turk, 2022, p.10), with AlphaZero using 19-39 residual blocks to reach 5185 Elo across Chess, Shogi, and Go (Klein, 2022, p.162). However, this generality requires computational resources that remain inaccessible to individual developers (Krakovský and Liberda, 2025, p.10), with training estimates reaching 510 days (Krakovský and Liberda, 2025, p.9) and performance bottlenecks consuming 40-50% of runtime in MCTS calculations (Krakovský and Liberda, 2025, p.7). The paradigm's reliance on pattern matching also creates failures, as seen when LCZero wasted 92.4% of search nodes on inferior moves (Maharaj, Polson and Turk, 2022, p.8-9) because its policy network misjudged position complexity.

Furthermore, this computational divide extends to accessibility: NNUE implementations like Stockdory demonstrate functional performance without mentioning scalability barriers (Chitale et al., 2024, p.5), while AlphaZero replication remains practical only for organizations with massive resources (Krakovský and Liberda, 2025, p.10). This also bleeds into availability for potential end users, as consumer hardware simply cannot support AlphaZero-style engines that

require TPU arrays or high-end GPUs, whereas NNUE maintains CPU efficiency (Klein, 2022, p.193).

4 · Conclusion

These sources demonstrate that NNUE represents the superior architectural choice for practical chess engine development. While AlphaZero-style systems offer domain-agnostic learning, NNUE demonstrates superiority in both performance and accessibility. Stockfish's ability to solve Plaskett's Puzzle at depth 40 (Maharaj, Polson and Turk, 2022, p.1) while maintaining 1,000 times greater search speed (Maharaj, Polson and Turk, 2022, p.9) proves NNUE's tactical dominance. More critically, NNUE's CPU efficiency (Klein, 2022, p.193) makes it deployable on consumer hardware, which can't be said with AlphaZero's 48 TPU requirement (Klein, 2022, p.160) and 510-day training estimates for replication (Krakovský and Liberda, 2025, p.9).

Although NNUE sacrifices cross-domain generality through chess-specific optimizations, this trade-off proves worthwhile. The comparatively fewer computational resources required means that individual developers could train domain-specific NNUE architectures for problems beyond chess, as adapting the principles to Go or Shogi remains feasible without organizational scale infrastructure.

Bibliography

Chitale, A.M., Cherian, A.M., Singh, A. and P, P., 2024. Implementing the Chess Engine using NNUE with Nega-Max Algorithm. [online] Available at: <https://ieeexplore.ieee.org/abstract/document/10677087/authors>.

Klein, D., 2022. Neural Networks for Chess. arXiv preprint arXiv:2209.01506. [online] Available at: <https://arxiv.org/abs/2209.01506>.

Krakovský, J. and Liberda, D., 2025. Chess Engine Inspired by AlphaZero. arXiv preprint arXiv:2209.01506. [online] Available at: <https://arxiv.org/pdf/2209.01506>.

Maharaj, S., Polson, N. and Turk, A., 2022. Chess AI: Competing Paradigms for Machine Intelligence. [online] https://doi.org/10.3390/e24040550.

Pálsson, A. and Björnsson, Y., 2023. Unveiling Concepts Learned by a World-Class Chess-Playing Agent. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23). [online] pp.4857–4865. https://doi.org/10.24963/ijcai.2023/541.

Sadmine, Q.A., Husna, A. and Muller, M., 2023. Stockfish or Leela Chess Zero? A Comparison Against Endgame Tablebases. [online] Edmonton, Canada. Available at: <https://webdocs.cs.ualberta.ca/~mmueller/ps/2023/ACG_2023_Stockfish.pdf>.

Świechowski, M., Godlewski, K., Sawicki, B. and Mańdziuk, J., 2023. Monte Carlo Tree Search: A review of recent modifications and applications. Artificial Intelligence Review, [online] 56, pp.2497–2562. https://doi.org/10.1007/s10462-022-10228-y.