Synthesis Matrix: NNUE vs AlphaZero Trade-offs

Source	Performance Characteristics	Computational Efficiency	Accessibility & Practicality	Generality vs Domain-Specificity
Klein (2022) *Neural Networks for Chess*	• AlphaZero: 5185 Elo, 19-39 residual blocks • NNUE: +80 Elo gain in Stockfish 12 • Maia: ">50%" move-matching for human play vs Stockfish's 35%	• AlphaZero: 80,000 nps • NNUE/Stockfish: 4.6M nps (58x faster) • Stockfish 8: 7.5M nps • Uses 8-bit integer math, SIMD (`VPADDW`, `VPSUBW`)	• AlphaZero: 48 TPUs required • NNUE: CPU-efficient, consumer hardware compatible • HalfKP: 81,920-bit input for incremental updates	• AlphaZero: General (Chess, Shogi, Go) • NNUE: Chess-specific optimizations • NNs fail on out-of-domain data
Maharaj et al. (2022) *Competing Paradigms*	• Stockfish: Solved Plaskett's Puzzle at depth 40 (mate in 29) • LCZero: Failed after 60M nodes • 92.4% of LC0 nodes followed inferior move • LC0 efficient when intuition correct (5.5M vs 500M nodes)	• Stockfish: 1.5×10^8 nps • LCZero: 1.4×10^5 nps (1000x slower) • LC0 given 34x more compute than tournament standard	• Stockfish: "Calculation engine" • LCZero: "Intuition engine"	• Stockfish: Better tactical calculation, edge cases • LCZero: Pattern matching, generalizable but fails when policy misjudges complexity
Sadmine et al. (2023) *Endgame Tablebases*	• Stockfish: Superior in 3-piece endgames • LC0: Fewer mistakes in 4-piece endgames (1.32% vs 1.47% errors) • LC0: Better at predicting draws • LC0: More accurate evaluations even when making mistakes	• Tests used raw policy networks (no search) • Small search budget: 400 nodes • Both engines rated 2850 Elo	• Isolated learning ability by removing search • Tested against Syzygy tablebases (perfect play)	• Stockfish: Tactical calculation in simpler positions • LC0: Superior "positional feel" in complex endgames
Krakovský & Liberda (2025) *AlphaZero Implementation*	• Only 1,200 games over 1 GPU-hour • AlphaZero: 44M games over 41 TPU-years • Engine learned to draw, failed to learn winning • Preferred draws by repetition/50-move rule	• 40-50% runtime in MCTS calculations • GPU speedup only 2x due to lack of batching • Python object conversion bottlenecks • 510-day training estimate to reach AlphaZero scale	• Python implementation severely limited • C++ or Rust necessary for efficiency • Demonstrates extreme accessibility barrier • Only practical for organizations with massive resources	• Attempted to replicate AlphaZero methodology • Used 2 residual blocks vs AlphaZero's 19 • Computational gap makes generality impractical for individuals
Chitale et al. (2024) *Stockdory Implementation*	• Stockfish: 80.19% accuracy (1922 game) • Stockdory: 45.28% accuracy (1922 game) • Stockdory: 52% vs Stockfish's 48% (1889 psychological game)	• Stockfish: 6m 50s (1922 game), 7m 5s (1889 game) • Stockdory: 9m 3s (1922 game), 6m 28s (1889 game)	• No computational barriers mentioned • Demonstrates NNUE accessibility for individuals/small teams • Functional implementation possible without organizational resources	• NNUE with Nega-Max algorithm • Domain-optimized for chess • Relative ease of building functional NNUE engine
Świechowski et al. (2023) *MCTS Review*	• AlphaGo: "Second major breakthrough" • Vanilla MCTS: Fails in tactical traps • General Video Game AI: 31.0% → 48.4% win rate (60 games)	• 4 phases: Selection, Expansion, Simulation, Backpropagation • 3 parallelization strategies: Leaf, Root, Tree • Global locks reduce efficiency	• Aheuristic: requires only environment rules • No domain knowledge needed initially	• MCTS: Flexible optimization framework • Domain-agnostic in theory • Requires domain-specific modifications for complex domains

	• Fails with high branching (StarCraft: 10^{50} actions)	• Virtual Loss for shared structures	• Domain modifications needed for practical performance	• Balance between generality and efficiency
Pálsson & Björnsson (2023) *Unveiling NNUE Concepts*	• NNUE: Statically detects forks, mating attacks, promotion • Classical concepts explain <50% of NNUE evaluation • Less weight on material, more on dynamic concepts • King safety: Shapley 0.086 (classical) vs 0.019 (NNUE)	• Probed 100,000 positions from LC0 training data • Linear surrogate models + Shapley value sampling • Probing accuracy increases after first linear layer	• Analyzed Stockfish 14.1 internal representations • Model probing techniques accessible for research	• NNUE: Discovers fundamentally different position logic • "Tactical intuition" without policy learning • Domain-optimized but discovers novel concepts • Trade-off: Performance vs interpretability

Key Themes Across Sources

Theme	Evidence
Speed Disparity	• NNUE: 4.6M-150M nps (Klein, Maharaj) • AlphaZero/LC0: 80K-140K nps (Klein, Maharaj) • **1000x difference** in node evaluation speed
Computational Accessibility	• AlphaZero: 48 TPUs, 41 TPU-years (Klein, Krakovský) • NNUE: CPU-efficient, consumer hardware (Klein, Chitale) • Replication estimate: 510 days for AlphaZero (Krakovský) • No barriers mentioned for NNUE (Chitale)
Tactical vs Positional	• NNUE: "Calculation engine," tactical depth (Maharaj, Sadmine) • AlphaZero: "Intuition engine," positional feel (Maharaj, Sadmine) • NNUE: Better in 3-piece endgames (Sadmine) • LC0: Better in complex 4-piece endgames (Sadmine)
Learning Mechanisms	• NNUE: Domain-optimized, HalfKP features, incremental updates (Klein, Pálsson) • AlphaZero: Self-play RL, domain-agnostic (Klein, Krakovský, Świechowski) • NNUE: Discovers novel concepts statically (Pálsson) • AlphaZero: Requires policy + value networks + MCTS (Świechowski, Maharaj)
Failure Modes	• LC0: 92.4% nodes on wrong move when policy misjudges (Maharaj) • Vanilla MCTS: Tactical traps, high branching (Świechowski) • NNUE: Less generalizable across games (Klein) • AlphaZero: Fails to learn winning without sufficient compute (Krakovský)
Practical Implementation	• NNUE: Stockdory functional without major barriers (Chitale) • AlphaZero: Python bottlenecks 40-50% runtime (Krakovský) • NNUE: 8-bit integer math, SIMD instructions (Klein) • AlphaZero: Requires C++/Rust for efficiency (Krakovský)