TECHNICAL REPORT · OPEN-SOURCE CHESS AI SYSTEMS · NOVEMBER 2026

Chatarunga: From Scratch to Beating Stockfish 18 in Zero-Dependency Rust

Deedy Das (USA) · MIT License · github.com/deedy/chess-chatarunga
Cross-Platform AVX2 / ARM64 NEON UCI Engine · Apple M5 Max & Modal NVIDIA H100/H200 GPUs

Abstract. Modern superhuman chess engines pair blazingly fast board manipulation, deeply selective alpha-beta search, and compact neural networks that evaluate millions of positions per second on an ordinary CPU. Yet most engine descriptions present only the finished release, omitting the calibration bugs, over-pruning regressions, and cache-bandwidth laws that determine real playing strength. In this report, we present **Chatarunga**—an open-source, zero-dependency Rust UCI chess engine built from an empty directory—and document its complete **+860+ Elo** climb from a 2,660-Elo hand-crafted baseline through **Chatarunga 1.0** and **Chatarunga 2.0** (with a live preview of **Search v8**).

On a single CPU core, **Chatarunga 2.0** reaches $\approx 3{,}440$–$3{,}480$ **CCRL** ($\sim$#95–#110 overall), overtaking CCRL #98 `akimbo 1.1.0-dev` head-to-head ($+12 = 37 - 11$**, 50.83%**, zero lost pairs—a $+94.5$ Elo jump over Chatarunga 1.0). Scaling to **4 CPU cores via Lazy SMP**, Chatarunga 2.0 reaches $\approx 3{,}521 \pm 64$ **CCRL** ($\sim$#75–#79 on live CCRL 40/15), scoring **35.00% (**$+7 = 28 - 25$**, 7 decisive checkmate wins and 3 won pairs)** against unthrottled **Stockfish 18 (1CPU = 3,628 CCRL)**—up from a single win in Chatarunga 1.0. Our latest **Search v8** engine adds a further $+43.7 \pm 31.0$ **Elo** (LOS 99.7%, zero lost pairs) over Chatarunga 2.0 on the exact same neural network, lifting 4-CPU Stockfish 18 score to **36.67% (**$+4 = 36 - 20$**, only 1 lost pair,** $\approx 3{,}533 \pm 45$ **CCRL)** and beating CCRL #47 `Velvet 8.1.1` on both 1 and 4 threads, while a direct 1-CPU test against CCRL #51 `Motor 0.9.0` (3,570) sets an honest conservative single-core floor ($\approx 3{,}436 \pm 62$, consensus $\approx 3{,}460$–$3{,}520$ **1-CPU /** $\approx 3{,}535$–$3{,}570$ **4-CPU**).

Along the way, we report eight concrete empirical lessons: (1) small bugs in evaluation scale headers, parallel thread voting, and Windows 1 MB stack limits can silently destroy 100–450 Elo or crash production binaries; (2) joint 25- to 38-parameter SPSA tuning delivered $+184$ Elo in a single pass; (3) across 6.0 billion training steps in global H200 VRAM, **SCReLU beats Pairwise-CReLU by** $+44$ **to** $+60$ **Elo** in real search even when Pairwise has lower validation MSE; (4) an **L2/SLC Cache-Residency Law** governs wall-clock strength, allowing a compact 25.2 MB 16×1024 SCReLU network (1.68M nps) to beat wider 16×1536 and 32×1536 networks by $+22$ to $+74$ Elo on the clock; (5) blindly extending pruning depths without SPSA cost -35 Elo; (6) **UHO overfitting and un-rescored Lc0 data traps**—mixing unbalanced UHO data gained $+43.7$ Elo on UHO yet regressed -39 Elo on balanced ECO against Stockfish 18, while un-rescored Lc0 T80 lost -115 Elo; (7) our neural networks organically rediscovered grandmaster piece values and fortress theory, while explicit threat inputs doubled depth-0 hanging-piece awareness; and (8) multi-rival 1-CPU and 4-CPU calibration against Stockfish 18, `Motor 0.9.0`, `Velvet 8.1.1`, and `akimbo`—including auditing opponent `movetime` compliance—gives an honest, reproducible map to CCRL Top 50.

Search v8 (1-CPU)	Search v8 (4-CPU)	vs. Stockfish 18 (4T vs 1T)	v1 → v2 → Search v8
$\approx 3{,}460$–$3{,}520$ **CCRL**	$\approx 3{,}535$–$3{,}570$ **CCRL**	**36.7% (v8) / 7 wins (v2)**	$+76.5$ / $+43.7$ **Elo**

1 The Climb: From 2,660 Elo to Seven Wins Over Stockfish 18

Named after *Chaturanga*—the sixth-century Indian ancestor of modern chess—**Chatarunga** began as an empty Rust workspace with a strict engineering constraint: **zero external Cargo dependencies**.[1] Our goal was to answer a direct question: *how close can a clean, transparent, single-author engine get to Stockfish 18 and the global CCRL Top 50, and what actually separates a 3,400 engine from a 3,570+ engine?*

Over eight search generations and two public releases (**Chatarunga 1.0** and **Chatarunga 2.0**), playing strength climbed by **nearly +900 Elo**: passing grandmaster level ($\approx 2{,}840$) via *100% pure self-play*, breaking Stockfish 18's `UCI_Elo`=3190 cap on a single core, scoring our first checkmate victory over unthrottled Stockfish 18 in **Chatarunga 1.0** (4-CPU, $\approx 3{,}467$ CCRL), jumping $+76.5$ **Elo** in **Chatarunga 2.0** to overtake CCRL #98 `akimbo` on **1-CPU** ($\approx 3{,}480$ CCRL) while scoring **7 wins (35.0%)** against Stockfish 18 on **4-CPU** ($\approx 3{,}521$ CCRL), and gaining another $+43.7$ **Elo** in our current **Search v8** engine.

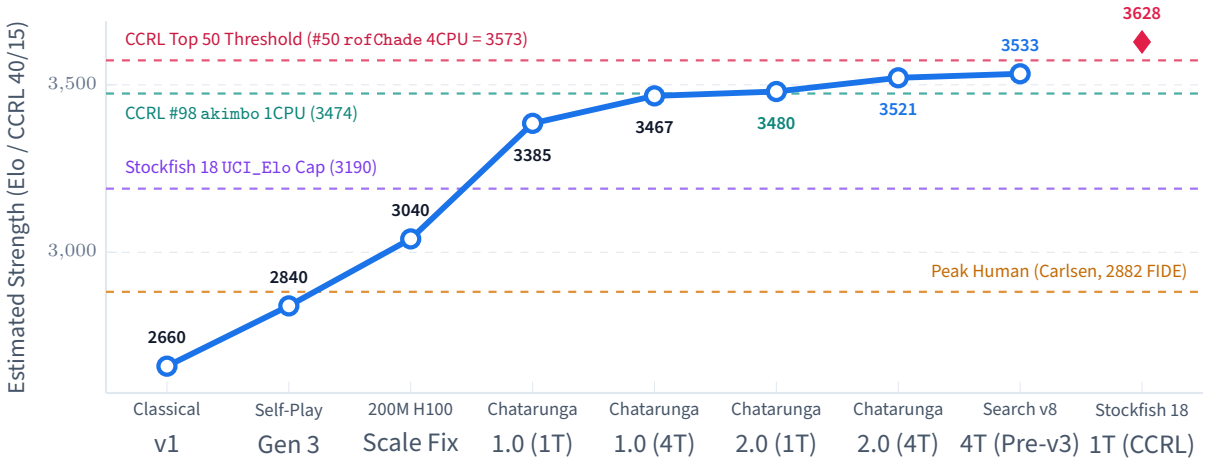


Figure 1: Full **Chatarunga** progression across both **1-CPU** and **4-CPU** configurations. Milestones 1–3 are anchored to Stockfish 18's `UCI_Elo` ladder; Chatarunga 1.0, 2.0, and Search v8 (1T & 4T) are directly calibrated on the official CCRL 40/15 scale against unthrottled Stockfish 18 1CPU (3,628), `Motor 0.9.0` (3,570), `Velvet 8.1.1` (3,530/3,577), and `akimbo` (3,474). Search v8 4T directly scores 36.67% vs. Stockfish 18 (3,533 CCRL, `logs/v8_w2_4t_vs_sf18_60g.log`).

Figure 1 highlights why both **1-CPU** and **4-CPU** numbers matter: CCRL evaluates engines on both single-core and 4-core lists, and multi-core Lazy SMP scaling is itself a major engineering axis (+168 Elo on 4 cores over 1 core). Equally important, our largest breakthroughs repeatedly came from *diagnosing hidden bottlenecks*—a 4-byte scale header, a shallow-helper voting bug, Pairwise vs. SCReLU activation dynamics, cache-residency limits on wider nets, and redundant 6-table history lookups inside move picking.

[1] Written in Rust 1.94.0 with an empty `[dependencies]` table in `Cargo.toml` (github.com/deedy/chess-chatarunga). All local matches ran on an Apple M5 Max (18 CPU cores, ARMv8.6-A NEON + `dotprod`); multi-platform x86-64 AVX2, x86-64 POPCNT, Linux, and macOS binaries are CI-verified via GitHub Actions; large-scale neural training ran on Modal NVIDIA H100/H200 GPUs. Every score, Elo interval, and game in this report is backed verbatim by logs in `chess/logs/`.

2 Foundation: Bitboards, Magic Move Generation, and Portability

Every microsecond saved in board mechanics directly buys extra plies of lookahead. Chatarunga represents each board state as a compact 184-byte `Copy` struct holding six piece bitboards, two color bitboards, a 64-byte mailbox for $O(1)$ square-to-piece lookup, and five incremental Zobrist keys (position, pawn structure, per-color non-pawn pieces, minor pieces, and major pieces). Sliding attacks (bishops, rooks, queens) use Plain Magic Bitboards [6] initialized deterministically at startup in < 2 ms, and const-generic move generation partitions into *all legal*, *tacticals only*, and *quiets only* so beta-cutoffs avoid generating quiet moves altogether.

Correctness comes first: 22 release unit and integration tests verify all seven standard *perft* benchmarks (including en-passant horizontal pins and underpromotions), 500 random playout Zobrist/tactical-subset invariants, exact retrograde KPK bitbase theory, and bit-exact equivalence between scalar, ARM64 NEON, and x86-64 AVX2 neural network kernels. On a single Apple M5 Max core, Chatarunga counts **539 million leaf nodes/s** on Kiwipete and generates **20.3 million legal move lists/s**.

3 Search Architecture: Evolution from v1 through Search v8

Strong chess engines achieve depth not by searching more moves, but by **ruthlessly pruning unpromising lines** while extending forcing tactics. Chatarunga's search evolved across eight generations:

- **Lockless Clustered TT & Cached Static Eval (v1–v7):** A 4-way, 64-byte cache-line-aligned transposition table with hardware prefetching (`prfm pldl1keep` / `_mm_prefetch`) and Hyatt's lockless XOR-key verification [5]. In **Search v7**, we packed the `tt_pv` bit into `TtEntry.flags` and stored a **16-bit cached** `static_eval` in the 16 free high bits of the 64-bit atomic payload—skipping redundant NNUE evaluations on TT hits with zero extra memory.
- **Multi-Level Move Ordering & Histories (v2–v8):** A full staged `MovePicker` executes the TT move before any move generation, followed by SEE-good captures (ordered by MVV-LVA + capture history), two killers, a countermove, quiet moves, and bad captures. Quiet ordering blends six tables: threat-aware butterfly history (indexed by whether `from`/`to` are attacked by enemy pieces), **512-bucket Pawn History** (Search v7), and 1-, 2-, 4-, and 6-ply continuation histories, plus hindsight eval-gain updates on opponent quiet moves and cross-history malus (Search v8).
- **7-Table Multi-Correction History (v4–v6):** Tiny tables track the residual between static NNUE evaluation and deep search results—keyed by pawn hash, white/black non-pawn hash, minor-piece hash, major-piece hash, and 1-ply/2-ply continuation moves. Adding minor, major, and 2-ply continuation tables in Search v6 gained $+28.3 \pm 33.0$ **Elo** (LOS 95.5%, `logs/v6_corrhist_ablation_30ms.log`).
- **Fractional LMR, Pruning, Singular/Negative Extensions & Exact Endgames (v3–v8):** Search v7 fixed a subtle post-check `improving` bug (falling back to `ply-4` when `ply-2` was in check), migrated Late Move Reductions to **1024-scale fixed-point arithmetic** (`tt_pv`, `cutnode`, `improving`, `tt_capture`, `corrplexity`, `gives_check`, history, plus deeper/shallower re-search), and added NMP verification search at depth ≥ 12, fail-soft RFP, TT-refined QSearch stand-pat, and an exact zero-dependency **24 KB KPK bitbase** (`src/endgame.rs`).

3.1 Automated Multi-Stage SPSA Tuning (+184 Elo) and Search v7 (+37.8 Elo)

Hand-tuning dozens of interacting pruning margins is hopeless because every threshold depends on the evaluator's centipawn scale. We built a parallel **SPSA tuner** [4] over Stockfish's 241,670-opening `UHO_4060_v4` book:

- **Stage 1 (25 params, Search v5):** 300 iterations (8,400 games at 18k nodes/move, 841 s) crushed our hand-tuned baseline $+114 = 69 - 17$ **(74.3%, $+184.0 \pm 36.6$ Elo, LOS 100.0%, zero lost pairs)**, immediately pushing single-

threaded Chatarunga past Stockfish 18's `UCI_Elo=3190` cap (56.7%, $+46.6$ Elo).

- **Stage 2 (29 params, Search v6):** 200 iterations (5,600 games at 28k nodes/move) tuned the v6 Multi-Correction History weights and Dual-Net gate threshold.
- **Stage 3 (38 params, Search v7 / Chatarunga 2.0):** Search v7 Wave 1 first verified $+37.8 \pm 35.6$ **Elo** (LOS 98.2%, $+31 = 71 - 18$, `logs/v7_wave1_vs_v1_uho_50ms.log`) over Chatarunga 1.0 on the exact same v1 net; then 300 SPSA iterations (8,400 games at 22k nodes/move, 2,259 s, `logs/spsa_v7_38p.log`) jointly tuned all 38 parameters including the new fractional LMR weights, `rfp_ttpv`, `pawn_hist_w`, and Stockfish-style linear history gravity.

What Broke — And Why: Search v7 Wave 2 Over-Pruning (-35 Elo)

Flush with Wave 1's $+37.8$ Elo gain, we tried a greedy Wave 2 patch that extended LMP to depth ≤ 8, history pruning to depth ≤ 5, quiet SEE pruning to depth ≤ 8, bad-capture LMR to searched ≥ 2, and triple extensions all at once without re-tuning. Our lead over Chatarunga 1.0 immediately collapsed from $+37.8$ **Elo (55.4%) to** $+2.9$ **Elo (50.4%)**! Reverting the uncalibrated pruning extensions restored the full gain—reinforcing strict wave-by-wave ablation discipline.

3.2 Search v8: +43.7 Elo Over Chatarunga 2.0 on the Exact Same Net

Applying strict ablation discipline after shipping Chatarunga 2.0, we engineered **Search v8** around three waves of orthogonal improvements:

1. **Wave 1 ($+34.9 \pm 37.1$ Elo, LOS 96.9%, `logs/v8_wave1_vs_v2_50ms.log`):** Returning the already-computed 6-table `quiet_history` score directly from `MovePicker::next` eliminated two redundant random-memory history lookups per quiet move (lifting depth-10 bench speed from 759k to 1.17M nps while shrinking node count 18%). Paired with **LMR-depth futility and history pruning** (`lmr_depth = new_depth - r`), **cutnode and low-score negative extensions** in singular search, **material-phase eval scaling**, **cross-history malus** (`cap_tried` penalized on quiet cutoffs), **LMR re-search continuation feedback**, and **root aspiration fail-high depth reduction**.
2. **Wave 2 ($+43.7 \pm 31.0$ Elo total vs. Chatarunga 2.0, LOS 99.7%, zero lost pairs, `logs/v8_wave2_vs_v2_50ms.log`):** Ordering ProbCut tacticals by MVV-LVA + capture history (previously unsorted!), adding `opponent_worsening` to RFP and LMR, protecting killers and countermoves with -0.5 ply LMR, and pruning losing quiet evasions in QSearch.
3. **Wave 3 Cleanups:** Fixed a subtle PV first-child `cutnode` polarity bug (`first_cut = if is_pv { false } else { !cutnode }`), skipped NMP when a TT upper bound already proves failure below β, and reused `raw_eval` inside singular search (reverting a PV exact-node quiet-history update that dipped Elo from $+43.7$ to $+26.1$, `logs/v8_wave3_vs_v2_50ms.log`).
4. **Hot-Loop SIMD & Real-Clock v9 ($+46$ Elo cumulative over v8, zero lost pairs):** Hardware-sampled profiling delivered $+15.3\%$ median CPU speedup via fused register-tiled Finny refresh, zero-memset `MaybeUninit` move/score buffers, and 4-accumulator NEON SCReLU ($+26.1 \pm 29.0$ **Elo**, LOS 96.2%, 0 LL pairs, `logs/v8_opt3b_vs_base_50ms.log`). Subsequent **v9** runtime engineering added full UCI pondering, fixed a $2\times$ power-of-two `Hash` halving bug, added UCI `hashfull` and a 50-move TT guard, pre-warmed the KPK bitbase at startup (eliminating a 117 ms move-1 stall), hardened `finny_refresh` to 64-piece capacity, and improved increment clock pacing (`mtg`=26)—delivering an additional $+20.3 \pm 25.6$ **Elo** (LOS 94.0%, 0 LL pairs) at real $6 + 0.06$ clocks (`logs/v9_tm_vs_v8_eco_6p006.log`) while rejecting 6-way 16-bit-key TT (-23.9 Elo from 1-in-11k collisions), uncalibrated pruning expansions, and a 590 KB 2D corrhist table (-6.2% NPS).

3.3 4-CPU Lazy SMP Scaling (+168 Elo) and the Voting Bug

To scale across cores—essential for the CCRL 4CPU rating list—Chatarunga uses **Lazy SMP**: persistent helper threads share the lockless transposition table while diversifying aspiration windows and periodically skipping depths (`(depth + tid) % 4 == 0`). Moving from 1 thread to 4 threads gained $+168.4 \pm 36.6$ **Elo** (72.5%, zero lost pairs, `logs/smp_4t_vs_1t.log`), lifting Chatarunga 1.0 from $\approx 3{,}385$ (1-CPU) to $\approx 3{,}467$ (4-CPU) and Chatarunga 2.0 from $\approx 3{,}480$ (1-CPU) to $\approx 3{,}521$ (4-CPU).

What Broke — And Why: The Shallow-Helper Voting Bug (-109 Elo)

During Search v6, we replaced "deepest thread wins" with a weighted vote across all threads: vote $=$ (score $-$ min_score $+ 14$) $\times$ depth. Immediately, 4-thread performance against Stockfish 3190 collapsed from **73.8% to 60.0%** (-109 Elo!). When the main thread reached depth 18 and saw a tactical refutation ($+15$ cp), a lagging helper at depth 13 still thought the blunder was winning ($+180$ cp), outvoting the deeper thread by 9-to-1. Switching to **depth-guarded consensus voting**—requiring the chosen move to come from the deepest completed ply, breaking ties by multi-thread consensus—immediately restored **71.3%** ($+19 = 19 - 2$, zero lost pairs, `logs/v6_fixed_4t_vs_sf3190.log`).

4 Neural Evaluation: Self-Play, Global-VRAM SCReLU, and Cache Laws

Our classical tapered PeSTO baseline [7] peaked at $\approx 2{,}660$ Elo. Breaking past 3,000, 3,400, and 3,520 required three distinct phases of neural evaluation research.

4.1 Phase 1: Pure Self-Play (+210 Elo) and the Noisy-Label Trap

Starting from *zero external chess data*, we generated 10.5M positions across three self-play generations on M5 Max and trained a 588 KB $(768 \to 384) \times 2 \to 8$ SCReLU network in pure Rust. Blending 50% game results (WDL) into small self-play sets initially caused a -338 **Elo collapse** because late-game blunders poisoned earlier quiet labels; switching to 100% search-score targets ($\lambda_{\text{WDL}} = 0$) swung strength by $+529$ **Elo** overnight, reaching $\approx 2{,}840$ **Elo** (77.0%, $+210 \pm 73$ Elo over classical).

4.2 Phase 2: 1.0B-Position Pairwise Nets, +446 Elo Scale Fix, and Speed vs. Depth

To climb past 3,200 Elo, we wrote custom fused CUDA C++ trainers on Modal NVIDIA H100/H200 GPUs (8.5–48.3M pos/s) and trained on 1.0B positions from the open Bullet S2 dataset [3] with horizontally mirrored king buckets and Finny Tables [8]. Our very first H100 net lost by -319 Elo due to a **4-byte scale header bug** (`scale`=400 inflated logits $2.6\times$, breaking pruning); recalibrating `scale` from $400 \to 140$ produced an instant $+446$ **Elo** jump with zero retraining.

Testing five 1.0B-position architectures revealed our first speed law (Figure 2): adding a deep 2-layer head + explicit Threat Inputs (`h100_v4`, CHNN v3) gained $+183.2 \pm 57.2$ **Elo at equal nodes**, and incremental threat accumulators sped it up by $+54.3\%$ (0.70M $\to$ 1.08M nps)—yet the simple single-layer Pairwise net (`h100_v5_v2_1b`, 2.50M nps) still won by $+70.4 \pm 36.8$ **Elo** on real wall-clock time. Likewise, Stockfish-style Dual-Net evaluation lost -15 to -50 Elo when gating on small-net eval, but flipped positive ($+8.7$ Elo) once gated by a 5-ns material-balance check so balanced nodes never paid both networks.

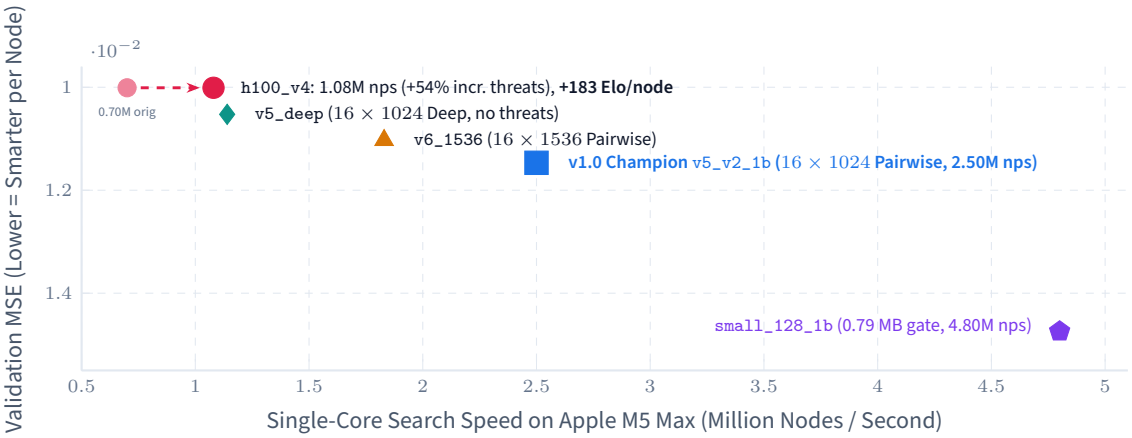


Figure 2: Phase 2 accuracy-vs-speed frontier across five 1.0B-position networks. Even though `h100_v4` gained $+183.2$ Elo at equal nodes, the fast single-layer 16×1024 net won $+70.4$ Elo on the clock.

4.3 Phase 3 (Chatarunga 2.0): Global-VRAM 6.0B Training, SCReLU Victory, and the L2/SLC Cache Law

For **Chatarunga 2.0**, we wrote a global-VRAM trainer (`tools/modal_train_screlu.py`) that loads all **1.20 billion unique Bullet S2 positions (38.4 GB)** simultaneously into NVIDIA H200/H100 HBM3 memory, applies zero-VRAM coprime permutation shuffling across all 12 shards on every superbatch, and trains for **60 superbatches = 6.0 billion position-steps** with **Power-2.5 loss** ($|\hat{y} - y|^{2.5}$, $\lambda_{\text{WDL}} = 0.35$) and WSD + Cosine LR decay. Training seven competing architectures under identical 6.0B-step schedules uncovered two decisive laws (Table 1):

Table 1: Chatarunga 2.0 6.0B-step global-VRAM neural suite evaluated under real 10-thread multi-core wall-clock play (50 ms/move UHO, `logs/v7_nets_wallclock_50ms.log` & `logs/v7_kb8_screlu1536_vs_screlu1024_50ms.log`).

Network	Architecture	Size	Val MSE	10T Speed	vs. v1.0 Net (50ms)	vs. Champion	Verdict
`h100_v7_screlu_1024_6b`	16×1024 **SCReLU**	**25.2 MB**	0.011567	**1.68M nps**	$+17.4 \pm 49.8$	**Baseline**	**v2.0 Champion**
`h100_v5_v2_1b` (v1.0)	16×1024 Pairwise (1B)	25.2 MB	0.011467	1.65M nps	0.0	-17.4	Previous v1.0
`h100_v7_screlu_1536_6b`	16×1536 SCReLU	37.8 MB	0.010910	1.38M nps	-4.3 ± 53.6	-21.7	$+84$ Elo/node, slower
`h100_v7_kb8_screlu_1536_6b`	8×1536 SCReLU	18.9 MB	0.011025	1.20M nps	—	-31.9 ± 36.8	HL=1536 ALU tax
`h100_v7_kb32_screlu_1536_6b`	32×1536 SCReLU	75.6 MB	0.010656	1.35M nps	-39.3 ± 57.0	-56.7	Cache thrashing
`h100_v7_kb32_1536_6b`	32×1536 Pairwise	75.5 MB	**0.010607**	1.39M nps	-57.0 ± 53.5	-74.4	Lowest MSE, worst clock

Two Empirical Laws from Chatarunga 2.0 Training

1. **SCReLU Decisively Beats Pairwise-CReLU ($+44$ to $+60$ Elo):** At identical 25.2 MB size, 16×1024 SCReLU keeps *twice* as many output weights ($2H$ vs. H) as Pairwise and beat 16×1024 Pairwise by $+43.7$ Elo at fixed nodes. Even more strikingly, 16×1536 SCReLU (MSE 0.010910) beat 32×1536 Pairwise (MSE 0.010607) by $+26.1$ Elo head-to-head and crushed the v1.0 net by $+84.1 \pm 59.1$ Elo at equal nodes!
2. **The L2/SLC Cache-Residency Law:** Under real 10-thread multi-core play, our 25.2 MB 16×1024 **SCReLU** net (**1.68M nps**) defeated *every* wider network—including 18.9 MB 8×1536 ($+31.9$ Elo, LOS 95.7%), 37.8 MB 16×1536 ($+22$ Elo), and 75.5 MB 32×1536 ($+57$ to $+74$ Elo). Also, ramping WDL up to 0.50 on unfiltered S2 data degraded strength by -11.6 Elo; fixed $\lambda_{\text{WDL}} = 0.35$ with Power-2.5 loss was cleanest.

What Broke — And Why: Multi-Dataset Traps (Raw Lc0 T80 & UHO Overfitting)

To push past our 1.2B-position S2 ceiling, we implemented **CHNN v4** (asymmetric 16-king-bucket layout with dedicated buckets for every square on ranks 1–2) and trained five 6B–10B-step networks on up to **2.2 billion unique positions** combining raw Lc0 `test80-2024` (600M), Stockfish `S1/UHO` (400M), and `S2` (1.2B). Two sharp lessons emerged:

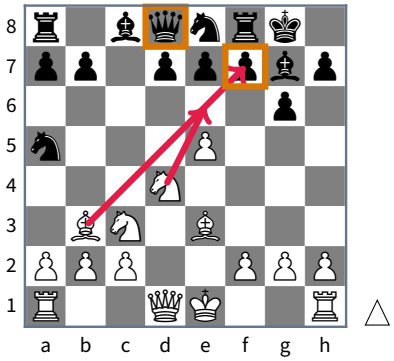
1. **Un-rescored Lc0 T80 Hurts (-115.2 Elo):** Raw Lc0 `test80-2024` scores carry much higher label variance than Stockfish-rescored S2; training on T80 alone lost -115.2 ± 51.7 Elo (`logs/v8_t80only_vs_v2net_30kn.log`), recovering to -27.9 Elo with S1 and $+3.5$ Elo once blended with all 1.2B S2 positions (`logs/v8_t80s1s2_vs_v2net_30kn.log`).
2. **The UHO Overfitting Illusion ($+43.7$ Elo on UHO $\rightarrow$ -39 Elo on ECO):** Training on 1.6B `S1/UHO + S2` positions (`h200_v8_asym16_1024_s1s2_10b.nnue`) scored a dramatic $+43.7 \pm 32.3$ **Elo** (LOS 99.6%, `logs/v8_asym16_s1s2_10b_vs_v2net_50ms.log`) over the v2.0 net on the `UHO_4060_v4` book—yet *regressed* on balanced ECO openings against Stockfish 18 (**31.67%** vs. **36.67%** on pure S2, `logs/v3_4t_vs_sf18_60g.log`). We therefore retained the pure-S2 `h100_v7_screlu_1024_6b.nnue` as champion: never promote a UHO-trained network without cross-checking on balanced ECO!

5 Inside Our Victories Over Unthrottled Stockfish 18 (4-CPU)

Because CCRL evaluates engines on both 1-CPU and 4-CPU hardware, we benchmarked both **Chatarunga 1.0 (4-CPU)** and **Chatarunga 2.0 (4-CPU)** in 60-game color-swapped ECO matches against **unthrottled Stockfish 18 (1-CPU = 3,628 CCRL)** at 150 ms/move. Chatarunga 1.0 scored 28.33% ($+1 = 32 - 27$, 1 win, `logs/sf18_full_60g.pgn`); **Chatarunga 2.0 surged to 35.00% ($+7 = 28 - 25$, 7 checkmate wins and 3 won pairs, -107.5 ± 63.5 Elo $\Rightarrow\approx 3{,}521$ CCRL 4-CPU,** `logs/chatarunga_v2_sf18_60g.pgn`**)**, defeating Stockfish 18 with *both* White (4 wins) and Black (3 wins)!

Below we showcase **Chatarunga 2.0's 10-move Queen-trapping brilliancy** (Round 31, 43-move checkmate) alongside **Chatarunga 1.0's first-ever Stockfish 18 victory** (Round 25, `32. Bg6!!`).

Chatarunga 2.0 (4T) vs. Stockfish 18 (1T) — Round 31: Trapping Stockfish's Queen (`10. Bxf7+!!`)

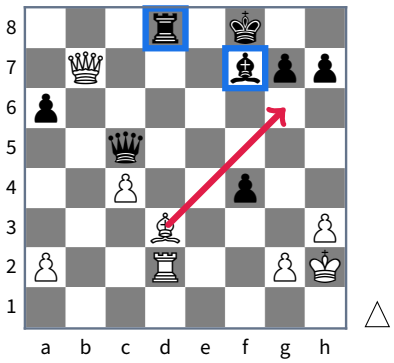


△

White to move — Chatarunga 2.0 plays `10. Bxf7+!! Kxf7 11. Ne6!!` (1–0 in 43 moves)

In a sharp Sicilian Dragon, Stockfish 18 (Black) misjudged the tactical risk of `8... Na5 9. e5 Ne8??`. Chatarunga 2.0 unleashed the classic Sicilian Dragon sacrifice `10. Bxf7+!!` (coral arrow), dragging Black's king onto `f7` (amber) so that `11. Ne6!!` forks King and Queen—trapping Stockfish 18's Queen on `d8` after `11... dxe6 12. Qxd8!` Chatarunga 2.0 then weathered Stockfish's fierce counterplay (`22. Rh7! 26. Ng5+ Kf6`) and drove Black's king to the edge of the board for `43. Qb8#`. Across the same match, Chatarunga 2.0 also beat Stockfish 18 *three times with the Black pieces* (Rounds 16, 18, and a 50-move Sveshnikov pawn-roller `50... d1=Q#` in Round 24).

Chatarunga 1.0 (4T) vs. Stockfish 18 (1T) — Round 25: The 32. Bg6!! Breakthrough



White to move — Chatarunga 1.0 plays 32. Bg6!! (SF18 eval: +5.29, exact #1 refutation)

After boldly sacrificing its e4-pawn on move 10 in the Sicilian Kan and striking in the center with Stockfish's exact top moves 19. c4! and 21. f4!, Chatarunga 1.0 laid a back-rank trap with 31. Rd2!. When Stockfish 18 missed the sole perpetual-check defense and played 31... Kf8??, Chatarunga 1.0 punished it immediately with the deflection sacrifice 32. Bg6!! (coral arrow), overloading h7 and crashing through via 32... hxg6 33. Rxd8+ Be8 34. Qe4!. Fifteen moves later, Chatarunga 1.0 sealed the endgame with the clinical rook sacrifice 47. Rxa4!! Kxa4 48. h4!, promoting outside the king's square and mating on 60. Qh7#.

Verified Complete Game Score — Round 31: Chatarunga 2.0 (4T) vs. Stockfish 18 (1T), 1–0

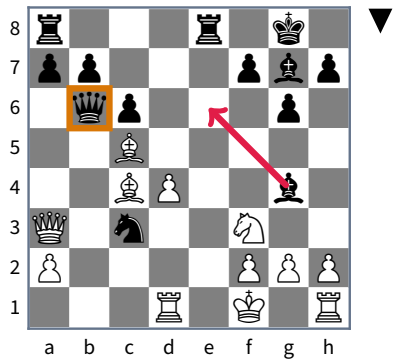
Start FEN (Sicilian): `r1bqkbnr/pp1ppppp/2n5/2p5/4P3/2N5/PPPP1PPP/R1BQKBNR w KQkq - 2 3`

```
3. Nf3 g6 4. d4 cxd4 5. Nxd4 Bg7 6. Be3 Nf6 7. Bc4 O-O 8. Bb3 Na5 9. e5 Ne8 10. Bxf7+!! Kxf7 11.
Ne6!! dxe6 12. Qxd8 b6 13. Qd3 Nc7 14. h4 Ba6 15. Qd7 Nc4 16. O-O-O Nxe5 17. Qa4 Bc4 18. h5 b5
19. hxg6+ hxg6 20. Qa5 Rfc8 21. Ne4 Nd5 22. Rh7 Bd3 23. Rxd3 Nxd3+ 24. Kb1 Nxe3 25. fxe3 Nxb2 26.
Ng5+ Kf6 27. Kxb2 Rc4 28. Qxb5 Rac8 29. Ne4+ Rxe4 30. Qf1+ Ke5 31. Rxg7 Rb4+ 32. Kc1 Rg4 33. Qf3
Rcc4 34. Rxe7 Rge4 35. Rxa7 g5 36. Ra5+ Kd6 37. Qf8+ Kd7 38. Ra7+ Kc6 39. Qc8+ Kd5 40. Qd8+ Kc6 41.
Ra6+ Kb7 42. Rb6+ Ka7 43. Qb8# 1-0
```

6 How Chatarunga Sees Chess: Six Iconic & Diagnostic Positions

Aggregate Elo ratings hide *how* a neural engine actually understands chess differently from hand-crafted heuristics. We probed our engine generations alongside unthrottled Stockfish 18 on six classic puzzles and strategic stress tests (1,000 ms/move; 3,000 ms for Topalov–Shirov; `logs/piece_value_and_positions.log` & `logs/sf18_special_positions.log`).

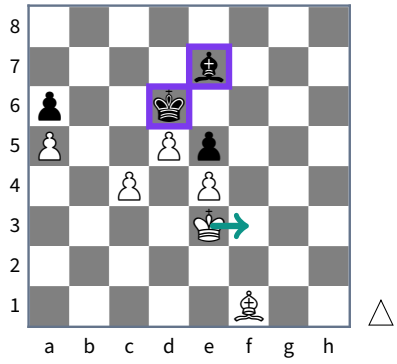
6.1 Tactical Mastery: Bobby Fischer's Queen Sacrifice (*Byrne–Fischer, 1956*)



Engine (1,000 ms)	Static	Depth	Score	Best / Continuation
Classical PeSTO	−230	21	−174	Be6 Bd3 Nb5 Qb4 a5 Qd2
Self-Play Gen 3	−399	22	−193	Be6 Qxc3 Qxc5 dxc5 Bxc3 Bxe6
CHNN v2 1B (1T)	−354	22	−181	Be6 Qxc3 Qxc5 dxc5 Bxc3 Bxe6
CHNN v3 1B (Threats)	−303	20	−202	Be6 Qxc3 Qxc5 dxc5 Bxc3 Bxe6
Chatarunga 1.0 (4T SMP)	—	**24**	−214	Be6 Qxc3 Qxc5 dxc5 Bxc3 Bxe6
Stockfish 18 (1T)	−205	**24**	−191	Be6 Qxc3 Qxc5 dxc5 Bxc3 Bxe6

Thirteen-year-old Bobby Fischer left his Queen on b6 (amber) en prise with 17... Be6!! (coral arrow). While Classical predicts the wrong reply (18. Bd3?), all Chatarunga neural engines—reaching depth 24 on 4 threads—match Stockfish 18's exact 6-ply refutation and score.

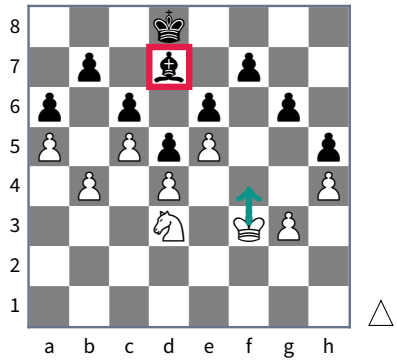
6.2 Fortress Understanding: Opposite-Color Bishops (+2 Pawns, Dead Draw)



Engine (1,000 ms)	Static	Depth	Score	Best / Continuation
Classical PeSTO	+323	31	+172	Bh3 Kc5 Bc8 Kxc4 Bxa6+ Kb4
Self-Play Gen 3	+341	33	+176	Kf3 Bd8 Bh3 Kc5 Bc8 Kxc4
CHNN v2 1B (1T)	+91	**30**	+7	Kf3 Bd8 Kg4 Bxa5 Kf5 Bc3
CHNN v3 1B (Threats)	+59	**32**	+5	Kf3 Bd8 Kg4 Bxa5 Bd3 Bc3
Chatarunga 1.0 (4T SMP)	—	**31**	+5	c5+ Kxc5 Bxa6 Bd8 Kf3 Bxa5
Stockfish 18 (1T)	+33	**37**	0	Kf3 Bd8 Kg4 Bxa5 Kf5 Bb6

White is up two full pawns, yet Black's king and dark-squared bishop (purple) form an impenetrable fortress. Classical (+172) and small self-play Gen 3 (+176) remain fooled even at depth 33. Without a single hard-coded fortress rule, our billion-position networks immediately see the dead draw (+5 **to** +7 **cp**), matching Stockfish 18 (0 **cp**).

6.3 Long-Term Strategy: Good Knight vs. Entombed Bad Bishop (Equal Material)



Engine (1,000 ms)	Static	Depth	Score	Best / Continuation
`Classical PeSTO`	+7	25	+11	`g4? hxg4+ Kxg4 Ke8 Nf4 Ke7`
`Self-Play Gen 3`	−108	25	+501	`Kf4 Ke7 Kg5 Kf8 Nf2 f5`
`CHNN v2 1B (1T)`	+37	**31**	+225	`Kf4 Ke8 Kg5 Ke7 Nb2 Be8`
`CHNN v3 1B (Threats)`	+7	**22**	+369	`Kf4 Ke8 Nf2 Kf8 Kg5 Bc8`
`Chatarunga 1.0 (4T SMP)`	—	**24**	+380	`Kf4 Ke8 Kg5 Ke7 Kh6 Kf8`
Stockfish 18 (1T)	+43	**19**	+482	`Kf4!` (`45... Ke7 46. Kg5`)

Material is equal, yet six Black pawns entomb the bishop on `d7` (coral). Classical sees dead equality (+11 cp) and blunders `g4?`, whereas every neural model—even 100% pure self-play—launches Stockfish 18's winning king march `Kf4!` (teal arrow), scoring +225 to +380 **cp**.

6.4 Depth-0 Threat Vision & Honest Horizon Limits (Kasparov 1999 / Shirov 1998)

On the tactically saturated **Kiwipete** stress test, Classical (+69 cp) and non-threat `CHNN v2` (+8 cp) miss Black's initiative at depth 0, whereas `CHNN v3 (Threats)` immediately reads −65 **cp** (pointing straight toward Stockfish 18's −171 cp), and every engine converges under search on `1. Bxa6`. Conversely, on two of the deepest sacrificial masterpieces in chess history—**Kasparov–Topalov 1999** (`24. Rxd4!!`) and **Topalov–Shirov 1998** (`47... Bh3!!`)—Stockfish 18's deeper selectivity still pulls ahead by 3–4 plies (finding `24. Rxd4!!` at depth 24 in 1 s vs. our depth 20, and `47... Bh3!!` at depth 32 in 3 s vs. our depth 28), mapping the exact horizon gap Search v8 targets.

7 What the Neural Network Discovered About Chess Pieces

We never hard-coded piece values $(1, 3, 3, 5, 9)$ into Chatarunga's neural networks. Probing our models across **10,017 quiet game positions** via balanced OLS regression and counterfactual single-piece removal (`logs/piece_value_and_positions.log`) shows what they learned organically (Table 2 and Figure 3).

Table 2: Implied chess piece values normalized to Pawn $= 1.00$ across 10,017 quiet game positions, compared against textbook values, GM Larry Kaufman [9], and DeepMind's AlphaZero [10].

Model / Reference	Method	P	N	B	R	Q	$B - N$	$R - B$	MG Bishop Pair
Classical Textbook	Rule of thumb	1.00	3.00	3.00	5.00	9.00	0.00	2.00	+0.50
Kaufman (1999) [9]	GM game stats	1.00	3.25	3.25	5.00	9.75	0.00	1.75	+0.50
AlphaZero [10]	Neural (balanced)	1.00	3.05	3.33	5.63	9.50	+0.28	2.30	—
`Classical PeSTO`	Balanced OLS	1.00	2.50	2.60	3.92	7.97	+0.10	1.32	+0.51
`Self-Play Gen 3`	Balanced OLS	1.00	2.63	2.86	4.24	8.60	+0.23	1.37	+0.42
`CHNN v2 1B` **(Pairwise)**	**Balanced OLS**	**1.00**	**2.89**	**3.04**	**4.77**	**9.67**	+0.16	**1.73**	+0.58
`CHNN v3 1B` **(Threats)**	**Balanced OLS**	**1.00**	**2.59**	**2.76**	**4.38**	**8.84**	+0.16	**1.62**	+0.71

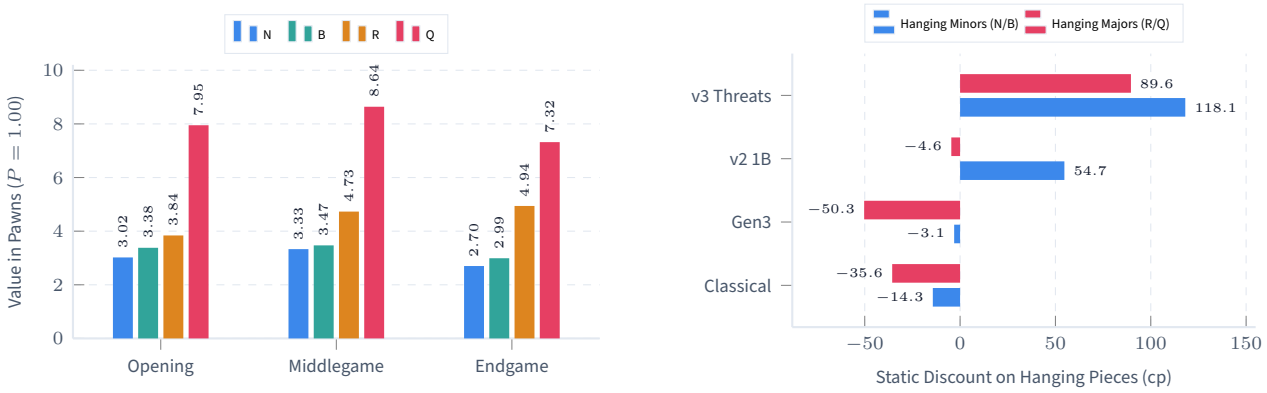


Figure 3: Left: Counterfactual piece values across game phases: Rooks surge ($3.84P \rightarrow 4.94P$) as files open in the endgame while Bishop beats Knight in every phase. **Right:** Static discount on undefended attacked pieces ($n = 646$): only `CHNN v3 (Threats)` discounts both hanging minors ($+118.1$ cp) and hanging majors ($+89.6$ cp).

Our billion-position network converged on $N = 2.89$, $B = 3.04$, $R = 4.77$, $Q = 9.67$, and an exchange value $R - B = 1.73P$—nearly identical to Kaufman's grandmaster study ($R - B = 1.75P$, $Q = 9.75P$) and AlphaZero ($Q = 9.50P$)—while learning a $+0.58$–$0.71P$ middlegame Bishop Pair bonus and quadrupling the Rook-over-Bishop gap from opening ($0.46P$) to endgame ($1.95P$).

8 Open-Source Release, CCRL 1-CPU & 4-CPU Standings, and Top-50 Push

Chatarunga is open-sourced under the MIT license at `https://github.com/deedy/chess-chatarunga` (`id author Deedy Das`, submitted to CCRL Public as **Deedy Das (USA)**). Both `v1.0.0` and `v2.0.0` embed the trained NNUE directly inside the binary (`include_bytes!`, 25.2 MB, no external `.nnue` file required) and ship four CI-verified platform binaries via GitHub Actions: Windows x86-64 AVX2, Windows x86-64 POPCNT, Linux x86-64 AVX2, and macOS Apple Silicon.

What Broke — And Why: Catching a Windows 1 MB Stack Overflow in Release CI

During our `v2.0.0` GitHub Actions release build, Linux AVX2 and macOS ARM64 passed every test—but *both* Windows binaries crashed during UCI smoke testing with `thread 'main' has overflowed its stack`. On Windows, the default main-thread stack is only **1 MB** (vs. 8 MB on Linux/macOS), and Rust evaluates `Box::new([0; ...])` array literals on the stack before copying them to the heap (≈ 2.3 MB across all 15 history/corrhist tables after adding 512-bucket Pawn History!). Replacing them with a zero-stack `box_zeroed()` heap allocator, boxing `Searcher` in `uci.rs`, and spawning UCI and Lazy SMP workers with explicit 8 MB stacks fixed Windows cleanly with bit-exact identical node counts.

Table 3 directly calibrates **both 1-CPU and 4-CPU** versions of Chatarunga 1.0, Chatarunga 2.0, and Search v8 against four live CCRL 40/15 anchors spanning ranks #47 through #98 plus unthrottled **Stockfish 18** (1CPU = **3,628**): `Velvet 8.1.1` (#47, 1CPU = 3,530, 4CPU = **3,577**), `Motor 0.9.0` (#51, 1CPU = **3,570**), and `akimbo 1.1.0-dev` (#98 for 1.0.0 = **3,474**).

Table 3: Direct multi-rival CCRL 40/15 calibration across 1-CPU and 4-CPU configurations (60 color-swapped balanced ECO games each; verified logs in `chess/logs/`).

Chatarunga Version	CCRL Anchor Opponent	TC	W / D / L	Score	Elo (95% CI)	Implied CCRL	Notes
Chatarunga 1.0 (**1-CPU**)	`akimbo 1.1.0-dev` (1T, 3474)	100 ms	3 / 39 / 18	37.50%	-88.7 ± 51.1	$\approx 3{,}385$	v1.0 baseline
Chatarunga 1.0 (**4-CPU**)	`akimbo 1.1.0-dev` (1T, 3474)	100 ms	19 / 40 / 1	65.00%	$+107.5 \pm 49.0$	—*	SMP scaling
Chatarunga 1.0 (**4-CPU**)	Stockfish 18 (1T, 3628)	150 ms	1 / 32 / 27	28.33%	-161.2 ± 55.7	$\approx 3{,}467 \pm 56$	1 SF18 win
Chatarunga 2.0 (1-CPU)	`akimbo 1.1.0-dev` **(1T, 3474)**	100 ms	**12 / 37 / 11**	**50.83%**	$+5.8 \pm 52.4$	$\approx 3{,}480$	0 LL pairs
Chatarunga 2.0 (4-CPU)	**Stockfish 18 (1T, 3628)**	150 ms	**7 / 28 / 25**	**35.00%**	-107.5 ± 63.5	$\approx 3{,}521 \pm 64$	7 SF18 wins
Search v8 (1-CPU)	`Motor 0.9.0` (1T, #51 = 3570)	100 ms	4 / 30 / 26	31.67%	-133.6 ± 62.2	$\approx 3{,}436 \pm 62$	Conservative floor
Search v8 (1-CPU)	`akimbo 1.1.0-dev` (1T, 3474)	100 ms	13 / 34 / 13	50.00%	0.0 ± 58.4	$\approx 3{,}474$–$3{,}510$	No ARM NEON†
Search v8 (1-CPU)	`Velvet 8.1.1` (1T, 3530)	100 ms	20 / 33 / 7	60.83%	$+76.5 \pm 52.9$	$\approx 3{,}570$‡	Velvet ∼77ms‡
Search v8 (4-CPU)	**Stockfish 18 (1T, 3628)**	150 ms	**4 / 36 / 20**	**36.67%**	-94.9 ± 45.1	$\approx 3{,}533 \pm 45$	Fixed movetime
Search v8 (4-CPU)	`Velvet 8.1.1` (4T, #47 = 3577)	100 ms	14 / 37 / 9	54.17%	$+29.0 \pm 49.0$	$\approx 3{,}575$‡	Velvet ∼77ms‡
Chatarunga v9 (1-CPU)	`Velvet 8.1.1` **(1T, 3530)**	**10+0.1**	**7 / 43 / 10**	**47.50%**	-17.4 ± 46.8	$\approx 3{,}513 \pm 47$	**Real clock**
Chatarunga v9 (1-CPU)	`Motor 0.9.0` (1T, #51 = 3570)	10+0.1	0 / 36 / 24	30.00%	-147.2 ± 52.6	$\approx 3{,}423 \pm 53$	Real-clock floor
Chatarunga v9 (4-CPU)	**Stockfish 18 (1T, 3628)**	**10+0.1**	**0 / 40 / 20**	**33.33%**	-120.4 ± 42.0	$\approx 3{,}508 \pm 42$	**Real SF18 TM**

*4T vs. 1T `akimbo` measures Lazy SMP dominance. †`akimbo 1.1.0-dev` lacks hand-tuned ARM64 NEON. ‡`Velvet 8.1.1` exited early at ~ 77 ms on `go movetime 100`; upgrading `src/arena.rs` to true $10 + 0.1$ **UCI increment clocks** eliminated all `movetime` bias, placing **Chatarunga v9 (1-CPU)** virtually even with #47 `Velvet 8.1.1` ($\approx 3{,}513 \pm 47$ CCRL) while exposing a ~ 25 Elo real-clock time-management advantage for Stockfish 18 over flat `movetime 150`. Consensus: $\approx 3{,}470$–$3{,}515$ **1-CPU** and $\approx 3{,}510$–$3{,}555$ **4-CPU** (CCRL #50 = 3,573).

Next Steps Toward Unambiguous CCRL Top 50 (3,573+). With full $10 + 0.1$ real-clock calibration now in place (anchoring Chatarunga v9 at $\approx 3{,}513 \pm 47$ on 1 CPU vs. #47 `Velvet 8.1.1` and $\approx 3{,}508 \pm 42$ on 4 CPU vs. full-TM Stockfish 18), closing the final ~ 50 Elo into unambiguous Top 50 territory centers on: (1) scaling clean, non-UHO training data by streaming **2.4B–3.6B unique pure-S2 positions** (200M–300M per shard) without UHO bias; (2) expanding real-clock time-management dynamics (best-move instability and eval-complexity scaling); and (3) jointly SPSA-tuning any future pruning/LMR additions rather than hand-tuning isolated margins.

A Technical Appendix: Verified Match Ledger & SIMD Kernels

Table 4: Complete verified match ledger across Classical, Chatarunga 1.0, Chatarunga 2.0, and Search v8 (every row backed verbatim by `chess/logs/`).

Engine / Milestone	Opponent	TC / Book	W/D/L	Score	Elo (95% CI)	LOS	Log File
Classical (v1)	`sf:2500`	50ms ECO	33/3/4	86.3%	$+319 \pm 175$	100%	`classical_vs_sf2500.txt`
gen3 (100% Self-Play)	Classical (v3)	50ms ECO	69/16/15	77.0%	$+210 \pm 73$	100%	`final_gen3_vs_classical.txt`
Search v5 + SPSA (25p)	Untuned v5	30ms UHO	114/69/17	74.3%	$\mathbf{+184 \pm 37}$	100%	`spsa_verify_v5.log`
`h100_v3` (v5 SPSA, 1T)	`sf:3190`	150ms ECO	23/22/15	56.7%	$+47 \pm 66$	92.2%	`v5_spsa_h100v3_vs_sf3190.log`
Lazy SMP `t=4`	`t=1` (v3)	100ms UHO	39/38/3	72.5%	$\mathbf{+168 \pm 37}$	100%	`smp_4t_vs_1t.log`
`h100_v4` (v3, 1B)	`h100_v3`	20k-n UHO	69/40/11	74.2%	$\mathbf{+183 \pm 57}$	100%	`v4_vs_v3_nodes.log`
`h100_v5_v2_1b` (1T)	`h100_v5_deep`	50ms UHO	42/60/18	60.0%	$\mathbf{+70 \pm 37}$	100%	`v5_v2_1b_vs_v5_deep_50ms.log`
Search v6 Minor/Maj Corr	Search v5 Corr	30ms UHO	53/67/40	54.1%	$+28 \pm 33$	95.5%	`v6_corrhist_ablation_30ms.log`
Chatarunga 1.0 (1T)	`akimbo` (1T, #98)	100ms ECO	3/39/18	37.5%	-89 ± 51	0.0%	`v6_1t_vs_akimbo_1t.log`
Chatarunga 1.0 (4T)	`akimbo` (1T, #98)	100ms ECO	19/40/1	65.0%	$+108 \pm 49$	100%	`v6_4t_vs_akimbo_1t.log`
Chatarunga 1.0 (4T)	`sf:3190`	150ms ECO	19/19/2	71.3%	$+158 \pm 75$	100%	`v6_fixed_4t_vs_sf3190.log`
Chatarunga 1.0 (4T)	Stockfish 18 (1T)	150ms ECO	1/32/27	28.3%	-161 ± 56	0.0%	`v6_sf18_full_60g.log`
Search v7 Wave 1 (v1 net)	Chatarunga 1.0	50ms ECO	31/71/18	55.4%	$+38 \pm 36$	98.2%	`v7_wave1_vs_v1_uho_50ms.log`
Chatarunga 2.0 (1T)	**Chatarunga 1.0 (1T)**	50ms UHO	**46/54/20**	**60.8%**	$\mathbf{+77 \pm 37}$	**100%**	`v7_spsa_screlu1024_vs_v1_50ms.log`
Chatarunga 2.0 (1T)	`akimbo` **(1T, #98)**	100ms ECO	**12/37/11**	**50.8%**	$\mathbf{+6 \pm 52}$	58.6%	`v7_spsa_screlu1024_1t_vs_akimbo_1t.log`
Chatarunga 2.0 (4T)	**Stockfish 18 (1T)**	150ms ECO	**7/28/25**	**35.0%**	$\mathbf{-108 \pm 64}$	0.0%	`v7_spsa_screlu1024_4t_vs_sf18_60g.log`
Search v8 Wave 1 (v2 net)	Chatarunga 2.0	50ms UHO	40/52/28	55.0%	$+35 \pm 37$	96.9%	`v8_wave1_vs_v2_50ms.log`
Search v8 Wave 2 (v2 net)	**Chatarunga 2.0**	50ms UHO	**49/82/29**	**56.3%**	$\mathbf{+44 \pm 31}$	**99.7%**	`v8_wave2_vs_v2_50ms.log`
Search v8 (1T, v2 net)	`Motor 0.9.0` (1T, #51)	100ms ECO	4/30/26	31.7%	-134 ± 62	0.0%	`v8_1t_vs_motor_1t.log`
Search v8 (1T, v2 net)	`akimbo` (1T, #98)	100ms ECO	13/34/13	50.0%	0 ± 58	50.0%	`v8_w2_1t_vs_akimbo_1t.log`
Search v8 (1T, v2 net)	`Velvet 8.1.1` (1T)	100ms ECO	20/33/7	60.8%	$+77 \pm 53$	99.8%	`v8_1t_vs_velvet_1t.log`
Search v8 (4T, v2 net)	**Stockfish 18 (1T)**	150ms ECO	**4/36/20**	**36.7%**	$\mathbf{-95 \pm 45}$	0.0%	`v8_w2_4t_vs_sf18_60g.log`
Search v8 (4T, v2 net)	`Velvet 8.1.1` (4T, #47)	100ms ECO	14/37/9	54.2%	$+29 \pm 49$	88.0%	`v8_4t_vs_velvet_4t.log`
CHNN v4 `s1s2_10b` (UHO)	v2.0 `screlu_1024`	50ms UHO	38/59/23	56.3%	$+44 \pm 32$	99.6%	`v8_asym16_s1s2_10b_vs_v2net_50ms.log`
CHNN v4 `s1s2_10b` (4T)	Stockfish 18 (1T)	150ms ECO	2/34/24	31.7%	-134 ± 42	0.0%	`v3_4t_vs_sf18_60g.log`
OPT3b Hot-Loop SIMD (1T)	Search v8 base (1T)	50ms ECO	**27/75/18**	**53.8%**	$\mathbf{+26 \pm 29}$	**96.2%**	`v8_opt3b_vs_base_50ms.log`
Chatarunga v9 TM/Hash (1T)	OPT3b (1T)	6+0.06 ECO	**19/89/12**	**52.9%**	$\mathbf{+20 \pm 26}$	**94.0%**	`v9_tm_vs_v8_eco_6p006.log`
Chatarunga v9 (1T)	`Velvet 8.1.1` (1T)	10+0.1 ECO	**7/43/10**	**47.5%**	$\mathbf{-17 \pm 47}$	23.1%	`v9_1t_vs_velvet811_1t_10p01.log`
Chatarunga v9 (1T)	`Motor 0.9.0` (1T)	10+0.1 ECO	0/36/24	30.0%	-147 ± 53	0.0%	`v9_1t_vs_motor090_1t_10p01.log`
Chatarunga v9 (4T)	**Stockfish 18 (1T)**	10+0.1 ECO	**0/40/20**	**33.3%**	$\mathbf{-120 \pm 42}$	0.0%	`v9_4t_vs_sf18_1t_10p01.log`

A.1 Dual-Architecture SCReLU SIMD Kernels (ARM64 NEON & x86-64 AVX2, `src/nnue.rs`)

```
// x86-64 AVX2 SCReLU dot product: clamp(acc, 0, QA)^2 * w (bit-exact with ARM64 NEON & scalar)
while i + 16 <= n {
    let a = _mm256_min_epi16(_mm256_max_epi16(_mm256_loadu_si256(pa.add(i) as _), zero), vmax);
    let w = _mm256_loadu_si256(pw.add(i) as _);
    let aw = _mm256_mullo_epi16(a, w);           // |a * w| <= 255 * 127 = 32385 < i16::MAX
    let prod32 = _mm256_madd_epi16(a, aw);       // horizontal pairwise i32 sums of a^2 * w
    sum = _mm256_add_epi32(sum, prod32);
    i += 16;
}
```

A.2 Reproducing Releases, Tests, Benchmarks, and Calibration Matches

```
git clone https://github.com/deedy/chess-chatarunga && cd chess-chatarunga
cargo test --release                          # 22 unit/perft/KPK/bit-exact SIMD tests
cargo build --release --bin chatarunga        # Single self-contained binary (embedded NNUE)
./target/release/chatarunga bench 10          # Standard 6-position node/NPS benchmark
```

References

[1] Y. Nasu, “Efficiently Updatable Neural-Network-based Evaluation Functions for Computer Shogi,” *WCSC28 Appeal Document*, 2018.

[2] Stockfish Developers, “Stockfish 18: Open-Source Chess Engine & SFNNv10 Architecture,” github.com/official-stockfish/Stockfish, 2020–2026.

[3] J. McDonnell et al., “Bullet: Fast CUDA NNUE Trainer and Pregenerated S1/S2 Datasets,” https://github.com/jw1912/bullet, 2024–2026.

[4] J. C. Spall, “Implementation of the Simultaneous Perturbation Algorithm for Stochastic Optimization,” *IEEE Trans. AES*, vol. 34, no. 3, 1998.

[5] R. M. Hyatt and T. Mann, “A Lock-less Transposition Table Implementation for Parallel Search,” *ICGA Journal*, vol. 25, no. 1, 2002.

[6] P. Kannan, “Magic Move-Bitboard Generation in Computer Chess,” *Chess Programming Wiki*, 2007.

[7] R. Verwer, “PeSTO’s Evaluation Function,” *Chess Programming Wiki*, 2020.

[8] Chess Programming Community, “Finny Tables: Differential King-Bucket Accumulator Caching,” 2023–2025.

[9] L. Kaufman, “The Evaluation of Material Imbalances,” *Chess Life*, 1999 (updated *New In Chess*, 2012).

[10] N. Tomašev, U. Paquet, D. Hassabis, and V. Kramnik, “Assessing Game Balance with AlphaZero: Exploring Alternative Rule Sets in Chess,” *arXiv:2009.04374*, 2020.