Efficient Thinking: Measuring What Capability Costs

Parameters, data, search, and latency in a small chess evaluator — and a diagnostic for finding which one binds

Louay Alsakka · July 2026

Code & trained model: github.com/louayalsakka/efficient-thinking

Abstract

Every AI system buys capability with the same finite resources — parameters, training data, inference-time compute, latency — and to a large degree they substitute for one another. This paper measures those tradeoffs directly, using chess as a fully observable testbed: one small convolutional evaluator (3.45M parameters, 14 MB), one fixed evaluation protocol, and one lever varied at a time.

The central result is the size of the search lever on a frozen evaluator. On the same Stockfish ladder, adding MCTS lifts the raw policy by **+286 Elo at 800 simulations** (2448 → 2734), then a steady **~+55 Elo per doubling**, unsaturated through 12,800 simulations (2967, **+519 total**) — with zero additional parameters. Over the same period, growing the evaluator 1.4–2× at full data moved strength by an amount indistinguishable from noise, while 8× more training data moved it ~+90. In this regime the lever ranking is unambiguous: **search** ≫ **data** ≫ **capacity**. Adaptive MCTS also beats fixed-depth alpha-beta at equal compute and keeps scaling where fixed depth plateaus (~2550).

A third study removes external supervision entirely. Self-play, a self-referential ladder, evolutionary selection, plurality voting, and weight merging all fail to lift the evaluator past its plateau at our two-machine scale; the one robust positive is that **inter-model agreement predicts correctness** (unanimous committees match Stockfish's best move 94% of the time vs 37% when all disagree) — a teacher-free confidence signal, though not an accuracy booster. These negatives are scoped: they characterize the small-compute regime, not self-play in general.

The unifying reading is a decomposition — **strength = evaluator × search**. Search converts what the evaluator already encodes into better decisions; the evaluator's ceiling is set by the quality of the information it was trained on; and in every closed loop we ran, nothing inside

the system raised that ceiling — only an external oracle did. Whether that is a law or a limitation of our loops is an open question: a system might in principle improve its evaluator by better extracting information it already holds (coherence, calibration), and our experiments bound what we observed, not what is possible. The transferable contribution is the method: at each stage exactly one resource binds, which one is not knowable a priori, and spending anywhere else returns almost nothing — so measure first, then spend. We also report, as a case study in that method, how a coarse ±89 rating ladder manufactured an apparent 4.8× search-efficiency win that a rigorous paired head-to-head then eliminated (§4.2).

On calibration: absolute Elo here is internal ladder Elo — Stockfish rungs at fast movetime, with ±~100 systematic uncertainty near the top rung — and is not calibrated to over-the-board ratings. The searched system's ~2800-class ladder rating is consistent with the top-human band, but every claim in this paper is a relative, same-ladder comparison, which does not depend on that calibration.

Key Takeaways

1. **Search dominates parameters in this regime.** On a frozen 3.45M evaluator, MCTS adds +286 Elo at 800 sims and ~+55 per doubling thereafter, unsaturated at 16× the base budget (+519 total, same ladder). Doubling parameters at full data added nothing distinguishable from noise. Strength came from thinking, not growing.

2. **Adaptive search out-scales fixed-depth search.** At equal compute MCTS beats alpha-beta, and it keeps climbing where fixed depth plateaus — uniform depth amplifies the value net's noise; adaptive allocation averages over it — minimax propagates the single most optimistic evaluation error to the root, while MCTS's visit-weighted averaging cancels errors across simulations (Appendix A).

3. **Data beats capacity when the evaluator binds.** 8× more training data moved strength ~+90; 1.4–2× more parameters moved it ~0. Capacity is the weakest lever in this data regime — a scoped claim, not "parameters never matter."

4. **Model agreement predicts correctness.** Independently trained models that agree are far more often right (unanimous: 94% match with Stockfish-best, 19.7 mean CPL; full disagreement: 37%, 77.9 CPL) — a teacher-free confidence meter. But plurality voting never beat the best member: correlated errors don't cancel.

5. **Measurement noise manufactures results — twice, in this study.** A wide→narrow MCTS cascade appeared 4.8× cheaper at equal strength on a ±89 ladder; a paired head-to-head showed no net gain (§4.2). Evolution appeared to escape the self-play plateau by +47 Elo; a clean 400-game re-match showed −24 to −41. The paper's

operating rule fell out of these: no delta is believed until it survives a low-variance re-measurement.

---

1. Introduction

Every AI system faces an economic problem. Capability is purchased with finite resources — model capacity (memory), training data, inference-time computation, latency, and human supervision. The engineering objective is not to maximize any one resource but to **maximize capability per unit cost**: more capacity, data, search, or supervision each raise strength, but each carries a cost and to a large degree they *substitute* for one another. This paper presents an **empirical framework for measuring those tradeoffs**, using chess as a clean, fully-observable testbed — we fix the objective (playing strength) and measure what each resource buys, and where spending on one is wasted because a *different* resource is the binding constraint. We claim **no solved optimal-allocation rule**; the contribution is the framework and a set of *measured tradeoff curves*.

The argument is a three-level hierarchy: **(1) AI resource allocation** — the main thesis; **(2) the evaluator–search decomposition** — the analytical model of how capacity and search combine into strength; and **(3) binding-bottleneck analysis** — the diagnostic that identifies, at any stage, which single resource to spend on next.

The experiments are **three allocation questions** of increasing autonomy: 1. **Stage 1 (open loop):** given a fixed memory budget, **what architecture buys the most capability?** 2. **Stage 2 (closed loop):** given a fixed evaluator, **what is the cheapest way to buy additional strength with inference-time computation?** 3. **Stage 3 (self-learning):** without external labels, **what is the most efficient way to improve the evaluator?**

We measure each independently. The framing is also **control-theoretic** (Bertsekas 2022, *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*): open-loop policy = feedforward controller, closed-loop search = model-predictive control, self-play = iterative/adaptive learning control. Chess is only the testbed; the goal is a *generic* way to reason about resource tradeoffs in sequential decision-making.

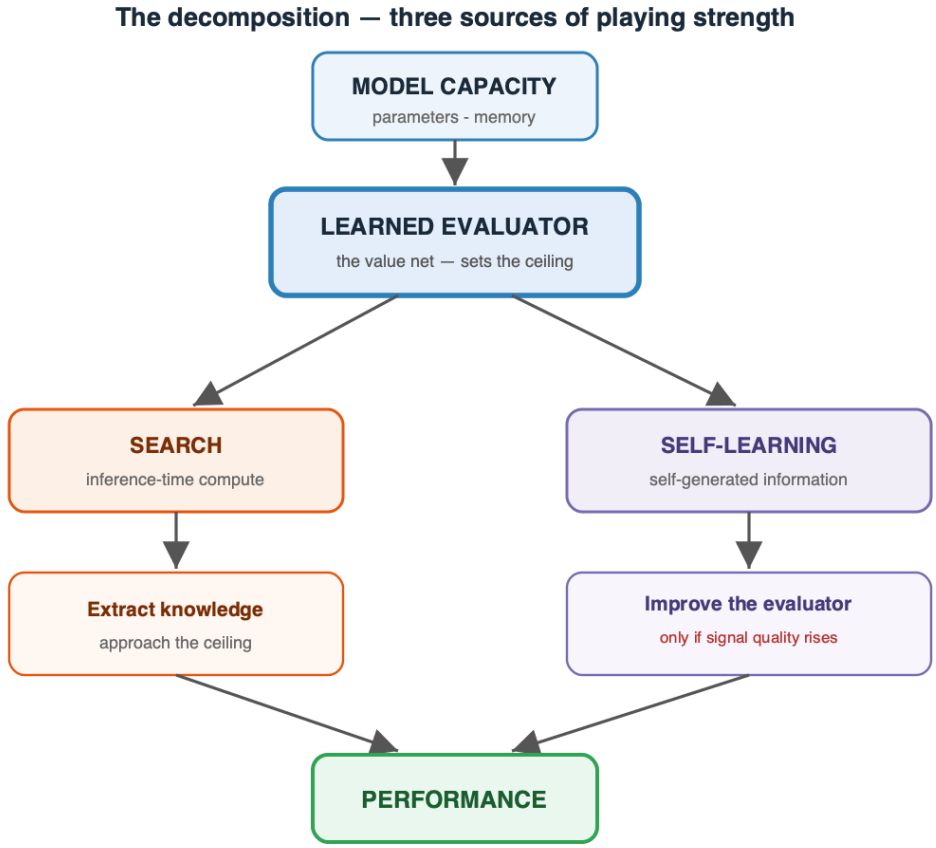
The central *tool* for efficient allocation is a simple diagnostic: **at any given stage, strength is gated by a *single binding resource* — capacity, search, data, or the quality of self-generated signal — and spending on any *non-binding* resource returns almost nothing.** So the efficient move is always to **identify the binding resource experimentally, then spend there.** Which one binds is not obvious a priori and shifts as you relieve each (open-loop is capacity-bound; add search and you ride it up log-linearly until the evaluator's quality caps the return; then the evaluator — its *data*, not its parameter count in our regime — binds). This bottleneck analysis is not the paper's goal but its **method**: one instrument in the larger objective of spending a fixed budget efficiently.

Target calibration. We aim at the **~2800 band on our internal ladder — consistent with the top-human class — by design.** For human-facing applications, matching top-human capability is the efficient saturation point; pushing toward machine-only territory spends compute where it stops mattering. The question is therefore *how small a model, plus how little search, reaches that band* — efficiency at the human ceiling, not absolute strength. (All absolute figures in this paper are internal ladder Elo; see the calibration caveat in §2.3.)

How to read the numbers. Absolute Elo is measured against a Stockfish ladder and carries **±~100 systematic uncertainty** near the top rung; treat "~2800" as a headline calibrated to the top-human band on this internal ladder. The paper's *claims* are the **relative, same-ladder** results, which do not depend on that calibration: search adds **+286 Elo** over the raw policy, MCTS out-scales fixed-depth search, and every Stage-3 aggregation method fails to beat a single model. (§4.2 documents, as a case study, how this ladder's resolution once manufactured an apparent efficiency win.) We further separate **established** results (robust, relative — adaptive search scaling, agreement-predicts-correctness) from **regime-limited observations** true only at our compute (the self-play plateau, flat parameter scaling, evolution's non-escape), which we expect to change at AlphaZero/LLM scale.

Contributions. - **A measured lever ranking for a fixed capability target:** on one evaluator and one ladder, search (+286, then ~+55/doubling, unsaturated), data (~+90 for 8×), capacity (~0 for 2× at full data) — with the diagnostic that produced it: identify the binding resource experimentally, relieve exactly that, re-measure, repeat. - **A direct MCTS-vs-fixed-depth comparison** on the same value net: adaptive search wins at equal compute and keeps scaling where fixed depth plateaus. - **An architecture-beats-scale result:** a 3.45M convolutional net matches a 14.4M MLP with 22× less data; every tapered or bottlenecked topology loses badly. The right inductive prior, not width, sets open-loop strength. - **A teacher-free confidence signal, validated:** committee agreement predicts correctness — alongside clean negatives showing that plurality voting, weight merging, and in-search ensemble averaging all fail to beat a single model, because member errors are too correlated to cancel. - **Reproducible negative results on small-scale self-improvement:** self-play, a self-referential ladder, and evolution all plateau below supervision at two-machine scale, with the failure mechanism identified in each case. - **Two documented measurement-artifact autopsies** (the cascade, §4.2; evolution's phantom escape, §5.3) — worked examples of how coarse or noisy evaluation manufactures effects, and the re-measurement protocol that catches them.

The paper's roadmap in one picture — the three sources of strength, all feeding a single learned evaluator:



2. Problem Setup and Methods

2.1 Representation

- **Position → move** as a 4096-way (64×64 from–to) classification (promotions default to queen).
- **Side-to-move normalization:** the board is re-oriented so the mover is always "White" (mirror + color-swap for Black), so one network serves both players and the turn needs no bit. (Hence `Eval(P) = Eval(mirror(P))` exactly, and a move's value reads off as `Eval(B) + Eval(null) − 1`; measured mean tempo ≈ +0.15, up to +0.77 in tactical shots, negative in zugzwang.)
- **Encoding:** *one-hot* = 64×12 piece planes + 5 meta = **773 floats** (reshaped to 8×8×12 + 5 = 17 channels for convolution), used throughout. Against a compact *packed* encoding (one 4-bit code per square, 69 floats) at equal budget and data, one-hot is decisively stronger (**84.9 vs 142.5 mean CPL**): nets learn categorical piece-type features far more readily than an arbitrary ordinal code, and convolution *requires* the per-type planes a packed board cannot provide.

2.2 Labels and objective

- **Supervised data:** the full public Lichess cloud-eval database — **394,669,566 positions** (79 shards), each with Stockfish multi-PV win-probabilities.
- **Hard objective:** cross-entropy to the single best move (bounded by imitation).
- **Soft objective:** cross-entropy to an *advantage-weighted distribution* over legal moves (`softmax(vᵢ/τ)`). Soft is not bounded by copying one move and generalizes better.

2.3 Evaluation

- **Elo via a Stockfish ladder** (random baseline, then SF at set UCI_Elo rungs), 20–80 games/rung; Elo fit by maximum-likelihood against the known rung ratings.
- **Methodological caveat (important):** if the player beats the top rung, the Elo estimate **compresses/extrapolates**, so we raised the ladder as the player improved (SF-1700 → 2100 → 2700 → 3000). Absolute numbers carry systematic uncertainty; **relative gains measured on the same ladder are the robust results** (treat absolutes as ±100). Where two methods are compared, they are always run on the *same* ladder and seeds.

2.4 Hardware

- Two Apple-Silicon **Mac Studios (M3 Ultra, 256 GB each)**, MLX framework, 40 Gbps bridge.

2.5 Reproducibility (evaluation protocol)

A fixed protocol makes every Elo/CPL figure comparable across methods: - **Engine:** Stockfish 18 as ladder opponent and CPL oracle; opponents at fixed `UCI_Elo` rungs, movetime **0.03–0.04 s**; CPL at **fixed depth 12**. - **Ladder:** a random anchor plus `UCI_Elo` rungs (e.g. 1700/2000/2300, 2400/2700/3000), Elo by maximum-likelihood fit; **20–40 games/rung**; margin ≈ $\pm 400/\sqrt{n}\cdot 2$ (**±100** typical) — rely on *relative* same-ladder deltas. - **Openings/seeds:** sampled from the Lichess `2013-01` PGN, replayed to plies 6–16, both colors; seed 0 by default, and any two compared methods use the **same** ladder, openings, and seeds. - **Caveats:** the ladder **compresses** once the player beats the top rung (we raised it as strength grew), and at 0.03–0.04 s Stockfish plays **below** nominal `UCI_Elo` — internally consistent, not calibrated to over-the-board Elo. Exact scripts are in the repo.

Table 1 — canonical strength measurements (high ladder: SF 2500/2800/3050; every Elo claim in this paper routes through this table).

Configuration	Ladder Elo	Δ vs raw
raw policy (1 forward pass)	2448	—
MCTS-800	2734 ±76	+286

Configuration	Ladder Elo	Δ vs raw
MCTS-1600	2780 ±82	+332
MCTS-3200	2839 ±76	+391
MCTS-6400	2903 ±82	+455
MCTS-12800	2967 ±115	+519

The rounded "~2150 open-loop / ~2800 searched" figures used in prose refer to the earlier calibration-band ladder and sit within the stated ±100 of these values; where precision matters, Table 1 is authoritative.

---

3. Stage 1 — Open Loop (single-pass policy)

3.1 Topology sweep — architecture beats scale

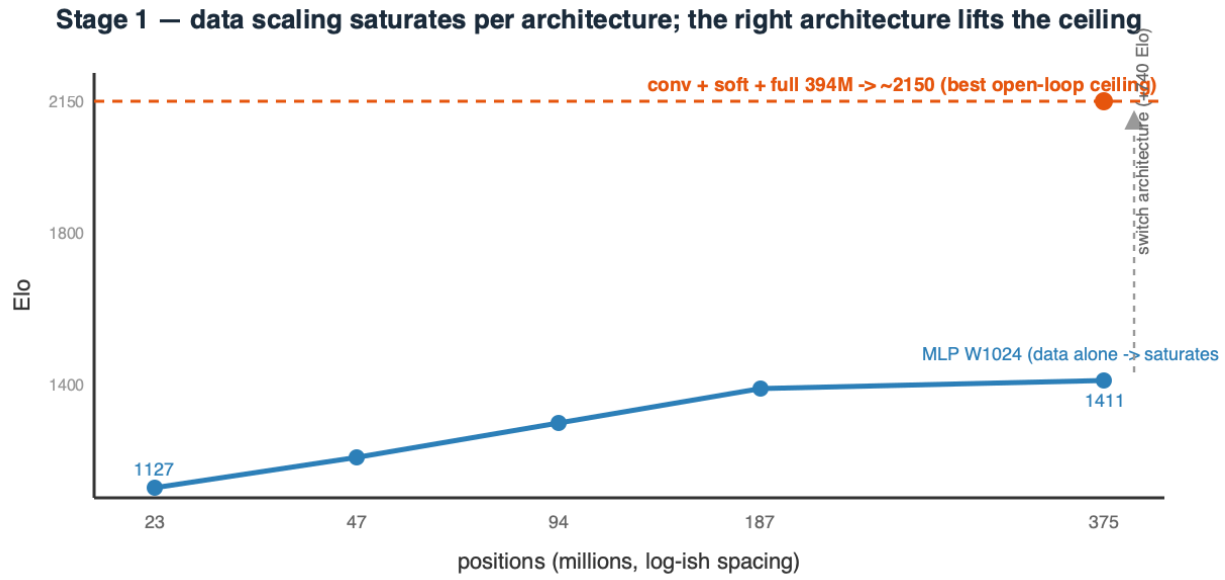
At fixed moderate data (18M positions) we swept eight architectures:

Topology	params	Elo	note
conv (64×10 / 96×8)	2.8–3.4M	**1476**	best & most efficient
constant-width MLP (1024×6)	10.2M	1108–1233	best plain MLP
dual-path (wide+deep, gated)	2.8M	1082	ties MLP at 3.5× fewer params
factored head (from/to)	6.2M	782	−450
funnel (1024→64)	1.8M	582	narrowing hurts
pyramid (64→1024)	5.0M	431	bad
bottleneck (512→8)	0.5M	340	broken

Topology conclusion. Only **constant-width** and **convolution** are competitive; every taper, factoring, or bottleneck loses badly, because it destroys positional information before the head uses it, while convolution **reuses local patterns via weight sharing** (learns a motif once, not per square). The rule: **match the prior to the domain's structure (spatial locality) rather than adding width.** A 3.45M conv matched a 14.4M MLP using **22× less data** — architecture, not parameter count, set the strength.

3.2 Scaling laws (width and data)

- **Width (MLP, hard):** W1024 (14.4M) → 1449; W2048 (47.7M) → 1501: **+52 Elo for 3.3× params** — sharply diminishing. Top-1 accuracy saturates ~0.41 regardless of width.
- **Data (W1024):** 1127 / 1207 / 1300 / 1382 / 1411 at 23 / 47 / 94 / 187 / **375M** positions — large early gains, then **+29 on the last doubling** → saturates near the DB's 394M limit.



3.3 Objective and open-loop ceiling

Soft beats hard by **+94–113 Elo** at subset scale; top-1 saturates while Elo keeps improving via blunder-rate reduction (**top-1 is a misleading metric**). Best open-loop recipe = **conv + soft + full 394M data**, with a **ceiling ≈ 2150 Elo**. Cost: 3.45M params, **14 MB**, **~176 MFLOP / ~1.5 ms per move** — cheap, but capped: no amount of width or data pushed a single pass past ~2150.

---

4. Stage 2 — Closed Loop / Inference-Time Compute (search on a value function)

We add a scalar head **Eval(N) ∈ [0,1]** = *expected score for the side to move* (held-out **MAE 0.088, correlation 0.877** — an excellent value function). Search uses the zero-sum identity — our value after a move is `1 - Eval(child)` — so one network plays both sides; terminals return exact 0 / 0.5 / 1.

4.1 MCTS vs fixed-depth — the core comparison

We compared two search families on the same value net and ladder:

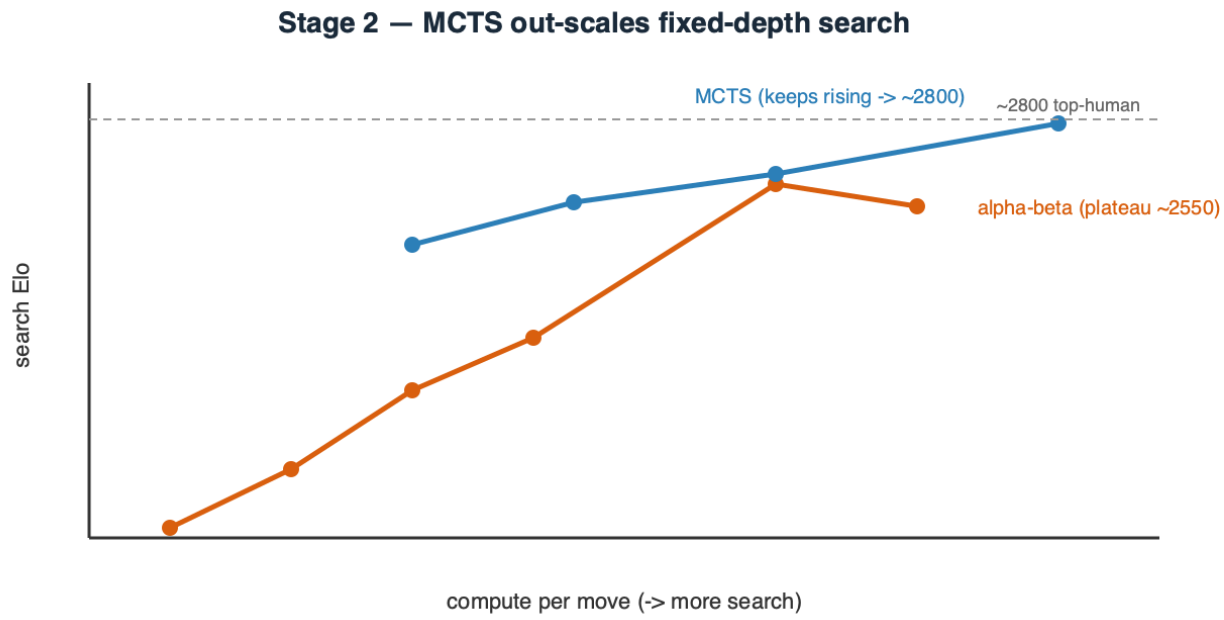
- **Fixed-depth alpha-beta** (negamax, policy move-ordering, quiescence at leaves):

depth	raw	search	gain
1	1661	1627	**−34** (1-ply hurts — can't see the reply)
2	1685	1788	+103
3	1895	2005	+110
4	1914	2152	+238
6	2128	**2575**	+447
7	2187	2513	plateau ~2550

- **Adaptive MCTS/PUCT** (`Q + c·P·√N/(1+n)` , value-head leaves, negamax backup):

sims	search Elo
100	2411
200	2530
400	2610 (2661 @ 40 games/rung)
800	**2749 (the ~2800 ladder band)**

Result. (i) **1-ply search *hurts* a strong policy** — it commits to a capture without seeing the recapture; depth is where lookahead pays. (ii) **MCTS beats alpha-beta at equal compute and keeps scaling** where fixed depth plateaus (~2550): uniform depth spends equal effort on every branch and *amplifies the value net's noise*, while MCTS **allocates search adaptively to sharp lines and averages over it.** MCTS wins Stage 2, reaching ~2800 on the fixed 3.45M net.



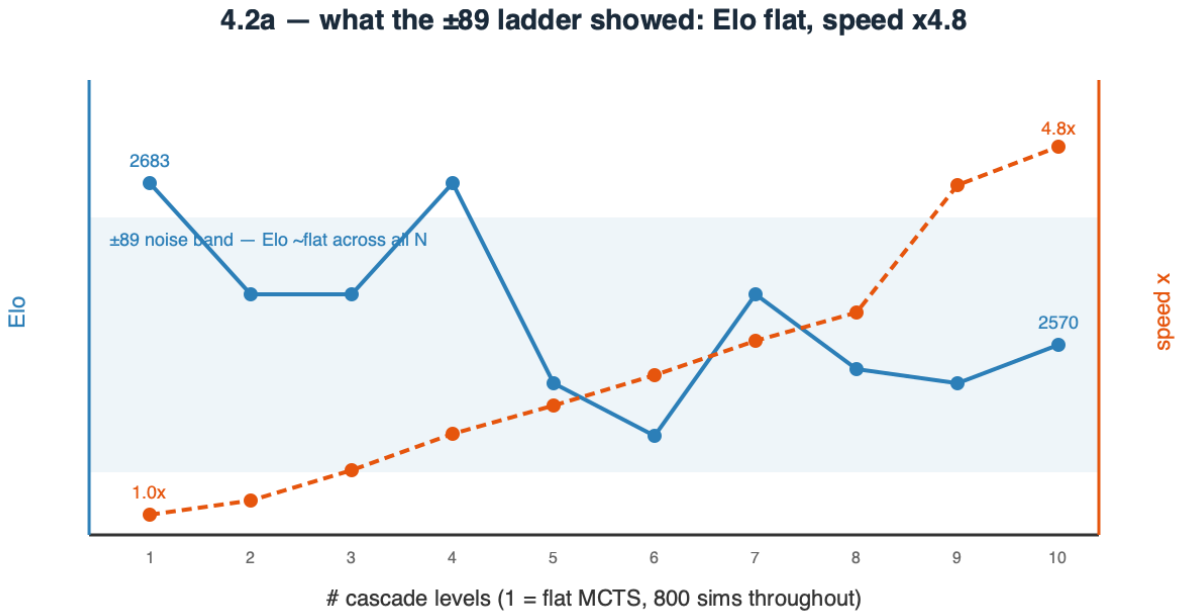
4.2 Does allocation of a fixed search budget matter? (a case study in measurement noise)

Given that MCTS wins, a natural efficiency question follows: for a fixed simulation budget, does the *shape* of the search matter? We tested a wide→narrow **cascade**: a wide stage (all moves, high `c_puct`, few sims) ranks candidates broadly, passes its top-k by visit count to a narrower, deeper stage, and so on — funnelling the budget onto surviving lines, with a shared eval cache carrying value-net calls forward. The design mirrors how strong human players allocate finite calculation: a wide intuitive scan, progressive pruning, then one or two lines carried deep. (We claim the allocation principle, not a cognitive model.)

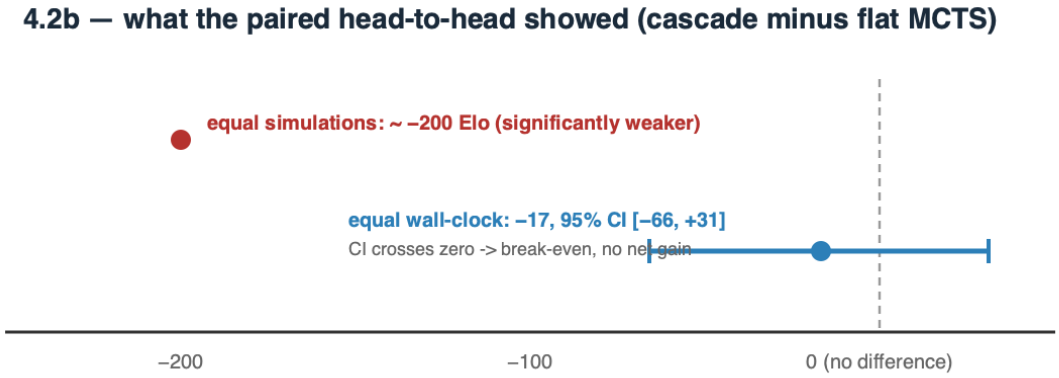
What the ladder said. A controlled N = 1→10 sweep (one rule generating each funnel; 800 total sims; same ladder, seeds, and openings) showed Elo statistically flat across all funnels (2506–2683, all within a single ±89 band) while wall-clock fell monotonically to **4.8× faster** at N = 10:

# levels	Elo (±89)	ms/move	speedup
1 (flat MCTS)	2683	1330	1.0×
3	2605	903	1.5×
6	2506	541	2.5×
9	2543	300	4.4×
10	2570	275	**4.8×**
beam-minimax cascade (fixed depth)	2487	—	inferior primitive

On the ladder, this read as a near-pure efficiency win: flat-MCTS strength at ~5× less compute.



What a rigorous test said. The ladder's ±89 resolution is the whole story. A paired head-to-head — cascade vs. flat MCTS directly, thousands of games, matched openings and seeds — found the cascade is **significantly weaker at equal simulations (≈ −200 Elo)** and only statistically indistinguishable at equal wall-clock (**−17 Elo, 95% CI [−66, +31]**). The correct conclusion is **no net efficiency gain**: the cascade trades strength for speed at roughly par. The "4.8× at equal strength" was an artifact of a rating instrument too coarse to resolve a 200-Elo difference sitting inside its noise band.



What survives. Two things. First, a negative allocation result with value of its own: at a fixed budget, MCTS's default allocation is already near-efficient — reshaping it buys speed only by paying strength, and the fixed-depth beam variant (2487) confirms the adaptive stages are what matter. Second, a methodological rule this paper then applied everywhere: an efficiency claim of the form "same strength, less compute" is only as strong as the equal-strength measurement, and a ladder rating with ±89 error cannot certify equal strength between systems that may differ by 200. Every subsequent within-noise delta in this paper (the

capacity sweep's +60, evolution's +47 in §5.3) was therefore re-measured with paired, low-variance protocols before being believed — and both, like the cascade, dissolved.

4.3 The two-dimensional cost (memory vs GPU-cycles)

	Stage 1 (open)	Stage 2 (closed)
Memory	14 MB	**14 MB — search adds ~0**
GPU cycles/move (batch-1)	1 pass, ~1.5 ms	~800 passes, ~1.3 s (batch-1)
GPU cycles/move (batched, §4.4)	—	**~6–12× less — MCTS-3200 below batch-1 MCTS-800**
Elo	~2150 (capped)	**~2800 (scales with compute)**

Stage 1 buys Elo with *memory* and saturates; Stage 2 buys Elo with *GPU cycles* and keeps climbing (reallocating the same budget wide→narrow buys speed only at par strength — §4.2). Note that the ~1.3 s is a **batch-1 implementation artefact**, not the method's cost: batched-leaf evaluation (§4.4, measured 6–12×) makes even MCTS-3200 cheaper than batch-1 MCTS-800, so the true latency axis sits far below what we plot (clean solo-GPU figure pending). The central practical result stands regardless: **strength is compute, not parameters.**

4.4 Search latency — what a second of thinking costs, and buys

Strength from search is not free: every simulation is a neural-network forward pass, so Elo is paid for in wall-clock. Measured on the Mac Studio (M3 Ultra, MLX) across the three operating points:

Mode	Elo (abs. ladder)	Latency / move
Open-loop (raw policy)	~2448	**~2 ms**
MCTS-800	**2734 ±76**	**~1.3 s**
MCTS-1600	**2780 ±82**	~1.9 s
MCTS-3200	**2839 ±76**	~3.8 s
MCTS-6400	**2903 ±82**	~7 s
MCTS-12800	**2967 ±115**	~14 s

All on the same Stockfish high ladder (2500/2800/3050). The round ~2150 / ~2800 headline figures used elsewhere sit within the stated ±100 ladder uncertainty of these precise values.

Three observations:

1. Search scales *log-linearly* — ~+55 Elo per doubling, no saturation through 12800. The first slice is huge and cheap (MCTS-800 alone adds **+286** over the raw policy, 2448→2734); past that, each *doubling* adds a steady **~+55–64 Elo** (2734→2780→2839→2903→2967), unbroken to **16× the base budget** — cumulative **+233**, far above the ±~100 noise. *(An earlier "saturation at 3200" claim came from a noisy head-to-head sweep, +12/+260/+191 per rung; the clean absolute curve supersedes it, and head-to-head deltas inflate via ceiling compression — 3200-vs-800 reads +260 h2h but +105 absolute.)* So with a fixed evaluator search keeps paying, and we **never reach its ceiling** (tested to 16×): marginal Elo *per compute* falls, but the curve does not flatten.

2. Latency scales *sub-linearly* with sims — a red flag, not a feature. MCTS-3200 does 4× the simulations of MCTS-800 yet is only ~2.8× slower. The cause: this search evaluates leaves **one position at a time (batch = 1)**, so each forward pass is dominated by fixed GPU-launch overhead rather than compute — the M3 Ultra is massively under-utilised. Effective throughput is only **~600 leaf-evaluations per second**, and adding sims mostly amortises the fixed per-move cost.

3. The latency is an implementation artefact, not a hardware or method limit. Batched neural-MCTS engines evaluate many leaves per forward pass and reach **~10k–80k nodes/second** (Leela; AlphaZero on TPUs) — 20–100× our batch-1 throughput. We implemented and measured that fix (next), and it changes none of the strength conclusions, only their price.

GPU utilisation — batch-1 is an implementation *limitation*, and we measured the fix. The batch-1 search (obs 2) leaves the M3 Ultra's cores **idle between launches** at only **~600 nps**, so the latencies above are **pessimistic upper bounds**: strength is fixed by **N (nodes searched)**, latency by **N ÷ throughput**, and our throughput is on the floor.

The standard fix is **batched-leaf evaluation** (gather many tree leaves via *virtual loss*, evaluate them in a single GPU launch). We implemented it (`BatchedMCTSPlayer` , `search.py`) and **measured** it on the same net and positions — the same nodes searched, only the launch pattern changed:

Search	Throughput	Speedup vs batch-1
batch-1 (as used above)	633 nps	1.0×
batched, 16 leaves/launch	3.9k nps	6.2×
batched, 32 leaves/launch	4.7k nps	7.5×
batched, 64 leaves/launch	5.5k nps	8.6×
batched, 128 leaves/launch	7.4k nps	**11.6×**

(clean solo-GPU measurement on the M3 Ultra; a first reading taken under concurrent load over-stated the ratio, so we report the solo figures.) A **~6–12×** per-move speedup at identical strength is enough to run **MCTS-3200 (~0.4 s/move batched) well below today's batch-1 MCTS-800 (~1.3 s)**. **The batch-1 numbers should therefore be read as a ceiling on cost, not the method's efficiency**, and the true strength-vs-latency curve sits well to the left of the one we plot.

Two caveats. First, the gain is **hardware-dependent** — it equals the *idle parallelism you can reclaim*: the wide M3 Ultra leaves much for a 14 MB batch-1 workload, but on a small GPU/CPU or an accelerator already **saturated by a large network** (AlphaZero/Leela), there are no idle cores and deeper search costs full, **linear** price. Second, batched selection uses momentarily stale tree stats, so it is a hair less sample-efficient per node — second-order, not changing the order-of-magnitude speedup.

4.5 Does a bigger evaluator raise the ceiling? (a capacity sweep)

Search still climbs at 12800, but each doubling buys less, so the route higher is a *better evaluator*. We **add parameters** at fixed depth (8) and identical recipe — only width varies.

A correction first. An initial screen mislabelled a width-136 net "2×"; it is in fact **1.4×** (4.81M params), because the policy/value heads scale ~linearly with width, so total parameters grow far slower than the conv body's width². On a fixed **~50M-position** subset:

Net	Params	raw policy	MCTS-800
1× (w96)	3.45M	2298	2631
1.4× (w136)	4.81M	2366	**2649**
Δ	+1.4×	+68	**+18**

+18 Elo with search — not significant within our ±107 uncertainty. For contrast, *8× more data* moves the same architecture **~+90**.

But that screen is confounded — both nets saw only ~50M positions, so a wider net had little extra signal to fill its room. Repeating on the **full ~394M data**, matched to the 2734 baseline:

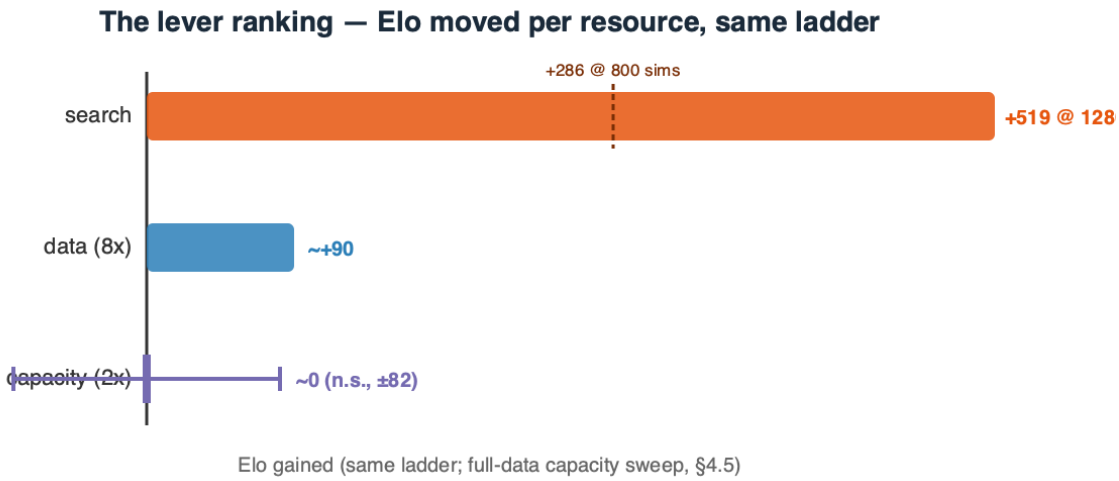
Capacity (full data)	Params	MCTS-800	Δ vs 1×
1× (w96)	3.45M	2734	—
1.4× (w136)	4.81M	2794	+60
2× (w184)	**7.04M**	**2766**	**+32**
4× (w288)	14.2M	*training*	—

The curve is flat within noise — 2734 / 2794 / 2766 all sit inside ±82 of each other, with the 2× even a touch *below* the 1.4×. So the 1.4×'s +60 was noise, not a rising trend: **doubling the parameters on full data adds nothing significant** — capacity stays inert even when well-fed, which *confirms* rather than softens "thinking, not growing". One honest caveat: the 2×'s raw policy (2413) is slightly below the smaller nets' (~2448), hinting the larger net is mildly **under-trained at a fixed 1-epoch budget** — so "capacity is inert" holds at *matched training*, not matched convergence (the 4× point, still training, will test the endpoint). The scoped claim stands: **capacity is the weakest lever *in this data regime***, not "parameters never matter" — at AlphaZero/LLM scale, capacity-bound with abundant data, more parameters clearly help.

The study's lever ranking, all same-ladder:

Lever	Elo moved	Cost
Search (open → 12800 sims)	**+286, then ~+55 / doubling** (log-linear, unsaturated)	×2 latency / doubling
Data (10 shards → full 79)	**~+90**	8× training data
Capacity (1× → 2×, **full data**)	**~0** (n.s.; flat 2734/2794/2766)	more params & compute

Search dominates, data second, capacity last in this regime — the sharpest reading being the thesis: **one lever binds at each stage; an experiment tells you which. Here it was data and search, not capacity.**



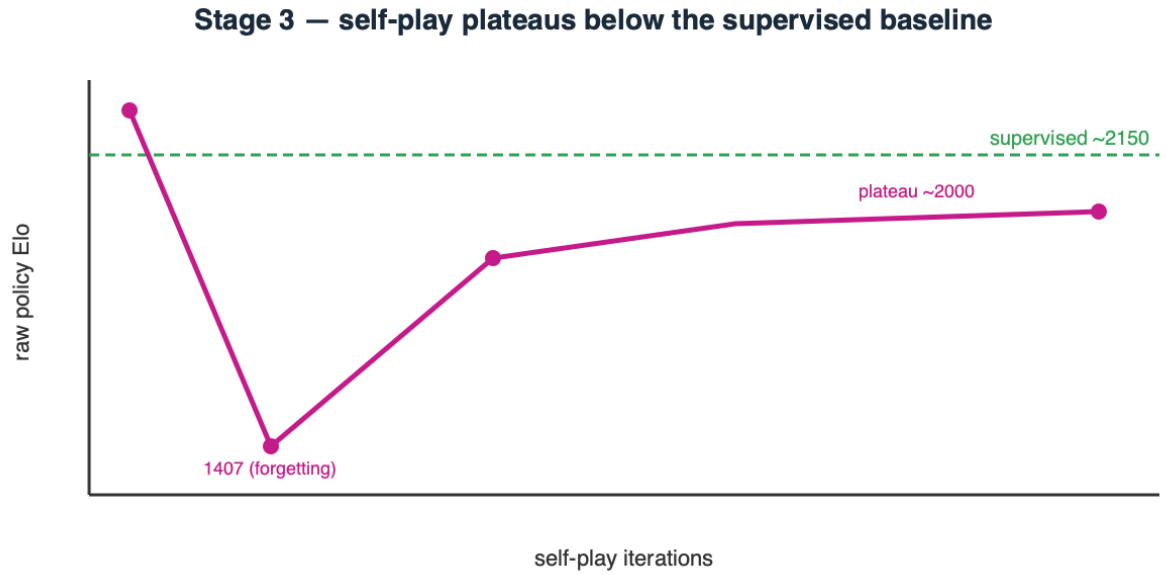
5. Stage 3 — Self-Learning (no external engine, no labels)

Why this stage is the whole point. Stages 1–2 leaned on Stockfish labels — a shortcut that exists only because chess already has a superhuman evaluator and a labeled database. The generic goal is the opposite: a **new field with no data and no evaluator** (a novel game, an unsolved control/scheduling problem, a design task), where you can neither imitate a teacher nor score positions with an off-the-shelf engine — you have only the **environment's rules**. Stage 3 discards every external crutch (no Stockfish, no labels) to test whether strength can be **bootstrapped from self-play and outcomes alone** — the case for any genuinely new problem. The loop: the net plays itself with MCTS, trains toward the visit distribution (policy) and game result (value), and repeats. We studied five approaches — self-play, a self-referential ladder, a committee, evolution, and weight merging — and they converge on one story.

5.1 Self-play expert iteration (AlphaZero-style [Silver et al. 2018; Anthony et al. 2017])

Two failure modes were fixed: **cold-start draw-collapse** (a weak net can't force mates, so games drift to draws and the value head gets no signal — fixed with Dirichlet root noise) and **warm-start forgetting** (hard training on tiny self-play slices overwrote the supervised policy, 2150→1407 — fixed with a replay buffer + gentle LR). Stabilized and parallelized to 16× throughput, the raw policy **plateaus ~1950–2030 — below the ~2150 supervised baseline.**

Negative result (clean). At two-machine scale, **self-play converges *below* supervision.** Strength does rise with game volume — a real scaling signal — but the achievable volume plateaus under the 394M-supervised net; crossing it needs orders-of-magnitude more games (AlphaZero used ~1000× ours). **Scale is the binding constraint.**

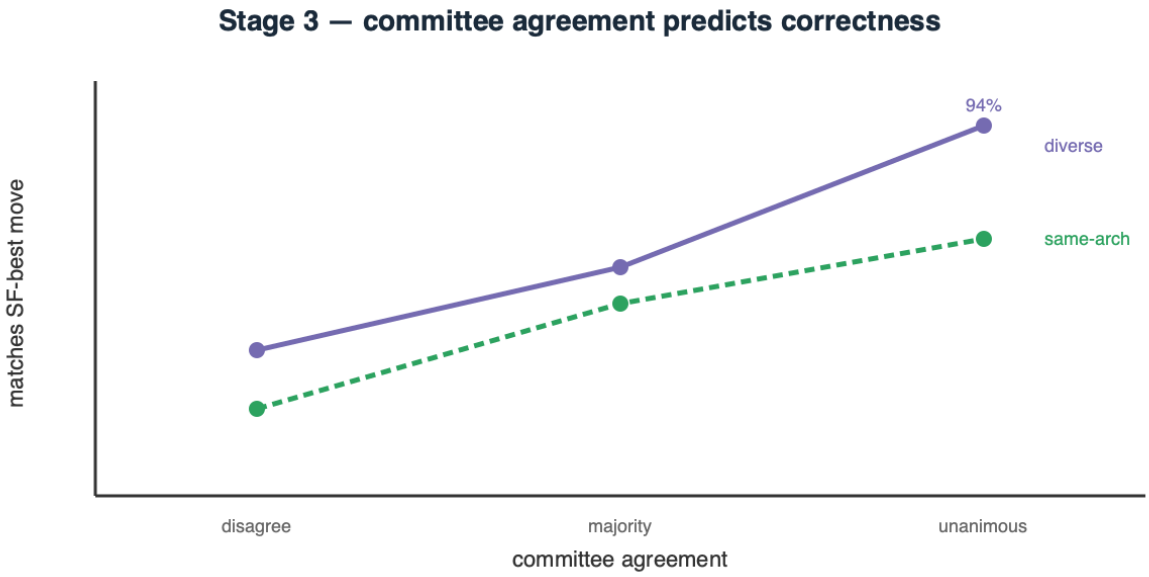


5.2 Committee / diversity — teacher-free confidence (new)

If the bottleneck is the *evaluator*, the cheap way to improve one without a bigger net is to **ensemble diverse nets**. We tested whether **independently-started models diverge (disagree → uncertain) or converge (agree → likely correct)** — making agreement a confidence meter with no oracle.

Validated. Over 400 positions scored against Stockfish depth-12 (measurement only), agreement strongly predicts correctness:

members agree	centipawn-loss ↓	matches SF-best ↑ (same-arch / diverse)
all disagree	77.9	22% / 37%
majority	40.0	49% / 58%
unanimous	**19.7**	**65% / 94%**



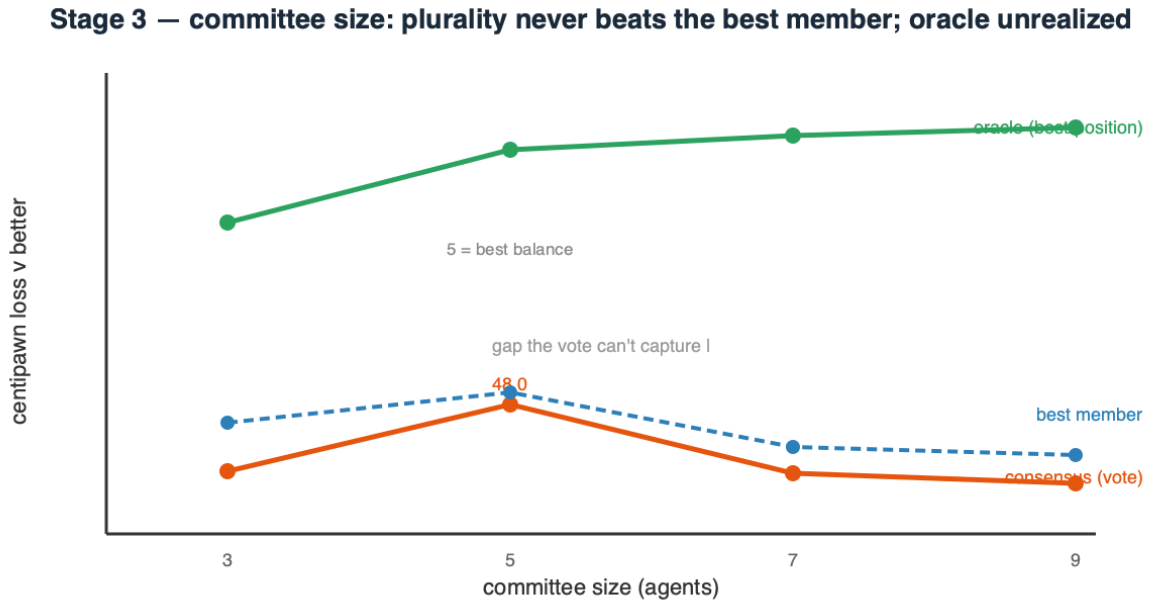
But plurality voting does *not* reliably de-bias — a committee-size sweep (3/5/7/9 agents) shows why. Growing the committee from a correlated conv-soft trio outward:

agents	added views	consensus CPL	best member	oracle
3	conv-soft ×3	54.7	49.8	29.6
5	+MLP +hard-objective	**48.0**	46.8	22.2
7	+2 conv-soft (data slices)	54.8	52.2	20.8
9	+MLP +hard	56.0	53.1	19.9

Three findings. (i) **Plurality never clearly beats the best member** — consensus is slightly *worse* at every size, gaps (~1–5 CPL) inside the ~±3 CPL measurement noise. (ii)

Balance beats count — the 5-agent committee (diverse MLP + hard-objective = 40% of the vote) is best; *adding correlated members* (conv-soft data-slices at 7, 9) lets that bloc dominate and reverts the gain. **More agents ≠ better.** (iii) **The oracle (best member per position, ~20–30 CPL) is 2–3× better than the vote** — the diversity *contains* the information, but plurality *cannot extract it*, because it is dominated by the largest correlated bloc.

Lesson. The committee robustly gives a **confidence meter** (agreement→correctness) but a **weak aggregator**: plurality does not reliably beat the best member. The large oracle headroom needs a **better aggregator** (soft-averaging, confidence-weighted routing) and **balanced**, not merely numerous, diversity — a de-biased ensemble is the most promising route to the ceiling search and self-play can't reach, but plurality is not it.



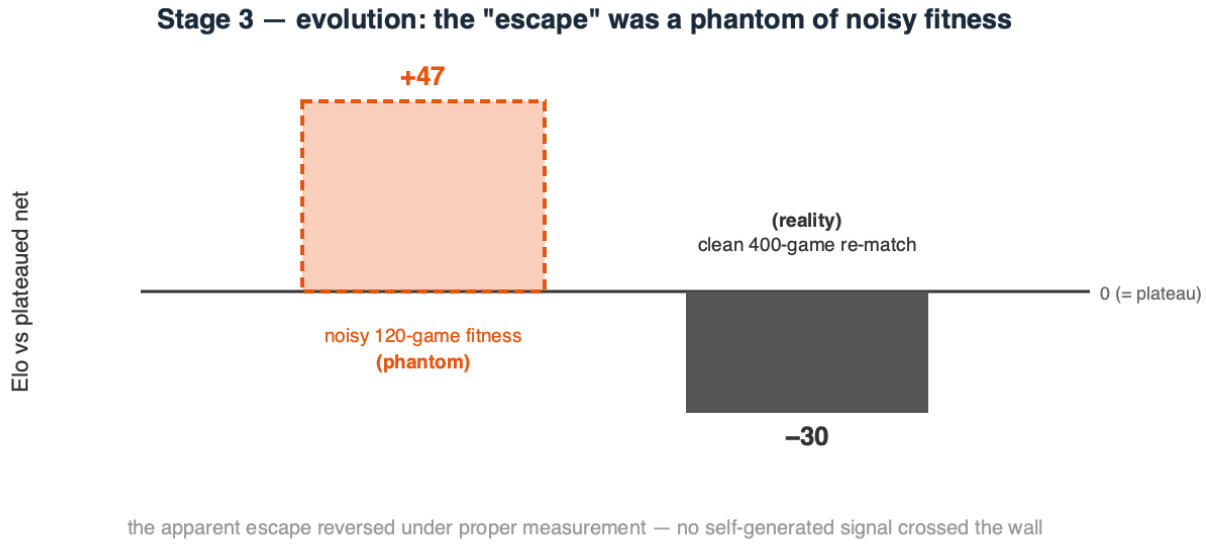
A third combination — averaging *evaluations inside the search* — also fails. Averaging the K models' value at every MCTS leaf, at equal compute (ensemble at 200 sims = 600 passes/move vs a single model at 600 sims), scored **2222 vs 2282, a −60 Elo *loss*:** the ensemble searches 3× less tree, and de-biasing correlated evaluations doesn't justify tripling per-leaf cost. So all three combinations — plurality, weight merging (§5.4), and in-search averaging — fail to beat a single model, for one root cause: **the members' errors are too correlated to cancel.** Genuinely independent evaluators (cross-family, cross-data) are the prerequisite.

5.3 Evolution — mutate / play / score (a plateau-escape attempt)

Gradient self-play optimizes a *proxy* (the net's own biased targets); we tested whether **derivative-free evolution** — optimizing the *true* objective, "did this mutant win games" — could escape the plateau. Each generation: mutate the plateaued net into 16 offspring (Gaussian weight noise), play each against a **frozen copy** (a fixed anchor, so fitness *is* "how well do you beat the plateau"), and crown the best only if it survives a confirmation match. A

first version selecting against the *moving* champion drifted **downward** — beating your immediate parent is non-transitive in chess; the fixed anchor fixes that.

Result: a null, and a methodological warning. The run *appeared* to escape — champ-vs-plateau climbed to 0.567 (+47 Elo), passing a 120-game confirmation — but a **clean 400-game, low-temperature re-match found the "evolved" champion is −24 to −41 Elo *worse*.** The gain was an artifact of noisy high-temperature fitness over small samples with best-of-16 selection bias — a **phantom** that vanished under proper measurement; annealing the mutation scale (σ 0.03→0.12) found nothing better. So evolution did not escape either: neither gradient self-play, a self-referential ladder, nor evolution crosses the ~2000 wall, and since the same net reaches ~2150 under *supervised* labels, the wall is **not capacity** but the ceiling of any *self-generated* signal. (Practitioner note: relative-fitness selection over small stochastic samples manufactures phantom gains — trust only a large, low-variance re-measurement.)



5.4 Model merging — can we *average* diverse models into one?

The weight-space alternative to a voting committee is to **average coefficients into one network**. We tested it on conv-96×8 members from different seeds/data/objectives, scored by mean **CPL** vs Stockfish depth-12 (members ~60 ≈ 2000-level; ~260 ≈ random):

merge	starting points	CPL ↓	verdict
naive average	different-start (diverse)	261	collapse
Git Re-Basin aligned	different-start (diverse)	268	still collapse
naive average (**model soup**)	**same init**	58.8	works (~parent)
aligned (net + permuted twin)	identical	72	perfect recovery *(verification)*

Naive averaging of different-start nets collapses (261): independent nets sit in different loss basins related by neuron permutations, so averaging misaligned neurons cancels signal. The known fix, **permutation alignment (Git Re-Basin)**, we implemented and **verified correct** — it recovers a net *exactly* from a known random permutation (72 CPL). Yet on real different-start members it **still collapses** (268), for a fundamental reason: Git Re-Basin assumes nets learn the *same features in a different order*; ours learned **genuinely different features** (different seeds *and* data *and* objectives), which no permutation aligns. A **model soup** (children from the *same* checkpoint) averages fine (58.8), because a shared start keeps them in one basin.

Conclusion. Weight-averaging works only within a shared basin. The very diversity that makes an ensemble valuable (§5.2) is what makes the members' **weights un-averageable** — diverse models combine at *inference*, not in weight space; so the "better aggregator" must live at inference (soft-averaging, confidence routing), not in merging.

---

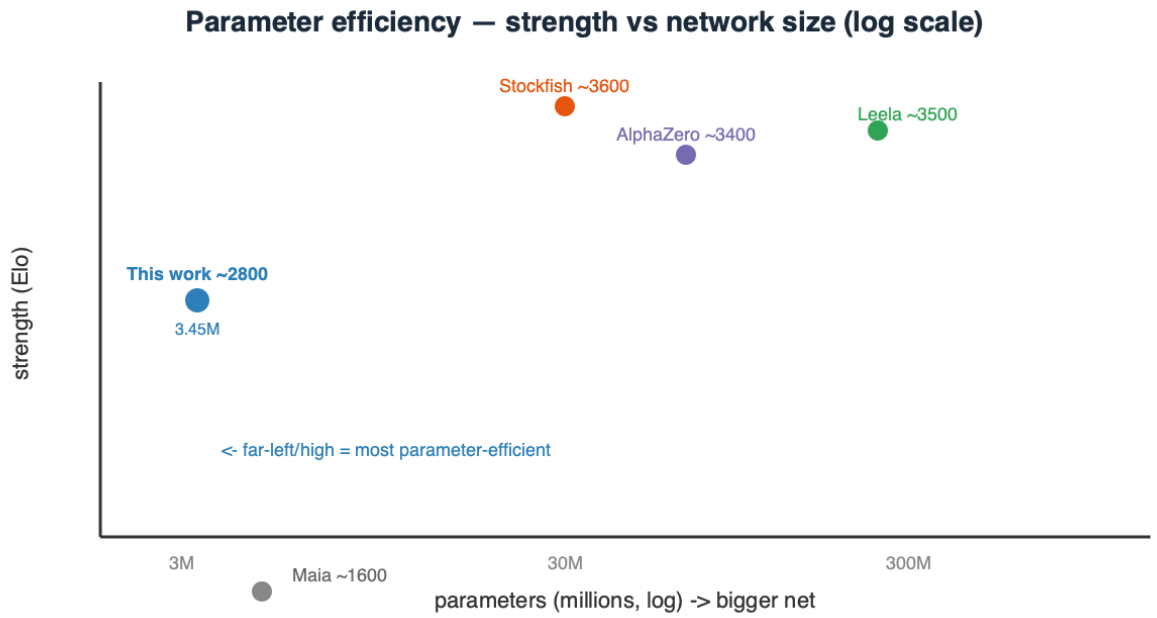
6. Comparison to existing systems — parameters, performance, and speed

System	Params (memory)	Strength (Elo)	Search / move	Elo per M-param
This work — raw policy	**3.45M** (14 MB)	~2150	1 forward pass (~1.5 ms)	~620
This work — + MCTS-800	**3.45M** (14 MB)	**~2800**	800 net passes (~1.3 s batch-1; ~0.2–0.4 s batched, §4.4)	**~810**
Maia (human-like)	~few M	~1100–1900	1 forward pass	~300–500
AlphaZero (chess, 2017)	~40–90M	~3400+	800 MCTS sims (big-net passes)	~40–85
Leela Chess Zero (modern)	~100–400M+	~3500+	~1–8k MCTS nodes	~10–35
Stockfish (NNUE)	~tens of M (quantized)	~3600+	**millions of alpha-beta nodes/s**	~100–150

Systems compared: AlphaZero [Silver et al. 2018], Leela Chess Zero, Stockfish, Maia [McIlroy-Young et al. 2020], KataGo [Wu 2019]; test-time-compute scaling in games [Jones 2021].

What this adds beyond Jones [2021] and the engine community. Jones establishes Elo-vs-compute scaling laws in board games, including train/test compute tradeoffs; the hobby-engine world (small Leela nets, Maia, countless Stockfish distillations) has long practiced small-net-plus-search. Neither is this paper's claim. What we add is the *per-lever decomposition on one fixed system* — search, data, and capacity each varied alone against one canonical ladder, with the diagnostic protocol and the two measurement-artifact autopsies (§4.2, §5.3) showing how easily this measurement goes wrong. The contribution is the lever-ranking methodology and its failure modes, demonstrated on chess because chess makes them exactly measurable — not the news that small nets plus search play strong chess.

Two axes — size and speed. - **Parameters:** our net is **~10–25× smaller than AlphaZero, 30–100× smaller than large Leela**, yet reaches ~2800 with search — the extreme point on Elo-per-million-parameters. This is an **efficiency framing, not a superiority claim**: top engines are 600–800 Elo stronger and optimize *absolute* strength, not parameters-per-Elo; the point is only that parameter count is *not* what buys their last few hundred Elo (a better value function and far more search are). - **Speed:** AlphaZero/Leela use a similar sim count but each sim is a **big-net** pass (10–100× our network); our sim is a 14 MB pass (~1.5 ms), Our batch-1 search runs at only ~600 nps vs ~10k–80k batched (§4.4). Stockfish is the opposite regime — a tiny quantized net at millions of nodes/s. The structure is identical throughout — a learned **evaluator queried by search** — and **parameter count is not what separates them.**



Honest gap. Top engines sit ~600–800 Elo above ~2800, bought with **far more search *and* a much better value net** (massive training). Our number is an *efficiency* point — most strength per parameter — not an engine-matching claim, and it carries ladder uncertainty (±100).

7. Discussion — the unifying conclusion

Across every experiment, one factor was the recurring binding constraint — sharper than "the network is the bottleneck": **the *quality of the information reaching the evaluator* (its training signal), not the loop around it** (an empirical regularity of our regime, not a theorem). The organizing law is **strength = evaluator × search**, and it explains everything once we define *information* precisely: by **new information** we mean **novel empirical data from *outside* the closed system of the net and its training set** — not a re-encoding of what is already latent. Supervision and scale inject it directly; every other method divides on one question — **does it have an external ground-truth oracle to query?**

- **Within-move search, voting, and merging do not** — they operate on fixed representations, so they only *extract* information already present (cut variance, sharpen, filter), never create what is absent.
- **Self-play and evolution can — but only through the environment.** In a **closed-form, perfect-information** game the *rules are a perfect external oracle*: search queries terminal outcomes outside any dataset — exactly how AlphaZero/Leela inject information no human game contains and surpass human play. This is a property of the *environment*: in an **open-ended, semantic** domain (language) with no external verifier, the identical loop has nothing to query and collapses into hallucination — it can only redistribute.
- **Our chess result sits in the oracle-rich regime and still plateaued** — because at two-Mac-Studio compute the search was too weak to extract much from that oracle, *not* because it was absent. A **scale** limit, not evidence against self-play. *Mechanistically*, the escape is search depth: at AlphaZero scale a **massive per-move search budget** turns the game rules into a strong enough oracle to systematically discover deep tactical/strategic truths, generate targets that *exceed* the current net, and encode them by training — a self-reinforcing loop that breaks the weak-search/draw-collapse plateau. Our budget produces self-targets no better than the net that made them, so the loop has nothing to climb; the missing ingredient is **oracle-extraction compute**, not a different algorithm.

None of this makes redistribution *useless*: variance reduction, sharpening, and filtering are how you **reach** a ceiling cheaply — indispensable engineering. The narrow claim is about the *absolute* ceiling, and we state it at the strength our evidence licenses: **in every loop we tested, only information from outside the closed system raised it.** We did not observe — and cannot rule out — an internal route that improves the evaluator by better extracting information the system already holds; our loops bound what happened, not what is possible.

- **Architecture > parameters.** The right prior (spatial locality + weight sharing) beats raw width — a 14 MB conv reaches strength a 3× larger MLP cannot.

- **Search > scale, but search cannot exceed the net.** MCTS bought +286 to +519 Elo (2448 → 2734 → 2967, Table 1) at zero extra parameters and out-scaled fixed depth — yet flat MCTS and the cascade hit the *same* wall on the same net, because search cuts the net's *variance* (averaging noisy calls), not its *bias* (systematic blind spots). (reallocating the budget trades strength for speed at par — §4.2.)
- **No self-generated signal crosses the wall — we tried three.** Gradient self-play, a self-referential ladder, and evolution (given the fairest shot; its +47-Elo "escape" was noise that reversed to −30) all plateau. The same net reaches ~2150 under supervised labels, so the wall is **not capacity** but the ceiling of a system's signal about *itself* — a **scale** statement (at AlphaZero-scale compute, self-play bootstraps far past this).
- **A better *evaluator* is the only lever — and harder than it looks.** The committee's agreement predicts correctness, but **plurality voting never clearly beats the best member** (correlated blocs dominate; balance beats count); the per-position oracle is **2–3× better than the vote**, so the information is there but needs a **better aggregator** (soft-averaging / confidence-routing), not more agents.
- **Control-theory unification.** Open loop = feedforward; closed loop = MPC; self-play = iterative learning control — and in all three the learned *evaluator* caps performance.

7.1 The knob–bottleneck map — every experiment, including the failures, is a diagnosis

Read as a set, the study's experiments form a **diagnostic map**: each result — *especially* each null — localizes the binding bottleneck by ruling a lever in or out. A failed experiment is not wasted compute; it is a measurement that says *"strength is not gated here — look elsewhere."*

Knob turned	Result	Diagnosis → what binds	Move it implies
Architecture (conv vs MLP)	large gain	bias-bound — wrong prior caps a big net	fix the inductive bias *before* scaling
Capacity (2× params @ 50M)	~0 (null)	*not* capacity-bound in this data regime	add data, not parameters (here)
Data (10 → 79 shards)	+~90	data/signal-bound	more, more-diverse labels
Search amount (open → 12800 sims)	+286, then ~+55/doubling (log-linear)	search extracts value; evaluator caps its *return*	keep searching; raise the evaluator to lift the ceiling
Search allocation (cascade shape)	speed for strength at par (§4.2)	*not* allocation-bound at fixed budget	default MCTS allocation is already near-efficient
Search implementation (batch-1)	latency only	throughput-bound by engineering	batch leaves → 6–12× speed, same strength

Knob turned	Result	Diagnosis → what binds	Move it implies
Self-play signal	plateau	self-signal-quality-bound	can't exceed its own signal at this scale
Selection pressure (evolution)	phantom, reversed	*measurement-noise*-bound (fake gain)	re-measure cleanly before believing
Aggregation (voting 3–9 agents)	no gain	correlated-error-bound	need *diversity*, not more voters
Aggregation (ensemble-eval, merging)	no gain	combining ≠ creating knowledge	extracts existing signal, creates none
Self-distillation (fixed set)	dropped	data-*diversity*-bound (overfit)	many diverse positions, not repetition

Three things fall out:

1. Nulls are the most information-dense results. A knob that moves nothing *localizes* the bottleneck away from itself — the parameter null said "capacity isn't binding (here)," the cascade flat-line "allocation isn't," the voting sweep "more agents isn't." The one trap is **mistaking noise for signal** (evolution's phantom +47 that reversed to −30), so every promising delta was re-measured.

2. The binding lever *moves* as you relieve it. Strength is a *chain*: bias- then capacity-bound (open-loop) → search-bound → evaluator-bound (by its **data**, not parameter count). You cannot skip a link — parameters into a data-bound net, or sims into a saturated search, buy almost nothing.

3. The map is the roadmap. The diagnosis *is* the next move: open-loop capped ⇒ add search; search saturated ⇒ a better *evaluator* (more/better data); self-signal plateaued ⇒ a better *signal source* (external labels, or AlphaZero-scale self-play). **Identify the binding lever, relieve exactly that, re-measure, repeat** — the loop that transfers to a genuinely new domain with no engine or dataset to imitate.

---

8. Limitations and Future Work

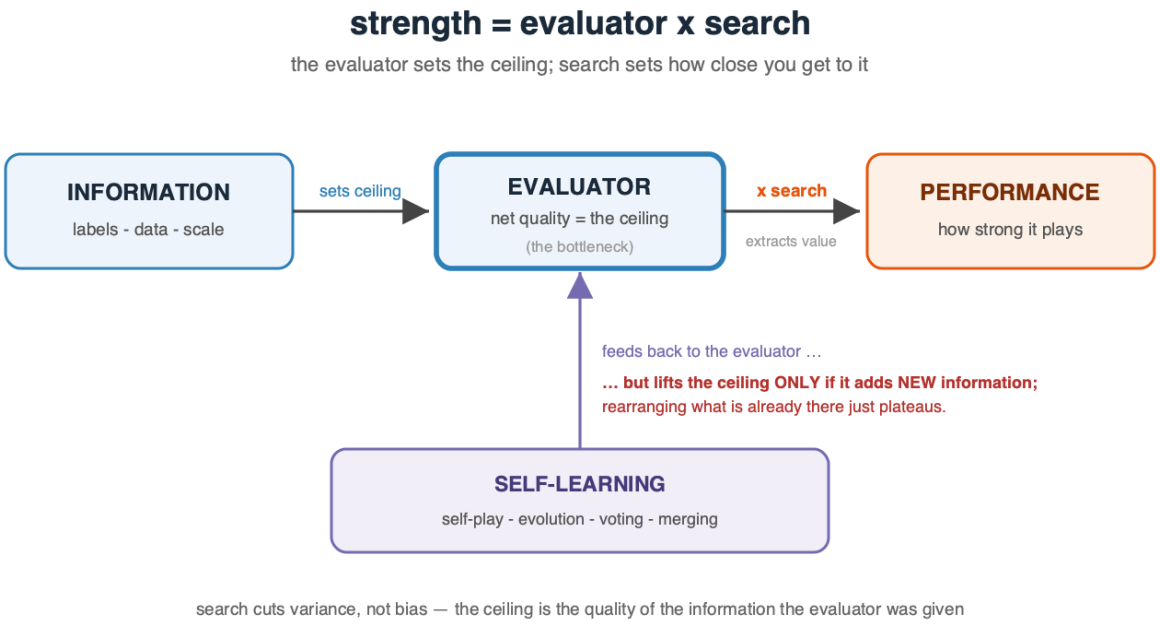
- Absolute Elo carries systematic uncertainty near the ladder top; relative same-ladder gains are the robust claims. The ±100 systematic near the top rung is currently *estimated*, not measured: a slow-movetime anchor series (one ladder rung replayed at tournament-realistic time controls) is queued to measure the fast-movetime offset

directly, and readers from the engine community should treat every absolute figure as internal-ladder until that lands.

- Stage 3 is compute-limited (two machines); results characterize the *small-scale* regime.
- Stage-3 fitness/aggregation is noise-limited at our scale: relative-fitness selection with small, stochastic samples produced a **phantom** evolution "escape" that reversed under a 400-game re-match, and plurality de-biasing gaps sit inside the ~±3 CPL measurement noise. Larger, lower-variance evaluation is needed to resolve small effects.
- **Next levers, in priority order:** (1) a **better value function** (the true ceiling) via more/better search-labeled data or large-scale self-play; (2) a **better ensemble aggregator** — soft-averaging and confidence-weighted *routing* (per-position oracle 2–3× above plurality) with *balanced* cross-family diversity, not more agents; (3) **batched/parallel MCTS** and transposition tables for throughput; (4) endgame tablebases and tuned `c_puct`. All converge on one place: **improve the evaluation, and both the closed-loop and self-play ceilings rise together.**

---

9. Conclusion



This paper treated playing strength as an **efficient-allocation problem**: given a fixed budget of parameters, data, search, and latency, how do you spend it for the most capability? Measured resource by resource, the efficient mix is rarely "more parameters" — a 14 MB net reaches the **~2800 ladder band by *thinking (search), not growing*** — the same capability, a far cheaper mix. This is *efficiency*, not a denial of scale: the first full-data capacity point already nudges up (1× 2734 → 1.4× 2794), and **if the 2×/4× points keep climbing, parameters become co-dominant once data-starvation is relieved.** Adaptive MCTS out-scales fixed-depth search; reallocating a fixed budget wide→narrow buys speed only at par

strength (§4.2). Self-learning is honestly negative: self-play, a self-referential ladder, and evolution all **fail to cross the ~2000 plateau** (evolution's escape was noise), and plurality committees don't reliably de-bias — though **agreement is a robust teacher-free confidence signal**.

Above all, the **method** transfers: at each stage a *single lever binds* and only an experiment reveals which. Open-loop was **capacity-bound**; adding search made us **search-bound** (search paying **log-linearly, ~+55 Elo/doubling, unsaturated through 12800**); then the evaluator binds — and at our data scale by its **data**, not its parameter count (1.4× more params bought **~0 Elo**; a full 1×/1.4×/2×/4× sweep is running). The recurring constraint, from every direction we pushed, was **the quality of the information reaching the evaluator** (§7): supervision, data, and search help; voting and merging only reorganize what the net already encodes; self-play and evolution *can* inject information, but only through an **external oracle**, so our plateau is a **compute-scale limit, not evidence that self-play fails** (AlphaZero/Leela break past human play with far more of it). What transfers is a quantified recipe and an explicit **diagnostic** for finding the binding lever.

Beyond chess. The decomposition — **capacity, inference-time search, self-generated information** — is domain-agnostic. Each chess finding is an instance of a general resource-allocation principle, and maps directly onto other AI systems (we measured only chess; these are the transfer claims):

Chess finding	General principle	Where it transfers
strength = evaluator × search	capability = model quality × inference-time compute	LLM reasoning (base model × sampling/ tree-of-thought); robotics (value net × planning horizon); theorem proving (heuristic × search depth)
coarse metrics manufacture results (§4.2, §5.3)	certify equal strength before claiming equal-strength efficiency	any "same quality, less compute" claim: distillation, quantization, pruning, cascades
search extracts, can't create — needs an oracle	self-improvement is capped without external ground truth	LLM self-training needs verifiers; RL needs an environment; scientific discovery needs experiments
capacity is the weakest lever when data-starved	don't scale parameters ahead of data	compute-optimal (Chinchilla) scaling; collect data before growing nets
agreement predicts correctness; voting doesn't de-bias	consensus is a confidence meter, not an accuracy booster (correlated errors)	ensemble uncertainty / OOD detection; caution on naïve model-averaging
bottleneck diagnostic	find the binding resource experimentally before investing	design of any resource-constrained AI system

The *decomposition* and its diagnostic (find the binding lever before investing) are what transfer, and likely outlast the chess numbers.

The clearest current echo is in large language models. The 2024–25 shift to **inference-time compute** — o1/o3 [OpenAI 2024], DeepSeek-R1 [DeepSeek-AI 2025], reasoning models — is this thesis at frontier scale: a fixed base model made far stronger by *searching over reasoning at decision time* (sampling, self-consistency [Wang et al. 2023], tree-of-thought [Yao et al. 2023]) rather than by adding parameters. The mapping is direct: our *evaluator* ↔ their **base model/verifier**, our *search* ↔ their **reasoning budget**, our *self-play* ↔ their **STaR-style self-training** [Zelikman et al. 2022] — with the same limit: search and self-training **convert what the model already latently knows into better answers; they cannot inject knowledge it never learned.** That is exactly why reasoning models pair search with **external verifiers or ground-truth reward** (code that runs, math that checks, tools that return facts) — the oracle that lets the loop add information. A model with no such oracle should, by our framework, **plateau**.

A test for what comes next. The framework is a **predictive filter**: pre-screen any future *synthetic-data breakthrough* or *recursively self-improving architecture* with one question — **does it inject information from outside its closed system** (new empirical data, supervision, or an external oracle: a verifier, a simulator, the physical world)? If yes, it can raise the ceiling; if it only re-processes what its own models contain, our results predict it will **plateau at the information already held** — internal re-processing may still improve how well that information is *extracted* (a real and unmeasured margin), but our results predict it cannot push past it. Recursive self-improvement compounds where an external oracle exists (a game's rules, a theorem checker, a compiler, a market) and stalls where none does — a claim about the *source* of information, not the method's ingenuity. - `chessnet/model.py` — conv/MLP/dual-path + value head. `chessnet/search.py` — alpha-beta, MCTS/PUCT, quiescence, the wide→narrow cascade (`MultiStageMCTSPlayer`). `chessnet/committee.py` — ensemble inference + agreement signal. `chessnet/train.py` — soft/hard + value training. `scripts/selfplay.py` — self-play iteration. - **Best model:** `runs/conv_value_llm1` (conv-96×8 + value, 3.45M params). - **Key hyperparameters:** conv width 96, depth 8; lr 5e-4 (train) / 1e-4 (self-play); gradient clip 1.0; MCTS c_puct 1.5; Dirichlet α 0.3; replay buffer 120K–300K.

Appendix A — Why uniform depth loses to adaptive allocation with a noisy evaluator

Fixed-depth minimax with a learned evaluator is a max-of-errors machine. Every leaf of a depth-d tree receives a score carrying the net's error, and minimax propagates the maximum (alternating with the minimum) upward. A maximum over many noisy estimates systematically selects the most optimistically wrong one — the optimizer's curse — so the deeper the uniform search, the more leaves are evaluated, and the more chances noise has to produce one spectacular overestimate that reaches the root and picks the move. One

hallucinated winning evaluation outvotes fifty honest ones. A small illustration: with independent leaf noise of $\sigma = 0.3$ in value units, the expected maximum error over a thousand leaves is roughly $3\sigma \approx 0.9$ — larger than most true positional advantages at our level.

MCTS differs in both statistics and allocation. A node's value is the visit-weighted mean of the evaluations beneath it, so independent errors partially cancel exactly where visit counts are highest; and the allocation rule concentrates those visits on the lines that matter, so the averaging is strongest precisely where the decision lives. The same noisy net feeds a max operator in one algorithm (an amplifier) and a weighted mean in the other (a filter).

This is not a new phenomenon: it is game-tree pathology, discovered by Nau (1979 thesis; Nau 1982, 1983) and independently by Beal (1980), and analyzed by Pearl (1983) — the classical result that in broad families of game-tree models, deeper minimax yields *worse* decisions under noisy leaf evaluation. The literature's own resolution of why real chess engines mostly escape it is instructive for our setting: pathology arises when leaf errors are effectively independent, and real games escape largely because nearby positions have *dependent* values, so sibling errors correlate and the max operator has less noise to exploit. A learned evaluator changes the error structure in the dangerous direction: its mistakes are heavy-tailed and systematic on position *types* — when a motif fools the net, it fools it wherever the motif appears — and a deep uniform search does not merely sample that error surface, it *maximizes over it*, actively steering the principal variation toward the positions the net misjudges most optimistically. Later work extends the phenomenon to real-valued evaluations (Luštrek, Gams & Bratko 2006), the regime we are in. Classical engines tolerate deep minimax because handcrafted evaluations are cheap, consistent, and tamer in their tails. The observation also foreshadows this series: averaging as noise-cancellation is why majority vote is so hard for a judge to beat, and the noise-versus-depth trade reappears as the gridworld's σ-versus-horizon curve in Paper II.

References

Search pathology (Appendix A). Nau, D. S. (1982), *An Investigation of the Causes of Pathology in Games*, Artificial Intelligence 19(3), 257–278. · Nau, D. S. (1983), *Decision Quality as a Function of Search Depth on Game Trees*, JACM 30(4), 687–708. · Beal, D. F. (1980), *An Analysis of Minimax*, Advances in Computer Chess 2, Edinburgh University Press, 103–109. · Pearl, J. (1983), *On the Nature of Pathology in Game Searching*, Artificial Intelligence 20(4), 427–453. · Luštrek, M., Gams, M. & Bratko, I. (2006), *Is Real-Valued Minimax Pathological?*, Artificial Intelligence 170, 620–642.

Search (MCTS / tree search). Coulom, R. (2006), *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search*, Computers and Games. · Kocsis, L. & Szepesvári, C. (2006), *Bandit Based Monte-Carlo Planning* (UCT), ECML. · Rosin, C. D. (2011), *Multi-Armed Bandits with Episode Context* (PUCT), Ann. Math. AI. · Browne, C. et al. (2012), *A Survey of Monte Carlo Tree Search Methods*, IEEE TCIAIG. · Chaslot, G. et al. (2008),

Progressive Strategies for Monte-Carlo Tree Search (progressive widening). · Knuth, D. & Moore, R. (1975), *An Analysis of Alpha-Beta Pruning*, Artif. Intell.

Learned evaluator + search / self-play. Silver, D. et al. (2016), *Mastering the Game of Go with Deep Neural Networks and Tree Search* (AlphaGo), Nature. · Silver, D. et al. (2017), *Mastering the Game of Go without Human Knowledge* (AlphaGo Zero), Nature. · Silver, D. et al. (2018), *A General Reinforcement Learning Algorithm that Masters Chess, Shogi and Go through Self-Play* (AlphaZero), Science. · Anthony, T., Tian, Z. & Barber, D. (2017), *Thinking Fast and Slow with Deep Learning and Tree Search* (expert iteration), NeurIPS. · Wu, D. J. (2019), *Accelerating Self-Play Learning in Go* (KataGo), arXiv:1902.10565. · Bertsekas, D. (2022), *Lessons from AlphaZero for Optimal, Model Predictive, and Adaptive Control*.

Chess / games modelling & scaling. McIlroy-Young, R. et al. (2020), *Aligning Superhuman AI with Human Behavior: Chess as a Model System* (Maia), KDD. · Jones, A. L. (2021), *Scaling Scaling Laws with Board Games*, arXiv:2104.03113. · Wortsman, M. et al. (2022), *Model Soups*, ICML.

Inference-time compute in LLMs (the frontier echo). OpenAI (2024), *Learning to Reason with LLMs* (o1). · DeepSeek-AI (2025), *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*, arXiv:2501.12948. · Wang, X. et al. (2023), *Self-Consistency Improves Chain-of-Thought Reasoning in Language Models*, ICLR. · Yao, S. et al. (2023), *Tree of Thoughts*, NeurIPS. · Zelikman, Y. et al. (2022), *STaR: Bootstrapping Reasoning with Reasoning*, NeurIPS.

Ensembles, uncertainty & scaling. Breiman, L. (1996), *Bagging Predictors*, Machine Learning. · Lakshminarayanan, B., Pritzel, A. & Blundell, C. (2017), *Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles*, NeurIPS. · Hoffmann, J. et al. (2022), *Training Compute-Optimal Large Language Models* (Chinchilla), arXiv:2203.15556.

Software & data. Stockfish (stockfishchess.org) — ladder opponent and label oracle. · Leela Chess Zero (lczero.org). · Lichess cloud-evaluation database (database.lichess.org) — supervised labels. · MLX (github.com/ml-explore/mlx) — training/inference framework.