Alphamon: Bit-Exact GPU Self-Play, a Training-Free Value Bound, and Search-Dominated Play in a Stochastic, Imperfect-Information Trading Card Game

Jovonni L. Pharr

Georgia Cyber Warfare Range / Blackmind

info@gacwr.org

September 10, 2026

v0.0.1

Abstract

We present **Alphamon**, a complete AlphaGo-style agent for the Pokémon Trading Card Game (TCG), and three results from building it that we believe transfer well beyond this game: one on systems, one on analysis, and one on evaluation. **The systems result.** We port a 1,267-card, commercial-scale TCG engine — a ∼ 130-opcode card-effect bytecode VM together with its turn driver, legal-action generator, and neural featurizer — to CUDA at *one full-fidelity game per thread*, and certify it *bit-exact* against the reference C++ engine over 70,000 games and 7,888,623 agent decisions with zero divergences, comparing a canonical hash of the complete state (including all hidden zones, RNG-visible bookkeeping, and mid-effect transients) at every step. The device engine sustains 31,951 games/s (5.05M agent-steps/s) in a batched lockstep wave loop and 89,000 games/s in a fused whole-game kernel, against 3,208 games/s for the reference engine on one CPU core. We give the certification methodology — a parity oracle, full-state hashing, loud-by-construction device faults, and a *de-circularized* gate that we adopted after catching ourselves with a rigged one — that makes such a port trustworthy rather than merely fast. **The analysis result.** Using that same engine, we give a cheap, training-free estimator of whether the AlphaZero recipe can work on a given game at all. Replaying a fixed position under fresh randomness flips the winner with probability F, and we prove $\frac{1}{2}\mathbb{E}[F] \le \mathrm{Err}^\star \le \mathbb{E}[F]$, where $\mathrm{Err}^\star$ is the Bayes sign-error of *any* value function. Measured here, $F = 0.207$, bracketing the Bayes value-accuracy ceiling to $[0.79,\ 0.90]$ — an afternoon of engine replays that predicts the ceiling it took us weeks of scaling to discover. Against that ceiling we show, with same-data controls, that the three classical epistemic levers are jointly spent: 48× more self-play data buys +0.048 and then overfits, doubling capacity buys +0.0004, and a *perfect-information oracle* that sees every hidden card buys +0.003. We further show the AlphaZero improvement operator is *structurally* inert here, not merely noise-limited: wiring a stronger value net into Gumbel reanalyze changes 0.00% of policy targets, because 91% of reanalyzed states have every action already visited and the value-completion term multiplies a zero mask. **The evaluation result.** Finally, on a live TrueSkill-style public ladder, no offline metric we could construct predicts rank: our most rigorous harness — a clone-invariant Nash gauntlet over seven board-anchored champions at their own ship configurations — attains rank correlation $\rho = 0.112$ ($p_{\mathrm{perm}} = 0.814$) and $R^2 = 0.018$ against the ladder, while resubmissions of *unchanged* bundles settle as much as 149 rating points apart (median 88 over five pairwise comparisons). We quantify this, and we retract one of our own conclusions that transient ratings supported and convergence overturned. Throughout, *search* — not any learned component we trained — is the dominant source of strength: holding weights fixed, replacing arg max play with PIMC-PUCT moves win-rate against the engine's reference agent from 0.425 to 0.912. We argue this asymmetry is exactly what the aleatoric ceiling predicts, and that it is the general signature of games in the *future-chance* rather than the hidden-information regime.

1 Introduction

AlphaGo and its successors established a now-canonical recipe: train a policy and a value network, improve them with Monte-Carlo Tree Search (MCTS), and iterate through self-play [1, 2, 3]. The recipe is spectacularly effective on Go, chess, and shogi. Those games share two properties the recipe quietly depends on: they are *deterministic* and *perfect-information*. "Who wins from state s" is therefore a learnable function of s that can be driven toward $\{0, 1\}$, and the flywheel sharpens without bound because a better value yields a better policy yields better targets.

The Pokémon TCG breaks both assumptions. Hands are hidden, decks are shuffled, and many mechanics resolve on coin flips. By the taxonomy of [7] and [8] this places it in the poker family rather than the Go family — but we will show that classification is itself misleading, and that the operative variance here is *future chance* (backgammon) rather than

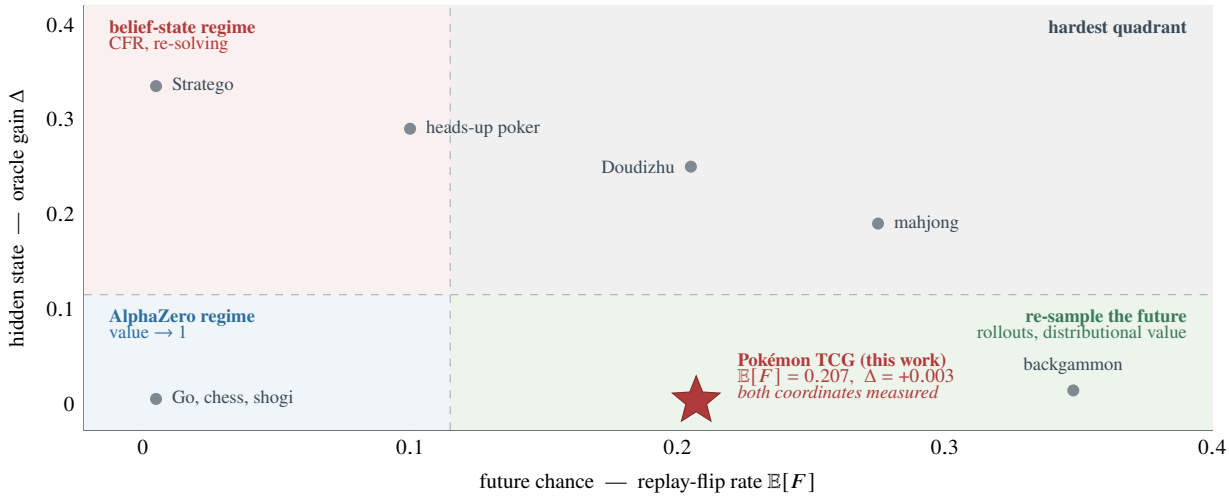


Figure 1. Two measurable coordinates decide which literature applies to a game. The horizontal axis is the *replay-flip rate* $\mathbb{E}[F]$ — how often re-running a fixed position under fresh randomness changes the winner — which bounds the Bayes ceiling on any value function (Prop. 3). The vertical axis is the *oracle gain* Δ: how much value accuracy improves when all hidden state is revealed, measured with a same-data control. Both of our coordinates are measured ($\mathbb{E}[F] = 0.207$, $\Delta = +0.003$); the other games are placed illustratively from their defining properties and the cited results. The Pokémon TCG is routinely classified as poker-family because it is imperfect-information, which would place it in the upper left. Measurement puts it in the lower right, where the applicable machinery is backgammon's rather than poker's — and where compute is better spent re-sampling the future than fitting its expectation (§7).

concealed present state (poker). The distinction is not academic: it determines which half of the imperfect-information literature is applicable, and we measure it directly.

This paper reports what a faithful AlphaGo-style stack does in that regime, and — more usefully — how to find out in advance. Our position is that the interesting object is not a leaderboard number but a *decision procedure*: given a new stochastic game, what should you measure, in what order, before committing months of compute to a value network? We give that procedure, we give the systems substrate that makes running it cheap, and we give the measurements for one game.

Contributions.

1. **A bit-exact GPU-resident TCG engine** (§3). We port a full commercial-scale card-game engine to CUDA — effect VM, turn driver, legal-action generation, and the neural featurizer — one game per thread, and certify it against the reference implementation on 70,000 games / 7.89M decisions with zero divergences. We report the certification methodology, the coverage burn-down that took usable device self-play from 87% to 99.9987%, and the throughput ledger. To our knowledge this is the first bit-exact GPU port of a card-game engine of this complexity.

2. **An Amdahl analysis of the AlphaZero loop** (§3.6) that localizes the residual bottleneck precisely: with the engine and the featurizer on-device, the remaining CPU-bound component is the *tree* itself (4% GPU utilization; raising concurrency 4× moves throughput 87 → 89 records/s). We give the tree-free reformulation — Gumbel sequential halving as a fixed-shape tensor program over certified kernels — that removes it.

3. **The replay-flip estimator of the aleatoric ceiling** (§5, Prop. 3), a training-free, two-sided bound on the Bayes sign-accuracy of any value function, computable from engine replays alone; plus the same-data exhaustion study of data, capacity, and information, and the sample-complexity argument explaining why high aleatoric noise both sets a floor *and* inflates the cost of approaching it.

4. **A structural inertness result for the improvement operator** (§5.6): value-completion in Gumbel reanalyze is a no-op on 100% of our targets for a combinatorial reason (full-visit saturation), independent of value quality.

5. **An evaluation-methodology study** (§7) of agent selection under a noisy live rating: $R^2 = 0.018$ between every offline metric we built and the ladder, a replication study of the ladder itself (149-point spread on unchanged bundles),

and a worked retraction.

Evidentiary standard. A claim appears here only with its receipt — a log, a paired-seed win-rate with an interval, a parity certificate, or a settled leaderboard score. Negative results are reported as first-class, and where a later measurement overturned an earlier conclusion of ours we say so explicitly and in place (§7.4).

2 Setting

The Pokémon TCG AI Battle Challenge provides a Python-callable C++ battle engine, a 1,267-card database, and a native determinized-search API (`SearchBegin/Step/End`). Agents submit a fixed 60-card deck and a policy; the organizers run a continuous ladder in which each submission plays ~ 48 games/day against skill-adjacent opponents (10% uniformly random), and rank agents by a TrueSkill-like Gaussian rating with $\mu_0 = 600$.

Two properties of the game shape everything that follows. First, the branching factor is small: a mean of 7.8 legal actions, $p_{95} = 19$, maximum 32. Search is therefore cheap and well-suited, which is what makes the search results of §7 so large. Second, the outcome is not determined by the state: coin-flip attack effects, shuffled deck order, and undrawn cards mean two continuations of the *same* position under fresh randomness reach different winners a fifth of the time (§5).

Two properties of the *competition* shape our methodology. The ladder rating is a live, non-stationary estimate: identical agents resubmitted by other teams were community-measured 150–400 points apart, and we reproduce the effect internally at 26–149 points over four resubmissions of unchanged bundles (§7.5). And only the two most recent active submissions count at the deadline, which turns submission scheduling into a real component of the final score — a point we return to in §7.5.

3 A GPU-Resident, Bit-Exact Card-Game Engine

AlphaZero's data appetite is measured in millions of games. The reference engine plays 3,208 games/s on a single core, but the *net-guided* self-play loop that AlphaZero actually needs ran at 12–15 games/s, because every node requires a round trip out of C++ into Python featurization and a batch-1 network evaluation. Closing a 200× gap by adding cores is not available on a two-GPU workstation. We closed it by moving the game itself onto the GPU.

3.1 Why the port is tractable: the engine is a bytecode VM

The decisive observation is that a modern card-game engine is not 1,267 hand-written card routines. It is a *data-oriented virtual machine*:

- every card effect is a *program* — an array of `Op{kind, p0..p4}` records in a generated constant table;
- the interpreter is a `switch` over ~ 130 opcodes (`OP_DAMAGE`, `OP_FLIP`, `OP_CHOOSE`, `OP_MOVE_ENERGY`, …);
- the game state is plain-old-data of fixed size — flat integers and inline small-vectors, no pointers and no heap allocation in the hot path — hence trivially copyable into a CUDA `__device__` struct.

The port is therefore bounded: an opcode interpreter plus a turn driver, over constant tables that transfer verbatim. It is not an open-ended per-card rewrite. We believe this observation generalizes — most contemporary TCG and board-game engines are compiled to effect tables for maintainability reasons, and are GPU-portable for the same reason.

3.2 Device representation

One CUDA thread hosts one complete game. The device state is STATE_W = 7,498 `int32` words (≈ 30 KB), covering both players' full hidden zones (deck order with known-masks, hands, prizes with face-up bookkeeping, discards) and every transient the reference engine carries: pending-decision descriptors *including option order*, the effect-stack frames, the post-program queue, coin and count registers, checkup state, and deferred post-attack resolution. Serialization round-trips bit-exactly, which is what allows the device state itself — not a reconstruction of it — to be the input to the parity hash (§3.3).

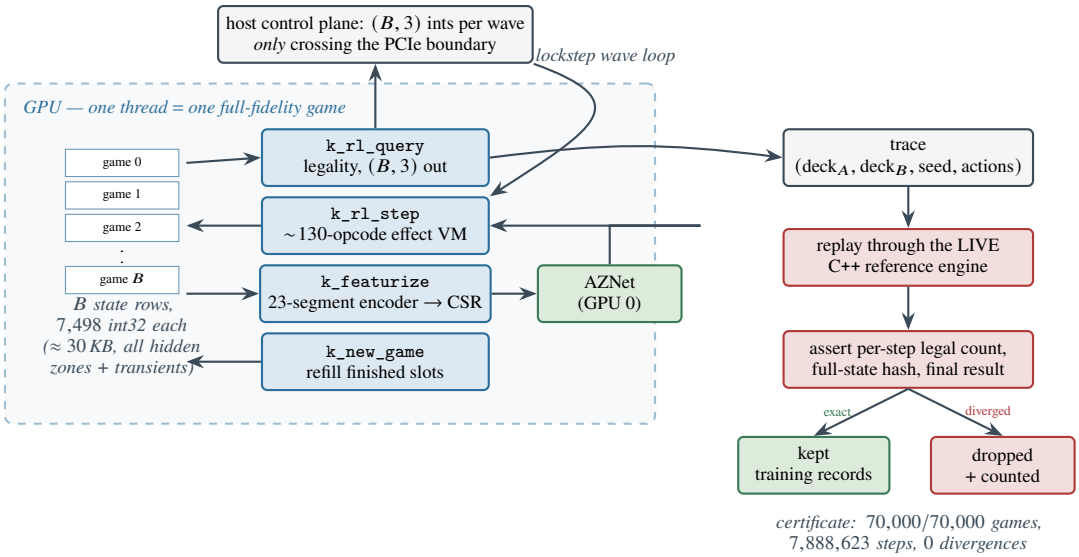


Figure 2. The GPU-resident self-play substrate and its parity gate. B games live on the device as fixed-width state rows and advance in lockstep waves: one batched legality query, one batched step, finished slots harvested and refilled in place so the batch never drains. Only a $(B, 3)$ integer control plane crosses the host boundary per wave. Because the external per-game seed schedule matches the reference engine's, a recorded $(\text{deck}_A, \text{deck}_B, \text{seed}, \text{actions})$ trace replays bit-exactly through the live C++ engine — which is simultaneously the training-data path and a continuous audit: any divergence is caught at its first step, and the game is dropped rather than featurized.

Fixed-capacity buffers replace the host's dynamic containers; where the true bound was mis-estimated the gate found it rather than corrupting a run. The energy-attachment cap was set at 24 and a stacked 25th attachment at a specific seed and step raised a loud device fault; we raised the cap to 40 (STATE_W 6,634 → 7,498) and re-verified the serializer. Two structural caps remain (legal-action and option-array width) and are handled as loud discards.

3.3 The certification methodology

A fast engine that is subtly wrong is worse than no engine, because the error enters the training data silently. We therefore built the oracle before the port, and gated every stage on it.

Parity oracle first. Deterministic games through the reference engine produce golden traces of (seed, action, canonical full-state hash) including hidden state and RNG-visible bookkeeping. The candidate must match bit-for-bit.

Full-state hashing, every step, no sampling. At *every* agent decision of every game, both engines receive the identical (seed, action) sequence — with actions chosen by the reference engine's own strong rules policy, so the comparison rides real play lines rather than random-legal ones — and we compare (i) the legal-move count, (ii) a SHA-1 over a canonical serialization of the complete state summary, and (iii) the final result.

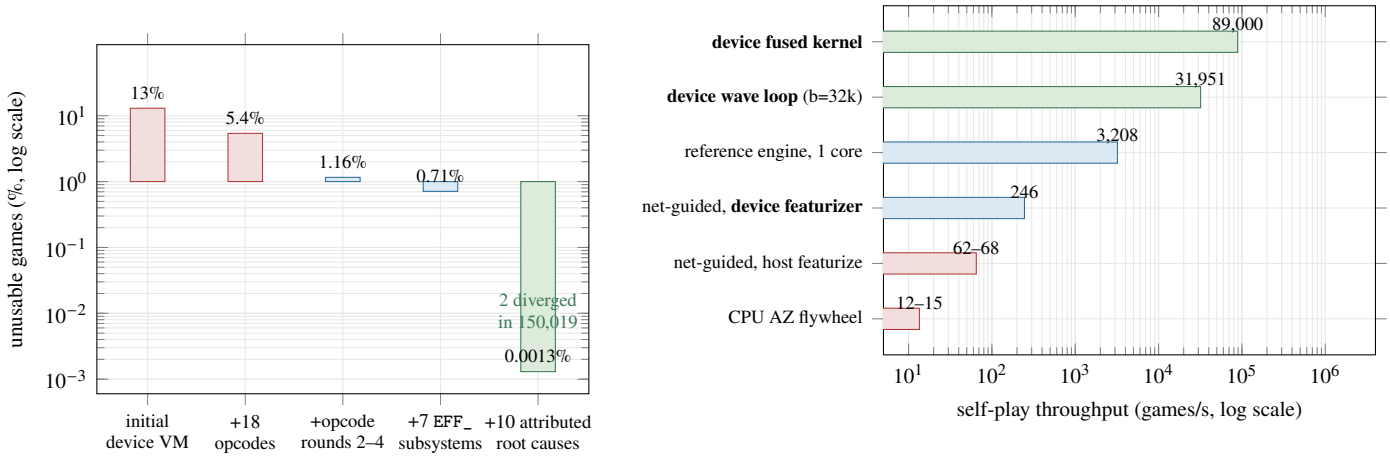
Loud by construction. Unimplemented device paths set an error register that surfaces as a distinguished hash value, so an unported opcode *cannot* pass silently; it guarantees a divergence at the exact step. Every real defect below was caught this way.

De-circularization. We report one methodological failure in full, because the lesson generalizes. Our first featurizer-parity harness gated "drift" on a quantity the featurizer *itself* produced; its committed artifact reported `PASS` over *zero* consistent samples, having excluded all 3,000 as drift. The claim "certified bit-exact, 0 bugs" was withdrawn and the artifact deleted. The rebuilt gate keys on the independently certified device legality query, so a featurizer bug can no longer hide inside its own drift filter. Any parity harness whose exclusion criterion is computed by the component under test is measuring nothing; we recommend making that check explicit in any such effort.

Table 1. Engine parity certificate. Candidate: the CUDA device VM (one thread = one full-fidelity game). Reference: the live C++ engine, driven by its own strong rules policy over the 35-deck registry (70 distinct deck pairings, 1,000 games each in the certificate run). "Exact" means identical legal-move count, identical canonical full-state hash *including hidden zones and mid-effect transients*, and identical result, at every agent decision.

Tier	Harness	Games	Agent steps	Result
Canonical	per-step parity, 24 games	24	~2,400	24/24 exact
Mid	batch parity, 1,050 games	1,050	117,508	1050/1050 exact
Reference (Python)	pre-port triage	700	77,543	700/700 exact
Certificate	batch parity, 70k games	**70,000**	**7,888,623**	**70,000/70,000 exact**

Certificate runtime 35.4 min on one RTX PRO 4000 Blackwell (≈2,300 games/min end-to-end including host-side hash verification); 0 divergences of any kind.



(a) Coverage burn-down.

(b) Throughput ladder (log scale).

Figure 3. Correctness first, then speed. (a) Usable device self-play — the fraction of generated games that replay bit-exactly through the reference engine and may therefore become training data — rose from 87% to 99.9987% as loud faults were ported and, later, as silent divergences were individually attributed to reference-engine lines. (b) Measured throughput. The two device bars are the same certified kernels; the fused bar runs whole games in one launch, the wave bar pays ~2.8× for the host round-trips that make traces and policies extractable. Note the honest asymmetry: the fastest streams carry the weakest targets (§3.6).

3.4 What the gate caught

The certificate is only interesting because it failed first. The burn-down was driven by triage rounds that ranked unimplemented surface by hit rate under real play, and it exposed defects that no unit test would have found:

- An ability-availability predicate approximated as always-true, silently offering abilities the reference engine gates out — ~20 legality divergences per 350 games, across ten distinct cards.
- A *single* mis-identified card constant — a tool-suppression check reading a Pokémon's identifier instead of the stadium card that actually suppresses tools — which was the shared root cause of ~ 190 divergences spread over four superficially unrelated symptoms: retreat legality, attack affordability, prize selection, and terminal early-wins. Attributing them to one line required replaying device traces through the reference engine and set-differencing the legal options at the first divergent step.
- An "all-bench" predicate that should have been "any-bench" (~465 divergences), a printed-HP test that should have read current HP (~167), and an energy-count that counted cards rather than provided units (~9).

Usable device self-play coverage improved 87% → 94.6% → 98.84% across opcode rounds, and then — once loud faults were near zero and the remaining loss was *silent* divergence — from 99.24% to 99.9987% (2 diverged in 150,019 games) as ten attributed root causes were fixed (Fig. 3a). We report one negative here too: a plausible stadium-interaction hypothesis for the residual was *disproven* by lockstep bisection and was not forced into a fix.

Table 2. Measured self-play throughput on the research rig (2× RTX PRO 4000 Blackwell, 16 cores; competing jobs live during all measurements). Streams are grouped by the *quality of the training target* they produce, because throughput is only comparable within a group.

Target quality	Stream	games/s	Bound by
MCTS visit counts	CPU AlphaZero flywheel (PUCT-64, 10 workers)	12–15	CPU tree search
	same, concurrency 4×	12–15	unchanged (see §3.6)
1-ply net-guided	device VM + host featurization	62–68	85% host per-slot loop
	device VM + **device featurizer** (in-pipeline RL)	**246**	GPU (scales with batch)
random-legal (value stream)	reference engine, one CPU core	3,208	CPU
	device wave loop, batch 32k	**31,951**	GPU (5.05M steps/s)
	device fused whole-game kernel	**89,000**	GPU

3.5 Throughput, and an honest accounting of it

Table 2 is the measured ledger. The headline is real and so is its caveat: the 2,000× figure compares a device stream carrying *random-legal* policy targets against a CPU stream carrying *MCTS visit-count* targets. Those data are not interchangeable. We therefore report the streams separately by target quality, and we use the random-legal stream only as a *value* stream (one decorrelated position per game, policy weight zero), which is what its targets support.

Within the comparable groups the gains are: 4× for net-guided self-play once the featurizer moves on-device (62 → 246 games/s), and 10× over a full CPU core for the value stream (3,208 → 31,951 games/s, i.e. roughly the whole 16-core machine's worth of engine throughput from one otherwise-idle GPU, while the cores do something else). End-to-end, the value-stream pipeline produces ∼10,300 games/s including host-side featurization.

3.6 Where the remaining bottleneck actually is

With the engine *and* the featurizer resident on the device, the AlphaZero flywheel is still CPU-bound, and this is the most useful negative result in the systems half of the paper. Measured over 240 games at 64 simulations: 87 records/s at concurrency 512, 89 records/s at 2048 — a 4× increase in batch for 2% — with average GPU utilization of 4% and a host load average of 13.7 on 16 cores.

The diagnosis is structural. Policy rollouts are embarrassingly parallel and fixed-shape, which is why they ported. MCTS is a *branching, sequential, per-move* computation with data-dependent control flow and per-node pointer chasing; the visit-count target that AlphaZero consumes is defined by exactly that structure. No configuration moves it to the GPU, and real AlphaZero implementations run the tree on the host for the same reason. Amdahl's law then caps the whole loop at the tree.

The tree-free reformulation. Gumbel sequential halving [6] dissolves the obstruction. Sample Gumbel noise per root action, take the top-m arms by g + logits, split a fixed simulation budget over $\log_2 m$ halving phases, and keep the better half each phase by g + logits + $\sigma(\hat{q})$; policy improvement holds in expectation at budgets as low as $n = 16$, *with no inner tree*. Every (game, arm, sim) triple then becomes an independent engine row, and the entire search per move is a fixed-shape tensor program over the *unmodified* certified kernels: snapshot the root rows, scatter-replicate to ($G \times m \times$ sims) rows, one batched step, seeded rollouts in lockstep waves, a segment reduction to $\hat{q}$, halve, repeat. Search seeds are drawn from a mixer over (game seed, move, arm, phase, sim) so that simulations decorrelate over chance while the whole search remains reproducible from the game seed — and, critically, search never advances the main-line seed schedule, so generated traces still replay bit-exactly through the reference engine.

At the measured device step rate this projects to ∼2,500 planned decisions/s against 87 records/s for the CPU tree. We report the architecture and the projection, and label it as such: the vertical slice is built and its parity guard is in place, but the at-scale measurement belongs to future work.

3.7 An engine fork for training targets

We also forked the reference engine's self-play binding to expose two things the AlphaZero recipe needs and the upstream API withheld: *per-root-child* Q and N (so that completed-Q Gumbel targets can be formed at all, rather than

only visit distributions), and cross-deck self-play (upstream dealt both seats the same decklist). The diff is 91 added and 27 removed lines in one file. We verified the sign convention of the exposed Q by tracing the backup path rather than assuming it, checked the invariants ($\sum_a N(a) = n_{\text{sims}}$ at every root, $\pi \equiv N/\sum N$ exactly, zeros for unvisited and out-of-range actions) over 2,020 records, and — because a fork that changes behavior when its features are unused is a silent data corruptor — proved byte-parity with the unmodified engine when the new argument is absent: 12/12 games with identical per-step legality counts, identical full-state hashes, and an identical SHA-1 digest over the entire legacy record stream.

4 The Agent

Alphamon comprises a learned evaluator (*AZNet*) and a deploy-time search actor (*PIMC-PUCT*).

4.1 AZNet: a permutation-invariant token-set transformer

A TCG state is a *multiset* of cards distributed over zones (active, bench, discard, hand) for both players. We encode it as an unordered token set and process it with a permutation-equivariant set transformer, following Deep Sets and the Set Transformer [17, 18]. Each of the 36 tokens (24 board + 12 own-hand; the opponent's hand is hidden and maps to a padded zone, making the encoder structurally leak-free) is embedded as a *hybrid* of a learned card-ID embedding and an attribute MLP over static card statistics, plus a zone marker and live per-Pokémon features:

$$h_i^{(0)} \;=\; E_{\text{id}}(c_i) \;+\; g_\phi(a_i) \;+\; E_{\text{zone}}(z_i) \;+\; W_\ell\, \ell_i \;\in\; \mathbb{R}^{128}. \tag{1}$$

The hybrid token resolves a cold-start pathology we measured directly: with CPU-scale self-play, a pure card-ID embedding table is starved — a single-deck corpus exercises ~ 9 distinct card identities — so rarely sampled cards keep random rows. The attribute term $g_\phi(a_i)$ gives such cards a meaningful representation from HP, type, stage, and attack cost. A depth-3 stack of Induced Set-Attention Blocks (16 inducing points, 4 heads, dim 128, hidden 384) produces contextual tokens, mean-pooled to $\bar{h}$. A value head $v_\theta(s) = \tanh(w_v^\top \text{MLP}(\bar{h})) \in [-1, 1]$ and a set-attention policy head over legal-option tokens follow. The deployed network is deliberately small ($\approx$0.5M parameters): the grader gives two vCPUs, so batch-1 CPU latency, not capacity, is the binding constraint — and §5 shows capacity buys nothing here anyway.

4.2 PIMC-PUCT: belief determinization plus tree search

At deploy time the hidden state is resolved by *Perfect-Information Monte Carlo*: sample K complete worlds (opponent decklists and hidden zones) from a belief $b(\cdot \mid o)$, run a PUCT search in each on the engine's native determinized API, and aggregate root visit counts across worlds. Each simulation descends by

$$a^\star \;=\; \arg\max_a \;\left[\, Q(s,a) \;+\; c_{\text{puct}}\, \tilde{p}_\theta(a \mid s)\, \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \,\right], \qquad \tilde{p}_\theta(a \mid s) = \frac{p_\theta(a \mid s)^{1/\tau}}{\sum_b p_\theta(b \mid s)^{1/\tau}}, \tag{2}$$

with $c_{\text{puct}} = 5$ (AlphaGo 2016, Extended Data Table 5) and a *prior temperature* $\tau = 2$ that de-sharpens the behavior-cloned prior to widen search. Values are stored in the root player's frame and opponent nodes maximize $-Q$. Determinization is known to be an optimistic evaluator — it grants the searcher knowledge of the sampled world, the classical strategy-fusion and non-locality problems [20, 21] — and we prove the optimism direction formally in Lean (§8); the mitigation is aggregating over K resampled worlds rather than committing to one, which is the PIMC end of the spectrum whose other end is information-set MCTS [19]. We use PIMC because the competition engine exposes a native determinized-search entry point, making each world's search run at engine speed.

The λ-mix leaf. The leaf evaluator blends the static value head with a full rollout to terminal under a fast rules policy:

$$V(s_L) \;=\; (1 - \lambda)\, v_\theta(s_L) \;+\; \lambda\, z_{\text{roll}}(s_L). \tag{3}$$

As $\lambda \to 1$ the value head is bypassed and the leaf becomes a pure Monte-Carlo playout — the AlphaGo-2016 regime. §7 shows the deployed configuration is exactly $\lambda = 1.0$, and §5 explains why that is the *correct* response to a variance-capped value rather than a defect.

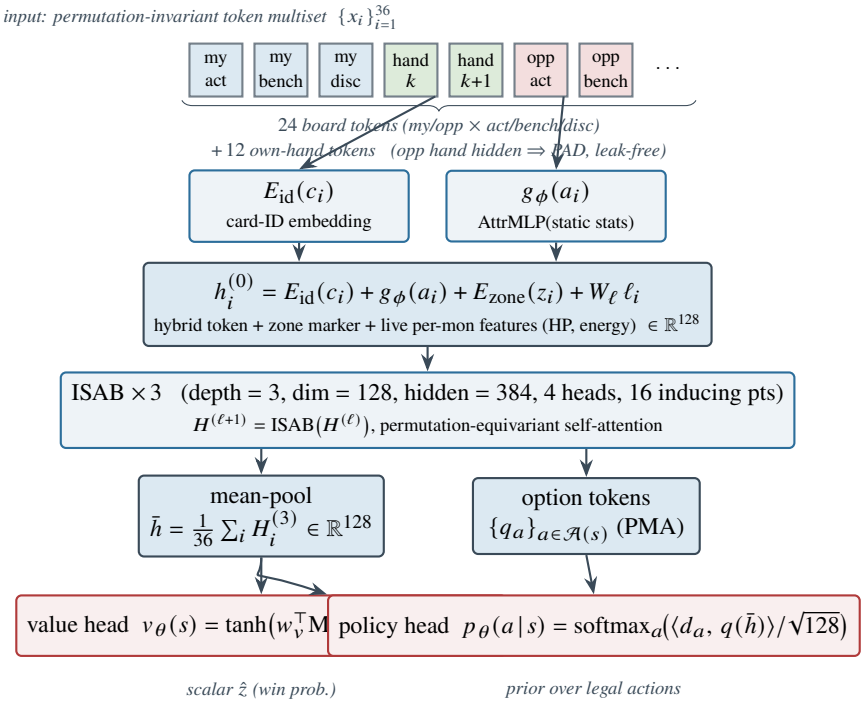


Figure 4. AZNet data flow. A permutation-invariant multiset of 36 hybrid tokens (card-ID embedding + attribute MLP + zone marker + live per-mon features) is contextualized by a depth-3 ISAB set-transformer trunk, mean-pooled, and read out by a tanh value head and a set-attention policy head over legal-option tokens. The opponent hand is hidden, so its tokens stay padded — the encoder is structurally leak-free.

Belief posterior. Let $\Theta = \{D_1, \ldots, D_M\}$, $M = 40$, be the measured ladder archetypes with prior counts n_m. Given the revealed opponent multiset r,

$$b(D_m \mid r) \;=\; \frac{n_m\, \mathbf{1}[\, r \subseteq D_m \,]}{\sum_{m'} n_{m'}\, \mathbf{1}[\, r \subseteq D_{m'} \,]}, \tag{4}$$

a count-weighted restriction to archetypes still feasible given what has been revealed. When the feasible set empties, b falls back to a card co-occurrence model estimated from 22,852 observed ladder decklists. The K worlds are i.i.d. draws from b, making the search value a Monte-Carlo integral over the belief. Note — foreshadowing §6 — that this belief feeds only the *determinizer*, not the value head; DeepStack would call that incomplete, and we show in §5 that here it is *inert*.

4.3 Training pipeline and the data program

Training follows the AlphaGo stage order — supervised behavior cloning, policy RL, a value stage, and AlphaZero-style self-play — under the standard coupled loss

$$\mathcal{L}(\theta) \;=\; \big(z - v_\theta(s)\big)^2 \;-\; \sum_a \pi(a \mid s)\, \log p_\theta(a \mid s) \;+\; c\, \|\theta\|_2^2. \tag{5}$$

The supervised stage deserves a note, because it is where a competition setting differs from a research one. Our first agents were graded against a handful of scripted opponents; the live ladder promptly showed that 59% of the field consisted of decks we had never trained against, and that offline 0.66 corresponded to live 0.49. We were optimizing the wrong opponents. We therefore built a measured-metagame program: a daily census of ladder decklists (ultimately 61,520 harvested lists, a 3,228-deck universe pool, and the 40-archetype belief model above), and a behavior-cloning corpus from 24,861 top-player games (1.8M decision records; later 3.48M for a full-corpus streaming run). Featurization always uses the *mover's* deck, an invariant we had to state explicitly after discovering that our deployed network had been trained on a different decklist than the one it piloted — its embedding rows for the deck's own cards were random initialization.

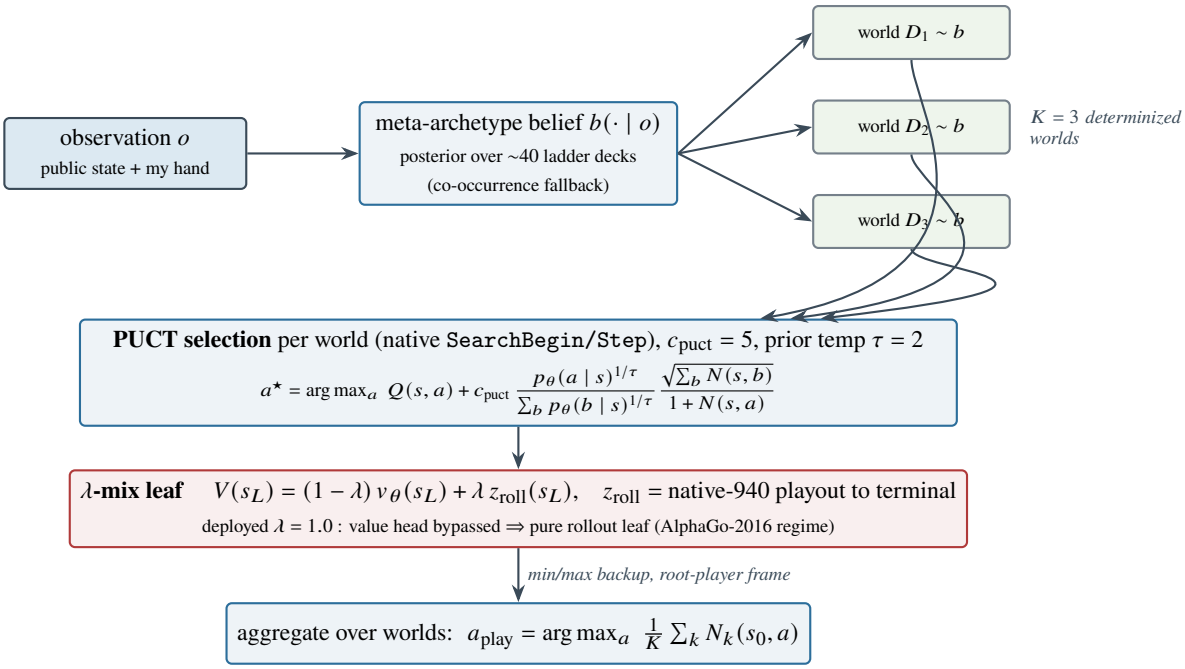


Figure 5. PIMC-PUCT deploy actor. A meta-archetype belief converts the partial observation into K=3 determinized worlds; PUCT (c_{puct}=5, prior temperature τ=2) searches each on the native engine; leaves are evaluated by the λ-mix with a rules-policy rollout to terminal. Root visit counts are averaged across worlds to choose the played action.

5 The Aleatoric Ceiling

This section gives the analysis and the measurements. The structure is: define what a value function can possibly achieve, give a cheap estimator of it, then show that the classical ways of approaching it are spent.

5.1 The value target is a conditional expectation

Fix a behavior distribution over states and let $z \in \{-1, +1\}$ be the eventual mover-frame outcome. The MSE-optimal value function is the conditional expectation

$$v^\star(s) \;=\; \mathbb{E}[z \mid s] \;=\; 2p(s) - 1, \qquad p(s) := \mathbb{P}(\text{mover wins} \mid s) \in [0,1], \tag{6}$$

with conditional variance $\mathrm{Var}(z \mid s) = 1 - v^\star(s)^2 = 4p(s)(1-p(s))$.

Proposition 1 (Bias–variance floor)**.** *For any measurable v_θ,*

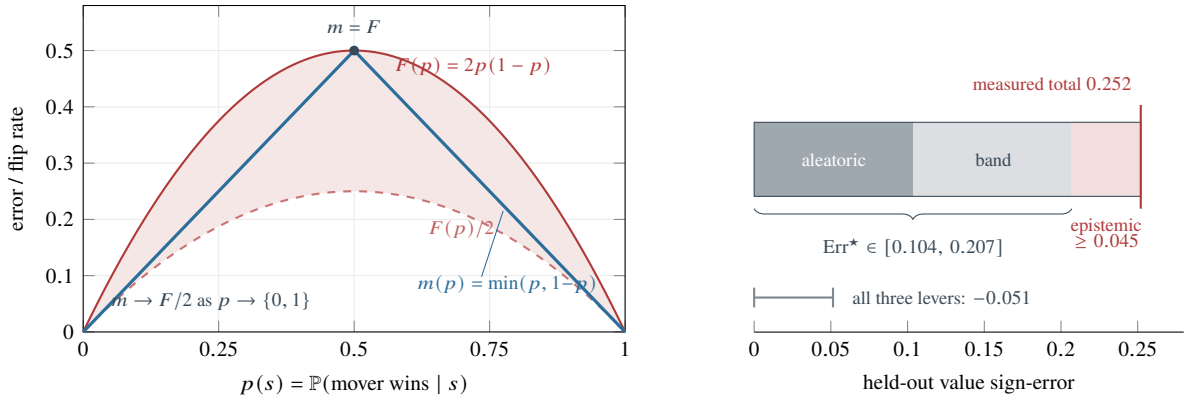
$$\mathbb{E}\big[(z - v_\theta(s))^2\big] \;=\; \underbrace{\mathbb{E}\big[(v_\theta(s) - v^\star(s))^2\big]}_{\text{epistemic (reducible)}} \;+\; \underbrace{\mathbb{E}\big[\,1 - v^\star(s)^2\,\big]}_{\text{aleatoric (irreducible)}}. \tag{7}$$

Proof. Add and subtract $v^\star(s)$ and expand; the cross term $\mathbb{E}[(v_\theta - v^\star)(z - v^\star)] = \mathbb{E}[(v_\theta - v^\star)\,\mathbb{E}[z - v^\star \mid s]] = 0$ since $\mathbb{E}[z \mid s] = v^\star(s)$. The second term is $\mathbb{E}[\mathrm{Var}(z \mid s)]$. □

The first term is what data, capacity, and information shrink. The second is a property of the *game*.

5.2 The sign-accuracy ceiling is the Bayes accuracy

We evaluate value networks by sign-accuracy: does $\mathrm{sign}\, v_\theta(s)$ match the realized z? The best possible predictor bets on the more likely side.



(a) The bound is geometric.

(b) Where the measured error lives.

Figure 6. Prop. 3, and what it says about our numbers. (a) The Bayes error $m(p) = \min(p, 1-p)$ is trapped in the shaded band between $F(p)/2$ and $F(p) = 2p(1-p)$ at every p, so the containment survives taking expectations — which is why an average flip rate, measurable without training anything, bounds the accuracy of every value function. The upper bound is attained at $p = \frac{1}{2}$ and the lower as $p \to \{0, 1\}$. (b) The measured held-out sign-error of 0.252, decomposed. The band width is the price of estimating the ceiling by replay instead of computing $p(s)$; what lies to its right is epistemic and, by §5.4, not reachable by data, capacity, or information — which together bought 0.0514.

Proposition 2 (Bayes sign-accuracy)**.** *Over the state distribution, no evaluator exceeds*

$$\mathrm{Acc}^\star \;=\; \mathbb{E}_s\big[\,\max\big(p(s),\,1-p(s)\big)\big] \;=\; \tfrac{1}{2} + \tfrac{1}{2}\,\mathbb{E}_s\big|v^\star(s)\big|, \qquad \mathrm{Err}^\star = \mathbb{E}_s\big[\,\min(p, 1-p)\big]. \tag{8}$$

$\mathrm{Err}^\star$ is zero *iff* $p(s) \in \{0, 1\}$ almost surely — the outcome is a deterministic function of the state. That is the Go regime. When many states have $p(s)$ interior, the floor is strictly positive and no amount of learning removes it.

5.3 Measuring the ceiling without training anything

Propositions 1–2 are only useful if $\mathrm{Err}^\star$ can be estimated, and $p(s)$ is not observable. It can: an engine that can be re-seeded gives it up directly.

Definition 1 (Replay-flip rate)**.** *For a state s and a fixed policy pair, let z_1, z_2 be the outcomes of two independent continuations from s under fresh randomness. The* flip rate *is $F(s) = \mathbb{P}(z_1 \neq z_2) = 2p(s)\big(1 - p(s)\big)$, and $\mathbb{E}[F]$ is its average over the state distribution.*

Proposition 3 (Replay-flip bounds the Bayes error)**.**

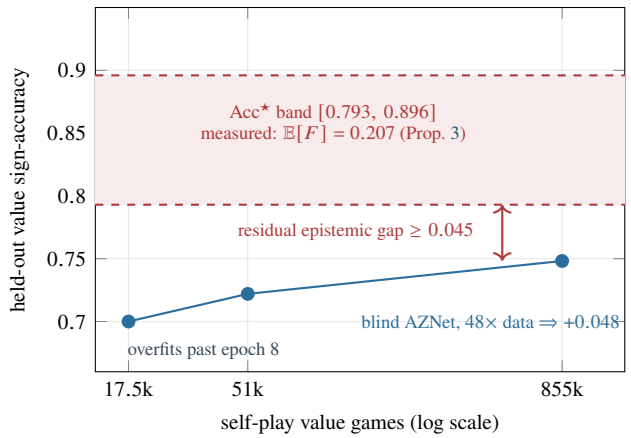
$$\tfrac{1}{2}\,\mathbb{E}[F] \;\le\; \mathrm{Err}^\star \;\le\; \mathbb{E}[F], \qquad \textit{equivalently} \qquad 1 - \mathbb{E}[F] \;\le\; \mathrm{Acc}^\star \;\le\; 1 - \tfrac{1}{2}\mathbb{E}[F]. \tag{9}$$

Proof. Write $m = \min(p, 1-p) \in [0, \frac{1}{2}]$, so $1 - m \in [\frac{1}{2}, 1]$ and $F = 2m(1-m)$. Then $F = 2m(1-m) \ge 2m \cdot \frac{1}{2} = m$ and $F = 2m(1-m) \le 2m \cdot 1 = 2m$; hence $m \le F \le 2m$, i.e. $\frac{F}{2} \le m \le F$ pointwise. Take expectations and apply $\mathrm{Err}^\star = \mathbb{E}_s[m(s)]$. □

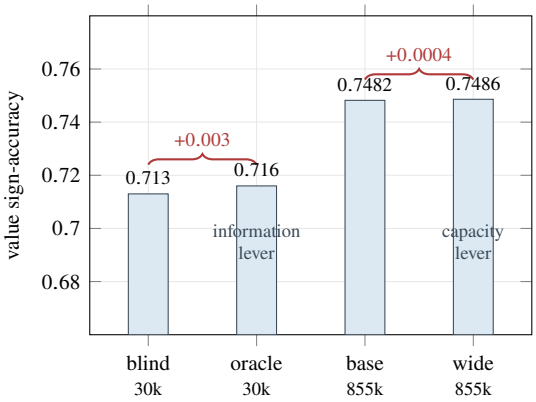
Both bounds are attained: $p = \frac{1}{2}$ gives $F = m = \frac{1}{2}$, saturating the *upper* bound, while $p \to \{0, 1\}$ drives $m/F \to \frac{1}{2}$, saturating the *lower* one. The quantity is computable from engine replays alone, with no network, no training, and no labels beyond game outcomes.

Measurement. Replaying held-out mid-game positions under fresh randomness, we measure $\mathbb{E}[F] = 0.207$ for the deployed deck (0.207 for the strongest ladder archetype, 0.256 and 0.45 for two others — decks differ substantially in how much they leave to chance, which is itself an actionable deck-selection criterion). Correspondingly, when a side is ahead at the 60% mark of the game it wins only 0.789 of the time. Prop. 3 then brackets

$$\mathrm{Err}^\star \in [0.104,\; 0.207], \qquad \mathrm{Acc}^\star \in [0.793,\; 0.896]. \tag{10}$$



(a) Data scaling saturates far below the measured ceiling band.



(b) Information and capacity are inert (same-data controls).

Figure 7. The classical epistemic levers are spent well below the Bayes band. (a) Held-out value sign-accuracy rises 0.700 → 0.722 → 0.748 as self-play data grows 17.5k → 51k → 855k (48×), then overfits; a wide network on the same corpus is identical. The shaded band is $\text{Acc}^\star \in [0.793, 0.896]$ from the replay-flip measurement (Prop. 3). (b) A perfect-information oracle scores 0.716 vs a blind network's 0.713 on identical 30k data, and a wide network 0.7486 vs 0.7482.

For calibration: the same quantity in Go or chess is $\mathbb{E}[F] = 0$ by determinism, and $\text{Acc}^\star = 1$. The single measurement that took one afternoon of engine replays is therefore the one that should have preceded the value-scaling program described next.

Remark 1 (Distributional caveat, stated plainly)**.** $\mathbb{E}[F]$ *was measured on mid-game positions under a fixed rules-policy pair, whereas the value network is evaluated on its own recorded state distribution, which includes early positions where* p *is closer to* $\frac{1}{2}$ *and* F *is correspondingly larger. The two are therefore not the same integral, and* (9) *should be read as an indicative band rather than a same-distribution bound on our specific evaluation set. The estimator is exact when measured on the evaluation distribution itself, which is what we recommend for future use.*

5.4 Every classical epistemic lever, measured

The epistemic term in (7) has three classical reducers — more *data*, more *capacity*, more *information*. We drove each to exhaustion with same-data controls (Fig. 7).

Data. Trained on an 855k-game self-play corpus (48× a 17.5k baseline), held-out sign-accuracy climbs 0.700 → 0.722 → 0.748 and then overfits past epoch 8. The marginal accuracy per doubling collapses toward zero: 48× the data buys 4.8 points.

Capacity. A wide AZNet on the identical 855k corpus scores 0.7486 against the base network's 0.7482 — a difference of 0.0004. The high value MSE at saturation (≈0.67) is the signature of fitting *noise*, not of failing to fit signal.

Information. The decisive control is an *oracle* value network trained with the opponent's hidden hand, prizes, and deck order revealed through a debug channel. On identical 30k data it scores 0.716 against the blind network's 0.713: perfect information is worth +0.003. If seeing *all* hidden state does not move accuracy, the residual is not about concealment.[1]

5.5 Why the residual does not yield: noise inflates sample complexity

It is tempting to read Fig. 7 as "the network is at the Bayes limit." It is not, and the difference matters. The measured accuracy 0.748 sits below the band $[0.793, 0.896]$, so a residual *epistemic* error remains — yet data, capacity, and

[1]We flag a data-scale pitfall we tripped on ourselves: one must not compare the 855k blind network's 0.748 against the 30k oracle's 0.716 and conclude "blind beats oracle." The valid comparison is same-data, 0.716 vs 0.713. The error was caught in review and the conclusion corrected.

information each moved it by essentially nothing.

The resolution is that aleatoric noise does not only set a floor; it inflates the cost of approaching it. Estimating a regression function to accuracy ε under label noise of variance σ^2 requires a sample size scaling as σ^2/ε^2. Here the labels are ± 1 with conditional variance $1 - v^\star(s)^2 = 2F(s)$, whose mean the flip measurement puts at $\mathbb{E}[\mathrm{Var}(z \mid s)] = 2\,\mathbb{E}[F] \approx 0.41$ on mid-game positions, against a maximum of 1 — where a deterministic game has exactly 0 and the label *is* the function. A 48× data increase reduces the statistical error by only $\sqrt{48} \approx 6.9\times$, which is exactly the regime in which a plateau appears long before the Bayes limit is reached.

This sharpens the conclusion rather than softening it. The three levers we tested vary *how much* signal is available; none of them varies the estimator's *noise robustness*. The remaining epistemic gap is therefore a target for variance-aware estimation — distributional value heads [14, 15], rollout-averaged targets, or explicit chance nodes [5] — and not for a bigger corpus or a bigger network. We regard this as the single most useful forward-looking statement in the paper, and note that it is only formulable *because* the ceiling was measured independently: without Prop. 3, 0.748 and a plateau are indistinguishable from a solved problem.

5.6 The improvement operator is structurally inert

Even granting a better value function, the AlphaZero flywheel here cannot consume it. Wiring the strongest value network (0.748) into Gumbel reanalyze changes 0.00% of policy targets. The reason is combinatorial, not statistical: completed-Q reanalyze uses the value only to *complete* unvisited actions, and 91% of reanalyzed states have every legal action already visited — a direct consequence of the small branching factor ($\bar{b} = 7.8$) relative to the simulation budget. The completion term multiplies a zero mask.

This is worth stating as a general design warning: in games with small branching factors, the value network's influence on AlphaZero policy targets vanishes at exactly the budgets that make search affordable, and a value-quality improvement can be perfectly invisible to the training loop that is supposed to compound it. The remedy is not a better value but a target that reads it — lower simulation budgets with Gumbel top-m (which deliberately leaves arms unvisited), or a leaf estimator that consumes the value directly.

6 Why the Imperfect-Information Toolbox Does Not Apply

The natural response to "imperfect information" is to reach for counterfactual regret minimization, public-belief-state search, and continual re-solving [7, 8, 9]. We tested the premise directly, and it does not hold here. The correct analogy is backgammon's *future-chance* variance, not poker's *hidden-state* variance.

Hidden information is inert. DeepStack's core idea is a value network whose input is both players' ranges and whose output is per-hand counterfactual values; it never predicts "who wins" [7]. Our oracle control tests precisely whether resolving hidden state matters, and it does not (+0.003). The opponent model is inert on two further axes: an oracle-deck determinizer ties the learned belief (0.648 vs 0.648, $p = 1.0$) and swapping the opponent's rollout actor changes nothing (0.647–0.648). We also built the DeepStack-shaped experiment rather than only arguing from the oracle: conditioning the value input on the belief gives $\Delta = -0.0016$ and -0.0046 sign-accuracy on two splits — null.

The variance is in the future. The irreducible term is dominated by coin flips and undrawn cards: events in the game's *future*, independent of what the opponent currently holds. This is the regime TD-Gammon handled with a plain value network and short rollouts [16]. It also explains, mechanistically, why a full rollout beats a static head at the leaf: a playout *re-samples the future chance* and returns a lower-variance Monte-Carlo estimate of $v^\star(s_L)$ than any one-shot head can produce. The deployed $\lambda = 1.0$ is therefore not a broken value head; it is the variance-optimal leaf given a capped one.

Poker machinery, measured. Table 3 reports what happened when we implemented the poker templates on top of our stack, changing only the stage each paper targets.

Table 3. Poker-derived interventions, each replacing exactly one stage of the Alphamon pipeline. Win-rates are paired-seed against champion **v12** (0.50 = tie).

Source	Stage replaced	vs v12	Why it does not transfer
AlphaHoldem [10]	Trinal-Clip PPO loss	0.362 *(control* 0.611*)*	clip cushions PPO's missing stable target; MCTS already supplies a low-variance one
DeepStack [7]	belief-conditioned value	null (−0.002)	belief adds no winner-signal; oracle is +0.003
AlphaHoldem [10]	end-to-end PPO + K-best pool	0.522–0.582	reconstructs a weaker policy improvement

Table 4. Alphamon against the relevant self-play/search literature.

System	Family	Core mechanism	Relation to the measured wall
AlphaGo/Zero [1, 2]	Go	MCTS + value/policy self-play	recipe adopted; $\mathrm{Err}^\star \approx 0$ there
MuZero [4]	Go	learned latent model	same aleatoric issue on stochastic games
Stochastic MuZero [5]	mixed	chance nodes in the model	models chance; does not remove its variance
DeepStack [7]	poker	CFR re-solve + range value	targets hidden state; measured inert (+0.003)
ReBeL / PoG [8, 9]	poker	public-belief-state search	principled but hidden-state-oriented
AlphaHoldem [10]	poker	end-to-end PPO, Trinal-Clip	variance-aware loss; tested, no transfer
Suphx [11]	mahjong	RL + oracle-guided distill	oracle-guiding; our oracle is inert
DouZero [12]	Doudizhu	Deep Monte-Carlo, search-free	closest game family
DeepNash [13]	Stratego	model-free Nash (R-NaD)	exploitability, not aleatoric, focus
C51 / IQN [14, 15]	RL	distributional value	models the variance we are capped by
TD-Gammon [16]	backgammon	value + short rollouts	the regime we are actually in

Position. Table 4 places Alphamon against the literature. The systems that win in poker, mahjong, and Doudizhu attack *hidden-state exploitability*; Alphamon's measured ceiling is *aleatoric*, an axis those methods do not target. The nearest applicable levers are distributional value [14, 15] and explicit chance modeling [5], which change how the variance is *handled*.

7 Results

The unit of truth for offline comparisons is a paired-seed win-rate against the champion **v12**; the unit of truth for deployment is the competition ladder. §7.3 shows the two are almost unrelated, which is itself one of our results.

7.1 Search dominates every learned component

The single largest effect we measured, by a wide margin, comes from search — at *fixed* weights. Holding one behavior-cloned network constant and varying only the actor, against the engine's reference rules agent (seat-balanced, draws counted as losses):

Actor (identical weights)	win-rate	95% CI
arg max policy (no search)	0.425	—
+ 1-ply lookahead, λ-mix leaf	0.688	—
+ PIMC-PUCT, 48 simulations	**0.912**	[0.830, 0.957]

An earlier, weaker checkpoint shows the same ladder with cleanly separated intervals ($0.237\ [0.158, 0.341] \rightarrow 0.637\ [0.528, 0.734]$ for arg max $\rightarrow$ 1-ply, $n = 80$). We emphasize the control: the network is byte-identical across rows, so the entire effect is the actor.

This is the empirical shadow of §5. When the value target is variance-capped, compute spent *re-sampling the future* (search and rollouts) buys strength that compute spent *fitting the expectation* (more data, more parameters) does not. Fig. 8 plots it against the learned-component ladder for the same period.

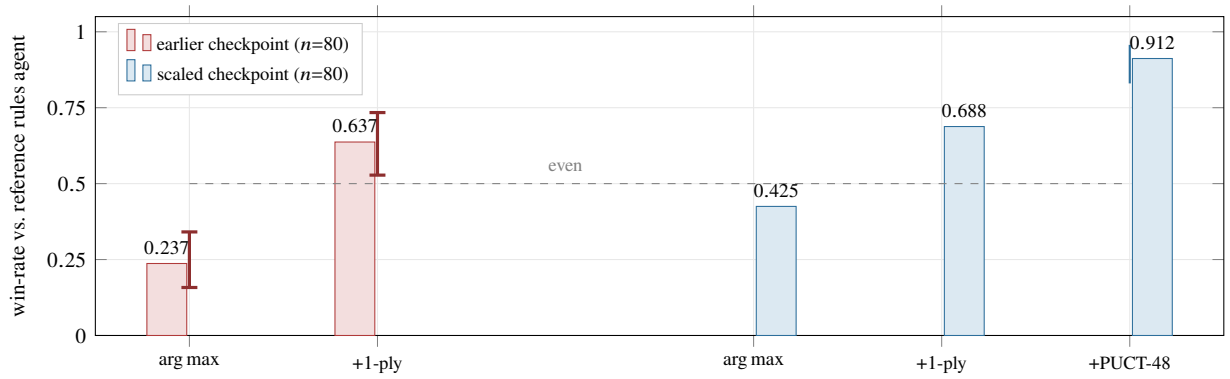


Figure 8. Search is the lever. Within each series the network weights are byte-identical and only the deploy actor varies, against the engine's reference rules agent (seat-balanced, draws counted as losses). Adding determinized PUCT more than doubles win-rate over arg max play, and the effect replicates across two checkpoints of very different strength. For scale, no retraining intervention in Table 5 moved the incumbent champion at all.

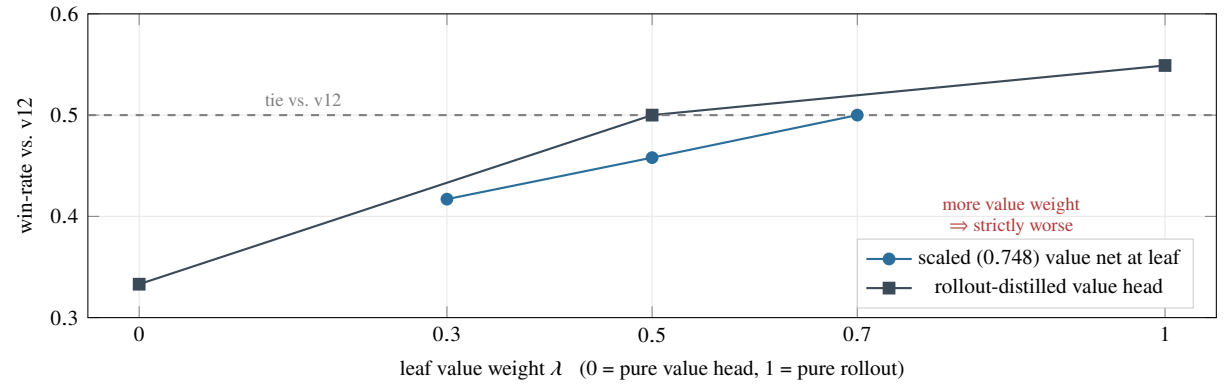


Figure 9. Value-at-leaf is dominated by rollout. Win-rate vs v12 as a function of the leaf value weight λ in (3), for a scaled (0.748) value network and a rollout-distilled head. Weighting the static head more is strictly worse; λ=1.0 (pure rollout) is the best deploy configuration — the head is bypassed, not broken.

7.2 Interventions against the champion

Table 5 is the full ledger. Two regularities hold across it. First, every training method that re-fits the same network object ties or loses to the champion: the champion's policy *is* the strength, and drift off it degrades faster than specialization helps — including when we deliberately overfit against the specific bots we were losing to, which made us worse against those very bots (0.20 against the target archetype, versus a 0.70 baseline). Second, every value-at-leaf configuration is dominated by the pure rollout (Fig. 9): win-rate falls monotonically as the leaf weights the static head more, and no $\lambda < 1$ clears the $\lambda = 1.0$ bypass. As §6 argues, that ordering is predicted by the variance analysis rather than by a defect in the head.

7.3 No offline metric predicts the ladder

Every offline metric we built failed board-validation. The most rigorous harness we could construct — a clone-invariant Nash gauntlet over seven board-anchored champions, each at its own shipped configuration, over 78 complete pairings — attains rank correlation $\rho = 0.112$ ($p_{\text{perm}} = 0.814$, CI $= [-0.25, 0.64]$) against the ladder, resolvable concordance $4/6$ ($p = 0.344$), and a hold-out correlation of $\rho = -0.46$ once the one configuration that dominated the fit is removed. The regression of ladder score on local win-rate gives $R^2 = 0.018$.

We do not think this is a harness defect, and the distinction matters for anyone running a similar program. The candidate agents were separated by an anchor spread of 118 rating points, while resubmitting an *unchanged* bundle moves its settled rating by up to 149 points (§7.5); the separation we were trying to resolve is smaller than the noise

Table 5. Every intervention against champion **v12** (paired-seed win-rate; 0.50 = tie). No re-trained network beats v12; no value-at-leaf configuration clears the λ=1.0 rollout bypass. Receipts in `runs/`.

Axis	Intervention	vs v12	Verdict
Training	self-play iter1 (unanchored)	0.31	collapse
	self-play iter2 (KL-anchored)	0.45	tie, promoted nothing
	targeted overfit vs winning bots	0.31	*worse*, incl. vs its own targets
	behavior clone of top-10 winners	0.45	tie
	full-corpus SFT (3.48M records)	0.47	tie
Value @ leaf	scaled (0.748) net, λ=0.3/0.5/0.7	0.42/0.46/0.50	monotone loss
	rollout-distilled head, λ=0.0/0.5/1.0	0.33/0.50/0.55	no λ<1 win
	value recalibration (×4 variants)	$\approx$ 0.50	dead (leaf bypasses the head)
	belief/range-conditioned value	null	−0.002 sign-acc; oracle is +0.003
Opponent model	oracle-deck determinizer	0.648[†]	ties native (p=1.0)
	opponent rollout actor (beam/net)	0.647–0.648[†]	inert
Deploy lever	PIMC worlds K=1 vs K=3	$\approx$ 0.50	K=1 ties; reclaims 3× budget
	first-play urgency (FPU = 0.2)	+0.063[‡]	directional only; see §7.4
	λ=1.0 pure rollout (shipped)	+0.11	best deploy configuration

[†]Absolute win-rate against the native baseline in the opponent-model canary, not against v12. [‡]n = 8, not significant.

on a single label. The ladder ranking of agents this close is not a learnable function of local play, because the labels themselves are not reproducible. The correct response is not a better harness but a decision rule that only acts on differences exceeding the label noise — and, where that is unaffordable, an explicit acknowledgment that candidate selection is unresolved. Our gate refuses to promote under these conditions, which we now regard as its most valuable behavior.

7.4 A retraction

An earlier version of this work reported that first-play-urgency was "board-refuted" by a 363-point gap between two submissions (750.5 vs 387.5) that differed only in that lever, and treated the gap as real because it exceeded the volatility band we assumed at the time. Those were *transient* ratings read before convergence. At convergence the same two submissions settled at 610.9 and 521.9 — a gap of 89 points, comfortably *inside* the noise band. The correct conclusion is that the experiment was uninformative in both directions: FPU was neither confirmed nor refuted, and the earlier claim is withdrawn. We report this in the body rather than a footnote because it is a concrete instance of the failure mode §7.3 warns about — reading a non-stationary rating as a measurement — and because we made it after writing the warning.

7.5 Competition outcome

Table 6 gives the settled ladder scores of every submission. The best converged score of any agent we shipped was 636.5; the final standing of the entry was 516.5, ranking 4741 of 6807 teams (top score 1398.2, median 621.5).

Two facts explain the gap between 636.5 and 516.5, and both are operational rather than algorithmic. Only the two most recent active submissions count at the deadline, and our last submissions were made three weeks before it; the champion was therefore not among the two agents that were scored. And the ladder moves agents that did not change. We resubmitted unchanged bundles four times and measured the spread each time (Fig. 10): two same-day duplicate pairs, uploaded twenty minutes apart with no opportunity to rebuild, settled 82.4 and 87.7 points apart; the champion bundle, fired three times, settled at 636.5, 610.9 and 487.7, a spread of 148.8. The median pairwise spread over the five comparisons is 87.7 points. This is a lower bound on the label noise of the ranking, measured on our own submissions rather than inferred, and it exceeds the 118-point spread separating the candidates we were trying to choose between (§7.3). We report this because submission scheduling under a non-stationary rating is a real and under-discussed component of results on live-ladder benchmarks, and because it is precisely the kind of detail that is usually omitted.

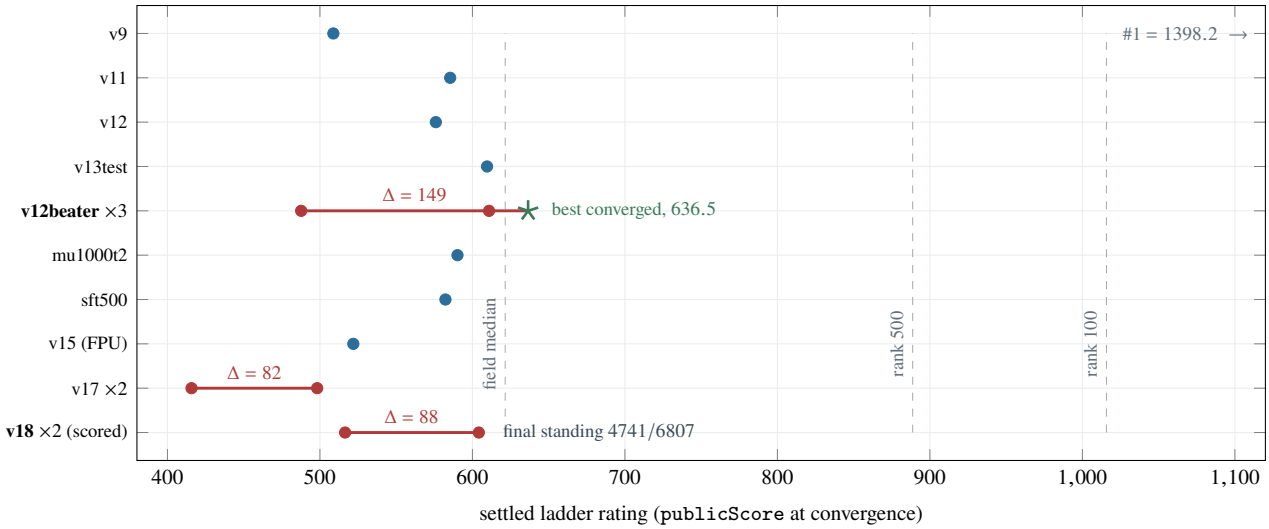


Figure 10. The ruler, measured. Settled ladder ratings for every net-based submission, against the field's own quantiles. Red segments join submissions of *unchanged* bundles: the champion bundle fired three times spans 148.8 points, and two same-day duplicate pairs — uploaded twenty minutes apart — span 82.4 and 87.7. The entire span of the distinct agents we produced, 508.8 to 636.5, is comparable to the noise on a single one of them, which is the quantitative content of "no offline metric predicts the ladder" (§7.3): the labels are not reproducible at the resolution the ranking demands.

8 Discussion

What the measurements establish. Three things, in decreasing order of generality. (i) A game's suitability for the AlphaZero recipe is measurable before any training, from engine replays alone, via Prop. 3; and the number for this game — a Bayes value ceiling of $[0.79, 0.90]$ rather than 1.0 — is what a value-scaling program is competing against. (ii) When that ceiling is low, the returns to compute invert: search and rollouts, which re-sample the chance, dominate data and capacity, which fit its expectation. We measure a $0.425 \rightarrow 0.912$ swing from the former and +0.048 from a 48× increase in the latter. (iii) The AlphaZero improvement operator can be inert for reasons unrelated to value quality; in small-branching games the completed-Q mask is empty and value improvements are invisible to the loop by construction.

What the engine establishes. That a full commercial-scale card-game engine *can* be moved onto the GPU bit-exactly, at one game per thread, with a certificate strong enough to gate training data on — and that the binding constraint afterwards is not the rules but the tree. The certification methodology is the transferable part: build the oracle first, hash the complete state including hidden zones and mid-effect transients, make unimplemented paths loud, and never let the component under test define its own exclusion criterion.

Limitations. (i) $\mathbb{E}[F]$ was measured on a mid-game distribution rather than the value network's own evaluation distribution, so (9) is indicative for our specific numbers; the estimator itself is exact when measured on-distribution. (ii) Ladder-based rulings ride a non-stationary rating whose noise we measure at up to 149 points on unchanged bundles; we require margins that clear it and, where none exists, report the question as unresolved rather than resolved. (iii) We did not build a full range-conditioned re-solver; our belief-value experiment and oracle control argue it would be inert here, but that is an inference from two controls, not a test of that exact system. (iv) The GPU Gumbel search is implemented and parity-guarded but its at-scale training value is projected, not measured. (v) The competition entry's final standing reflects submission scheduling as well as agent strength (§7.5).

What we would do next. Everything consistent with the diagnosis changes the *estimator* or the *actor*, not the amount of data. Ranked: a stronger learned rollout policy at the leaf, which improves the Monte-Carlo estimate of $v^\star$ directly and is the most AlphaGo-faithful lever; distributional, variance-aware value that plays to minimize variance when

Table 6. Settled ladder scores for every submission (`publicScore` at convergence). Local win-rate against v12 does not predict rank ($R^2 = 0.018$), and resubmitting an unchanged bundle moves its settled score by up to 149 points.

Submission	Settled score	Note
v9	508.8	early net baseline
v12	576.0	reference champion (AZ flywheel)
sft500	582.3	full-corpus SFT policy
v11	585.4	
mu1000t2	590.2	full-data policy; the "more data" bet did not pay
v13test	609.6	
v14 (≡ v12beater bytes)	610.9	re-fire of the champion, md5-verified identical
v16 (champion bundle again)	487.7	3rd firing: 148.8 below its own best sample
v12beater	**636.5**	**best converged score** (bal400, λ1.0, τ2.0)
v15 (FPU = 0.2)	521.9	89 pts from v14: inside the noise band (§7.4)
v18 / v17 (final pair)	516.5 / 415.8	scored at the deadline; **final standing** 4741/6807

ahead and maximize it when behind [14, 15]; explicit chance nodes so that evaluation is computed *over* the dice [5]; and the low-simulation Gumbel regime, which is both the tree-free GPU target of §3.6 and the one target format that can actually read a value improvement. None is a bigger corpus.

Conclusion. Built faithfully, the AlphaGo recipe transfers to a stochastic imperfect-information TCG in its *search* component and stalls in its *learning* component, and the stall is predictable in advance from a quantity that costs an afternoon to measure. The Bayes value ceiling is a property of the game, the sample complexity of approaching it scales with the game's own noise, and the improvement operator that is supposed to compound value gains can be structurally unable to see them. Naming these three quantities, giving a cheap estimator for the first, and providing a certified GPU substrate on which they can be measured for other games, is the contribution we intend to be reusable.

Reproducibility and Formal Support

Every quantitative claim regenerates from on-disk artifacts: the engine parity certificate (`results/parity/gate_cuda_70000.json`, per-matchup counts, zero divergences) and its burn-down (`results/parity/triage_*.json`); the device featurizer certificate (`results/parity/featurizer_device_cert.json`, 5,362 engine-consistent samples, zero featurizer failures); value scaling (`runs/e2_scale*.log`); the oracle canary (`runs/oracle_canary.log`); the λ sweeps (`runs/vrecal_rollout/`); the gauntlet board-validation (`runs/gauntlet_cal_v2/validation.json`); and the leaderboard ledger via the competition API. Throughput numbers are steady-state walls printed by the tools themselves (`runs/cuda_selfplay_pilot/*.stats.json`).

The game-theoretic assumptions the search rests on are machine-checked in Lean 4 (`lean/`, no Mathlib, compiles clean): bounded zero-sum outcomes and sound min/max backups (`GameTree`, `Basic`), UCB in-range selection with unvisited-arm priority (`Bandit`), PUCT prior-scaling monotonicity (`PUCTPrior`), budget monotonicity (`BudgetMonotone`), Gumbel sequential-halving budget accounting (`GumbelBudget`), truncated-leaf and deep-backup soundness (`TruncatedLeaf`, `DeepTreeBackup`), belief-value consistency (`BeliefValue`), graded-target sign/order preservation (`GradedTargets`), deck legality (`DeckRules`), gate composition (`CompositeGate`), and — most relevant to §4 — that determinization is an *optimistic* upper bound, with an explicit strict-gap witness (`Determinization`).

References

[1] D. Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[2] D. Silver et al. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

[3] D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362:1140–1144, 2018.

[4] J. Schrittwieser et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, 2020.

[5] I. Antonoglou et al. Planning in stochastic environments with a learned model (Stochastic MuZero). *ICLR*, 2022.

[6] I. Danihelka, A. Guez, J. Schrittwieser, D. Silver. Policy improvement by planning with Gumbel. *ICLR*, 2022.

[7] M. Moravčík et al. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356:508–513, 2017. arXiv:1701.01724.

[8] N. Brown, A. Bakhtin, A. Lerer, Q. Gong. Combining deep reinforcement learning and search for imperfect-information games (ReBeL). *NeurIPS*, 2020. arXiv:2007.13544.

[9] M. Schmid et al. Player of Games. arXiv:2112.03178, 2021.

[10] E. Zhao et al. AlphaHoldem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. *AAAI*, 2022.

[11] J. Li et al. Suphx: Mastering Mahjong with deep reinforcement learning. arXiv:2003.13590, 2020.

[12] D. Zha et al. DouZero: Mastering Doudizhu with self-play deep reinforcement learning. *ICML*, 2021. arXiv:2106.06135.

[13] J. Perolat et al. Mastering the game of Stratego with model-free multiagent reinforcement learning. *Science*, 378:990–996, 2022. arXiv:2206.15378.

[14] M. G. Bellemare, W. Dabney, R. Munos. A distributional perspective on reinforcement learning. *ICML*, 2017. arXiv:1707.06887.

[15] W. Dabney, G. Ostrovski, D. Silver, R. Munos. Implicit quantile networks for distributional reinforcement learning. *ICML*, 2018. arXiv:1806.06923.

[16] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[17] M. Zaheer et al. Deep Sets. *NeurIPS*, 2017.

[18] J. Lee et al. Set Transformer: A framework for attention-based permutation-invariant neural networks. *ICML*, 2019.

[19] P. I. Cowling, E. J. Powley, D. Whitehouse. Information Set Monte Carlo Tree Search. *IEEE Trans. Comp. Intell. AI in Games*, 4(2):120–143, 2012.

[20] I. Frank, D. Basin. Search in games with incomplete information: A case study using Bridge card play. *Artificial Intelligence*, 100(1–2):87–123, 1998.

[21] J. R. Long, N. R. Sturtevant, M. Buro, T. Furtak. Understanding the success of perfect information Monte Carlo sampling in game tree search. *AAAI*, 2010.