A GPU ENVIRONMENT FOR CHINESE STANDARD MAHJONG WITH A JUDGE-REPLAY FIDELITY CONTRACT AND REFERENCE RL WORKLOADS

Anonymous authors
Paper under double-blind review

ABSTRACT

Accelerator-native simulators have made reinforcement learning (RL) on board games cheap, but when a game's payoff is fixed by an external rulebook, a fast simulator is useful only if it is demonstrably faithful: a mis-scored hand still yields a plausible learning curve. We present a GPU-native JAX environment for Chinese Standard Mahjong (MCR), a four-player imperfect-information game in which a win is legal only if it scores at least eight fan (pattern points) in an 81-pattern catalogue, with a fidelity contract that replays 12,288 adjudicated competition games and requires exact agreement on every judge event, per-seat win-eligibility value and final score. The reference engine that the JAX environment mirrors passes all 12,288 games on every field and the JAX environment reproduces their events and final scores (not the win-eligibility values), each with a per-game record; we also report the corpus's rule coverage. We find that the simulator is not what limits RL training: it steps up to 6.9M transitions per second on one laptop GPU, while a training iteration spends 64% of its time in the policy network. Reference workloads show the benchmark is hard to learn. From scratch, reward shaping teaches agents to reach a ready hand, but legal win rates never exceed 2.8% and end at zero. Fine-tuning a strong imitation policy leaves its win rate unchanged and raises score per game by 0.33 points (95% CI [0.05, 0.60]), an exploratory result from one training run.

1 INTRODUCTION

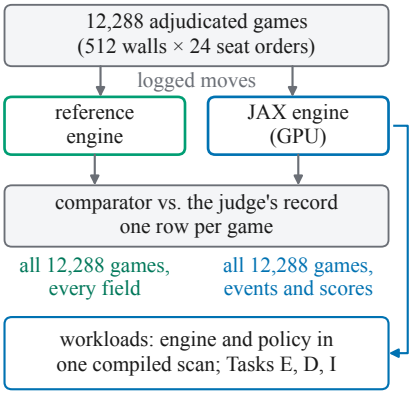
Hardware-accelerated environments have changed what an RL experiment costs. Brax, Gymnax, Jumanji, JaxMARL and Pgx compile the environment together with the agent so that thousands of episodes advance in one kernel call (Freeman et al., 2021; Lange, 2022; Bonnet et al., 2024; Rutherford et al., 2024; Koyamada et al., 2023); EnvPool attacks the same cost by batching CPU environments (Weng et al., 2022). In almost all of these the reward is defined by the environment itself, so a scoring bug is a self-consistency bug. Some of the most studied imperfect-information games do not work that way. In Mahjong, legality and payoff are fixed by an external rulebook and an official adjudicator; a simulator has to *reproduce* that rulebook, and an error in a rare branch (a mis-ordered pair of simultaneous claims, a pattern that should not stack with another) does not crash anything. It runs, trains and produces a learning curve, and the defect later surfaces disguised as an algorithmic conclusion.

The corpus we validate against contains a concrete instance. A feature layer that is widely copied in open-source MCR agents, combined with the standard open-source fan calculator, disagrees with the official judge on 118 of 12,088 winning hands, and on 20 of them it would reject a legal win as scoring below the 8-fan minimum. The calculator is correct; the fault is one flag in the layer that feeds it (the check for the last remaining copy of a tile ignores the tile just drawn). Two implementations that share such a layer agree with each other and are both wrong (Figure 2 shows one such game). Fidelity has to be measured against the adjudicator's own records, game by game, not against a second implementation.

We study Chinese Standard Mahjong (MCR), the ruleset of the official international competitions (Lu et al., 2023) and of the Botzone platform (Zhou et al., 2018). The accelerated Mahjong tooling

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

(a) Fidelity contract and training loop



(b) A contested discard in a corpus game

Seat 1 (S wind) discards 8C and two seats claim a win.

seat	concealed tiles | melds	fan	dist.
2	7C 9C 9C 9C | W W W | 1C 1C 1C | 2C 3C 4C	9	1
0	7D 8D 9D 6C 6C 8C 8C | 6D 6D 6D | 6B 7B 8B	12	3

Rule: a win outranks other claims; of competing wins, the seat nearest downstream takes it. Engine: argmax of 8·priority − distance.

source	outcome	scores of seats 0-3
judge log	seat 2 wins, 9 fan	−8 −17 +33 −8
reference engine	identical, every field	−8 −17 +33 −8
JAX engine	seat 2 wins, 9 fan	−8 −17 +33 −8

Figure 1: **The fidelity contract, and one contested discard from the corpus.** (a) Both engines are driven by the logged moves of all 12,288 games and compared with the judge's record: the reference engine on every field, the JAX engine on the events and final scores; the same JAX engine runs inside the training scan. (b) Turn 140 of the lowest-numbered corpus game with competing wins. Seats 0 and 2 can both win on the discarded 8C (12 and 9 fan); the tile goes to the nearest seat downstream, not to the larger hand, and the judge and both engines agree on the winner and all four scores. Tiles: 1–9 of characters (C), bamboo (B), dots (D); W: West wind; melds follow the bar.

that exists targets Japanese Riichi (Koyamada et al., 2022; Nishimori et al., 2026) or simplified variants (Koyamada et al., 2023; Zha et al., 2020); MCR differs from Riichi in the wall layout, the claim rules, the legality condition of a win and the entire scoring function (Section 3.1). This paper makes four contributions; its RL runs are workloads that show what the environment makes measurable, not a proposed training method, and the benchmark does not aim at the strongest agent.

1. **An MCR environment for accelerators.** A pure-functional JAX engine whose game state is a fixed-size tensor tree and whose transition is a `lax.switch` phase machine with branch-free claim resolution; only the fan calculator runs on the host, once per won game (Section 3).
2. **A validation contract with per-game evidence.** Byte-exact replay of all 12,288 adjudicated games by the reference engine under a strict comparator, replay of their events and final scores by the JAX engine, a per-game record for each engine, a coverage account of which rule surfaces the corpus exercises, and two correctness gates that a batched environment needs because it cannot raise exceptions (Section 4, Figure 1).
3. **Throughput reported by level.** Environment-only, rollout and full training rates on stated hardware, with the like-for-like comparison stated before the largest ratio (Section 5).
4. **Reference workloads.** Three benchmark tasks (fidelity, optimizer diagnosis, improvement beyond imitation), an evaluation protocol, and baseline runs that locate where the difficulty lies, including negative results reported as such (Section 6).

2 Related Work

Accelerated and batched game environments. Pgx vectorizes board games in JAX and includes a simplified Sparrow Mahjong (Koyamada et al., 2023); Jumanji and JaxMARL provide JAX suites, the latter reaching imperfect information through Hanabi (Bonnet et al., 2024; Rutherford et al., 2024; Bard et al., 2020); OpenSpiel and RLCard cover many games on CPU, RLCard with a simplified Mahjong (Lanctot et al., 2019; Zha et al., 2020); EnvPool batches CPU environments in C++ (Weng et al., 2022). Our engine follows the Pgx pattern (fixed-size state, masked phase dispatch); what is new is the rule surface it has to match and the contract used to show that it matches.

Mahjong environments and agents. Mjx is a Tenhou-compatible Riichi framework with a C++ core (Koyamada et al., 2022); Mahjax ports Riichi to JAX, validates it against downloaded Tenhou

environment	Mahjong rules	batched on accelerator	official scoring	replayed against adjudicated logs
RLCard (Zha et al., 2020)	simplified[a]	×	×[a]	×
EnvPool (Weng et al., 2022)	none	×[b]	–	–
Pgx (Koyamada et al., 2023)	Sparrow (3 players)	✓	×[c]	×
Mjx (Koyamada et al., 2022)	Riichi, Tenhou rules	×	✓	✓ Tenhou logs
Mahjax (Nishimori et al., 2026)	Riichi, Tenhou rules	✓	✓ on device	✓ Tenhou logs
Botzone judge (Zhou et al., 2018)	MCR	×[d]	✓ 81 fans	adjudicator itself
ours: reference engine	MCR	×	✓ 81 fans	✓ all 12,288 games
ours: JAX engine	MCR	✓	✓ 81 fans, host	✓ all 12,288 games[e]

Table 1: **Game environments against the properties an MCR contract needs**, each cell read from the project's paper, documentation or code. [a]Any four sets and a pair win; payoff ± 1. [b]CPU environments batched in a C++ thread pool. [c]Simplified scoring with a minimum score. [d]One game at a time. [e]Events and final scores only. Published throughputs use other games, hardware and step definitions and are not compared.

play logs following Mjx, computes hand values on device from precomputed per-suit tables, and reports up to 2M environment steps per second on eight A100 GPUs (Nishimori et al., 2026). For MCR, Botzone and the official international competitions provide an adjudicator and game logs but no vectorized simulator (Zhou et al., 2018; Lu et al., 2023); Tjong reports a strong MCR agent without an environment release (Li et al., 2024). On the agent side, Suphx combines deep RL with global reward prediction, oracle guiding and run-time adaptation for Riichi (Li et al., 2020), and Actor-Critic Hedge replaces the return objective with a weighted counterfactual-regret objective for 1-on-1 Mahjong (Fu et al., 2022). Our workloads use none of these components; they measure how far a standard fine-tuning recipe gets inside the environment. Table 1 sets the environments side by side on the four properties the contract depends on.

Fine-tuning imitation policies with RL. Fine-tuning a cloned or distilled policy (Pomerleau, 1991) with policy gradients and a KL penalty to the reference policy is standard in games and in learning from human preferences (Ziegler et al., 2019). IRPO shows that with imperfect demonstrations such KL regularization can bias and limit online improvement, and alternates imitation and RL phases to lift it (Gong et al., 2024); our fine-tuning workload is the plain KL-regularized regime IRPO starts from. PopArt rescales value targets when returns span a wide range (van Hasselt et al., 2016); one workload separates its effect from that of extra value-fitting steps.

3 THE ENVIRONMENT

The game. MCR is played by four seats with 136 tiles (34 kinds, four copies each, no flowers). Each seat owns a private 34-tile wall segment and draws from its back; kong replacements come from the drawer's own segment, and the game is drawn when the seat due to draw has an empty segment. On a turn a seat draws and either declares a win, declares a concealed or added kong, or discards. The other seats may claim the discard: Hu (win) outranks Peng (triplet) and melded Gang (kong), which outrank Chi (sequence, only from the left neighbour); among competing wins the seat nearest downstream of the discarder takes it (Figure 1b). A promoted kong can be robbed by a win. A hand wins only if it forms four melds and a pair (or a special shape) *and* scores at least 8 fan under the official catalogue of 81 patterns, which compose and exclude one another. A self-drawn win with fan total f receives $3(8+f)$, each other seat paying $8+f$; a win on a discard receives $24+f$, the discarder paying $8+f$ and each bystander 8; a drawn game pays nothing.

State and transition. The state is a fixed-size tree of small integer tensors (wall, per-seat draw pointers, 4×34 hand counts, discards, up to four melds per seat, phase, seat to act, prevalent wind, terminal fields). `step` is a pure function dispatched by `lax.switch` over five phases: *draw*, *discard-and-claim* (the three other seats choose simultaneously), *kong replacement*, *added kong* (which the others may rob) and an identity *terminal* phase, so that finished games in a batch are no-ops. Claim resolution is one `argmax` of $8 \cdot \text{priority} - \text{distance}$ over the claimants, and a granted Peng or Chi carries the claimer's follow-up discard in the same step, as the adjudicator records it.

Scoring as a host callback. Legality and payoff of a win come from the open-source MCR fan calculator PyMahjongGB, which matches the judge on all 12,088 winning hands of the corpus once the feature-layer flag above is correct. It is called through `jax.pure_callback` only when a win is declared, so the rest of the rollout stays on the device. We did not tensorize the catalogue. The obstacle is not search (over the 5,921 winning hands of 6,000 corpus games the four-melds-plus-pair form admits at most 2 decompositions) but the median 4 and up to 8 patterns that fire per hand under data-dependent exclusion rules. Mahjax moves Riichi hand evaluation onto the device with precomputed per-suit tables (Nishimori et al., 2026), and an analogous table for MCR may be possible; the callback costs 0.047% of rollout time (Section 5), so there is little throughput to gain.

Interface. A seat observes a $38 \times 4 \times 9$ tensor (the two winds, its hand in four count planes, and all discard histories and exposed melds, relative to the observer). It chooses among 235 actions (pass, win, 34 discards, 63 Chi, 34 Peng, and 34 each of melded, concealed and added Gang) under a Boolean legality mask. The reward is the terminal, zero-sum game score divided by eight.

Two correctness gates. A batched environment has no exceptions to catch; an illegal transition silently corrupts part of the batch. (1) *Illegal-win forfeit*: the on-device mask admits any structurally complete hand, so a declared win below 8 fan is forfeited (the declarer pays 24, the others receive 8) instead of paid; otherwise an agent can collect the payout of a zero-fan "win". (2) *Meld cap*: claims are masked once a seat holds four melds, since a fifth would write out of bounds. Both have regression tests, and neither affects replay, since adjudicated games contain neither case.

3.1 Why not adapt a Riichi engine?

Mjx and Mahjax share MCR's draw–discard–claim loop, and we reuse their design pattern (fixed-size state, masked phase dispatch, legality masks); the rules that differ touch every phase branch and the whole scorer (Appendix A). MCR's adjudicated wall is four private 34-tile segments read from the back, with kong replacements from the drawer's own segment, instead of one shared wall with a dead wall and dora indicators. Competing wins have a single winner (Tenhou's rules allow two), and there is no furiten restriction. A win needs a total of 8 fan rather than one scoring pattern, and the scorer sums 81 patterns with non-stacking rules into the fixed-base payment above instead of han and fu with dora and riichi deposits; MCR has no riichi declaration and no multi-hand rounds. Adapting a Riichi engine would replace its wall indexing, claim resolution, win test and scorer, leaving the design pattern, not the code. The validation does not transfer at all: Tenhou logs cannot certify MCR behaviour, so the contract below is what another MCR implementation must satisfy.

4 The validation contract

Corpus. We use a public engine test set built from the complete logs of the 12,288 games of the final stage of the IJCAI-2026 MCR competition, played on Botzone by four finalist programs and adjudicated by the platform's official judge: 512 walls, each under 24 seat and wind permutations, with no protocol errors. Every record holds the full wall, the prevalent wind, every request the judge sent to each seat, every public judge event, the four responses, and the adjudicated ending, fan list and scores. Strong, mutually adversarial programs exercise claim conflicts and kong edge cases far more often than random play.

Comparator. An engine under test is driven by the logged responses turn by turn and must reproduce the judge's stream exactly. That stream is every per-seat request string and every event (claims granted, kong type, win detection). Each event carries the per-seat win-eligibility field (the exact fan a seat could win with at that event, or a code for "cannot win") and the remaining-wall counts; the terminal fan total and four scores close the stream. We call agreement on all of these fields for every turn of a game *byte-exact*. The ground truth is the adjudicator's record, not a second engine.

Results. The reference engine, a pure-Python implementation of the transition function that the JAX engine mirrors field for field, reproduces 12,288/12,288 games under the strict comparator. The JAX engine reproduces 12,288/12,288 games under an event-and-score comparator: per event the action, seat and tile (and the Chi tile), and at the end the ending, winner, fan total and four scores (a drawn game, which pays nothing, has no score check). The supplement lists the pass/fail status of every game for both engines. The JAX comparator omits the per-seat win eligibility, the wall counts, the request strings and the fan names; on those fields its agreement is inferred from its construction, not

event class	full	golden	event class	full	golden
win on a discard	8,436	109	promoted (added) kong	1,204	54
self-drawn win	3,652	92	melded kong	939	25
drawn game (wall exhausted)	200	20	concealed kong	519	23
win at exactly 8 fan	1,778	36	robbed kong	28	28
fan total $\geq$ 48 (max 93)	20	10	win on a kong replacement	46	8
$\geq$ 2 seats claim one discard	1,622	57	win on the last wall tile	9	5
$\geq$ 2 competing wins on one tile	140	20	win on the last discard	4	4

Table 2: **Coverage of the replay corpus** (12,288 games; golden subset 221 games; a game carries every class it contains). Counts of games, from the test set's per-game tags.

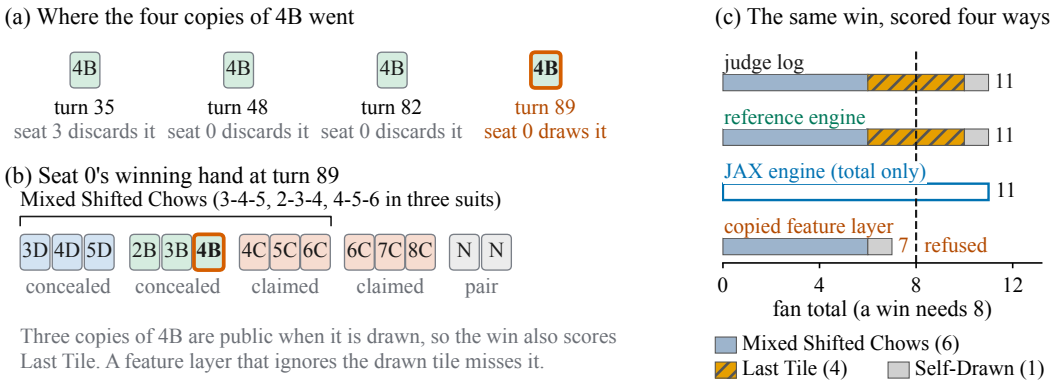


Figure 2: **A legal win that the widely copied feature layer refuses**: the lowest-numbered of the 20 such corpus games. (a) Three copies of 4B are discarded (turns 35, 48, 82; MCR lets a seat win on a tile it has discarded) before seat 0 draws the last one. (b) The winning hand, grouped as the judge scores it. (c) The same win in one view per scorer: the judge log, the reference engine (every field, including the win eligibility of 11 at the draw) and the JAX engine (terminal fan and scores) give 11; the feature layer with the open-source calculator gives 7.

measured. The golden subset of 221 games is selected deterministically to contain every rare event class of Table 2 (e.g., all 28 robbed-kong wins and all 20 games on which the widely copied feature layer rejects a legal win); on it both engines were also compared side by side, one row per game (221/221 strict, 221/221 event-and-score passes).

Coverage and its limits. Table 2 reports how often the corpus exercises the parts of the rule surface that engine implementers worry about: claim conflicts in 1,622 games, competing wins on one tile in 140, 227,920 Chi, Peng and kong events, 1,778 wins at the exact legal minimum and fan totals up to 93. The limits matter as much. The records list the adjudicated fans, but our released tabulation keeps only fan totals and the four fan tags of Table 2, so we do not report how many of the 81 patterns the corpus fires. A pattern that never occurs is untested by replay, a risk that the calculator's agreement with the judge on every corpus win bounds but does not remove. The rarest terminal situations have single-digit counts (4 wins on the last discard, 9 on the last wall tile), and 512 walls played by four programs under-sample the rule paths those programs avoid. Replay cannot test transitions that competent play never produces; the two correctness gates cover those.

A case from the subset. Figure 2 follows one of the games on which the feature layer of Section 1 and the judge disagree. Seat 0 wins by drawing the last copy of a tile whose other three copies are public, so the judge adds Last Tile (4 fan) to Mixed Shifted Chows (6) and Self-Drawn (1) and pays the winner +57. The feature layer counts only the copies on the table, so Last Tile cannot fire on a self-drawn win and the total drops to 7: an agent gated by it declines the win, and a simulator scored by it treats the win as illegal. The same cause explains all 118 disagreements, and both of our engines agree with the judge on this game.

A portable fidelity package. Another MCR implementation satisfies the contract if it (i) fixes the corpus and ruleset version, (ii) emits one row per replayed game with pass/fail and the first

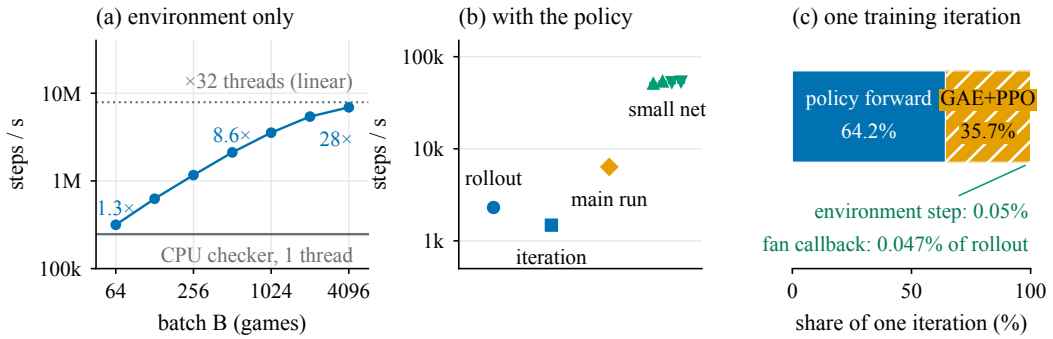


Figure 3: **Throughput at each level** (steps per second, log scale). (a) The JAX engine alone on a laptop GPU, one timing per batch, against the single-threaded CPU replay checker (median and range of three repeats, 246.0k–247.3k) and its linear extrapolation to 32 threads; labels are ratios. (b) With the 40-block policy: rollout and full iteration from one profile (laptop GPU, fp32, B=512); the main fine-tuning run (desktop GPU, bf16, B=256; median 6,358, 5–95% range of its logged intervals 6,124–6,378); the small network trained from scratch (four runs). (c) Time shares of one training iteration, and of the fan callback within the rollout.

mismatching field, (iii) reports the per-class results of Table 2, and (iv) passes the two gate tests. The supplement contains the comparator, the per-game records and the tests, so a reader can check any single game rather than the total.

5 Throughput, reported by level

Figure 3 separates the levels that are often merged, and Table 5 (Appendix B) lists the hardware, precision and batch of each. The environment alone scales from 316k to 6.91M transitions per second between batch 64 and 4096. A training iteration at batch 512 runs at 1,482 steps per second, and its time splits into the policy forward pass (64.2%), GAE and the PPO update (35.7%), and the environment step (0.05%). Training is therefore bound by the policy network, and the multi-million figure describes the simulator, not the training rate. The small network used from scratch trains at 50.7k–54.0k steps per second.

What the CPU comparison shows. The only CPU baseline we measured is the replay checker that accompanies the test set, single-threaded on the same machine: 247k judge events per second (2,691 games per second). At the batch used in training (512) the JAX environment is 8.6× faster, from 1.3× at batch 64 to 28× at batch 4096. Three caveats bound these ratios. The checker validates while it steps, so it understates CPU speed. Logged games average 91.8 judge events, whereas no timed game ended within 120 engine transitions, so a per-game ratio would be smaller. And linear scaling to all 32 CPU threads would give about 7.9M events per second, the same order as the GPU environment; we did not measure an EnvPool-style multi-process CPU environment. The GPU engine's case is thus not raw speed but that the environment runs inside the same compiled scan as the policy, so a rollout needs no per-step transfer between host and device. Only the fan callback visits the host (85% of profiled games, 0.044 ms per call, 0.047% of rollout time).

6 Reference workloads

6.1 Benchmark tasks and evaluation protocol

We grade three tasks separately, because an environment can be exact and still teach nothing, and an optimizer can be repaired without the agent becoming stronger. **Task E (fidelity)** is the contract of Section 4, with throughput reported by level. **Task D (optimizer diagnosis)** asks whether changing one training component fixes an optimization failure, measured by a training diagnostic. **Task I (improvement beyond imitation)** asks whether a learned policy beats a frozen imitation anchor.

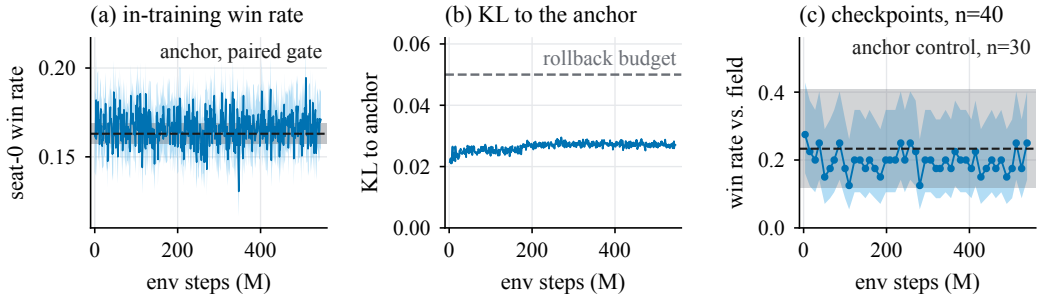


Figure 4: **The main fine-tuning run (one training seed).** (a) Seat-0 win rate in in-training evaluations of 1,000–2,000 games, with Wilson 95% intervals; dashed line and grey band: the anchor's win rate and interval in the paired gate. (b) KL to the anchor on the evaluation games (the log keeps only its mean), below the rollback budget. (c) Win rate of 45 checkpoints against a three-anchor field (n=40 each, Wilson 95%), and one anchor control of n=30 games reused for every checkpoint.

Evaluation. A candidate plays seat 0 against three copies of a reference opponent on seeded walls with greedy masked actions; the control is the reference policy itself in seat 0 on the same walls, so each wall yields a paired game. We report win rate and mean game score (in points; a win is worth at least 32) with the number of games; a ladder (random legal play, a greedy tile-efficiency heuristic, the anchor, a multi-network ensemble) gives an absolute scale.

Exploratory and confirmatory claims. A checkpoint curve is exploratory. A positive Task I claim must fix the checkpoint rule and the paired statistic in advance, evaluate at least three independently trained seeds on common wall blocks not used for selection, with at least 1,000 paired games per seed, and retain one row per game. The fine-tuning result below meets the game-count part of this rule but not the seed part, and we label it accordingly.

6.2 Training setup

The imitation anchor is a strong supervised policy distilled from expert play: a 40-block residual network with batch normalization removed. The learner starts from it in seat 0; seats 1–3 are frozen copies of the anchor. A separate three-layer convolutional critic is trained from scratch. Rollouts use B fresh games of T=256 steps; only seat-0 decisions with more than one legal action enter the buffer (at most 20,480 per update). The reward is the terminal game score of seat 0 divided by 8. Returns are undiscounted (γ=1) and advantages use GAE with λ=0.95 (Schulman et al., 2016) over seat-0 decisions; a game still running at T receives zero terminal reward without bootstrapping (in the profiled rollout 0 of 512 games were unfinished at T). Advantages are standardized per update. Per minibatch the loss is

$$\mathcal{L} = c_v \tfrac{1}{2}\big(\hat{v}_\phi(s) - \tfrac{\hat{R}-\mu}{\sigma}\big)^2 + \beta\,\mathrm{KL}\big(\pi_\theta(\cdot|s)\,\|\,\pi_{\mathrm{ref}}(\cdot|s)\big) + \mathbb{1}[\overline{\mathrm{KL}}_{\mathrm{mb}} < 0.7\,\kappa]\,\big(\mathcal{L}_\epsilon^{\mathrm{clip}} - c_H\,\mathcal{H}(\pi_\theta)\big),$$

averaged over the valid decisions, with the PPO clipped surrogate $\mathcal{L}_\epsilon^{\mathrm{clip}}$ (ϵ=0.2; Schulman et al., 2017), c_v=0.5, c_H=10^{-3} and KL budget κ=0.05; 3 epochs of 1,024-sample minibatches are followed by 6 value-only epochs with the policy frozen. μ, σ are exponential moving averages (decay 0.99) of the return targets (PopArt-style normalization, van Hasselt et al., 2016); the critic's output is denormalized, $\hat{v}\sigma + \mu$, wherever it enters GAE, and with normalization off μ=0, σ=1. After each update the KL is averaged over all buffer decisions; above κ the policy step is rejected and β doubles (trust-region rollback; Schulman et al., 2015), otherwise β adapts (Appendix B gives the schedule and learning rates). The main run uses B=256, bf16, normalization and value-only epochs; it logged 500 evaluation records (4 of 1,000 games, 496 of 2,000) over 545.6M steps in 30.6 h (Appendix B). Its evaluation games use greedy actions on fresh walls, and the KL it logs is measured on those games; the per-update rollback decisions are not in the released log.

6.3 Optimizer diagnosis

The KL budget is not reached. In the main run the seat-0 win rate stays flat at a mean of 0.166 (range 0.131–0.195) over 545.6M steps, while the evaluation KL to the anchor averages 0.026 and

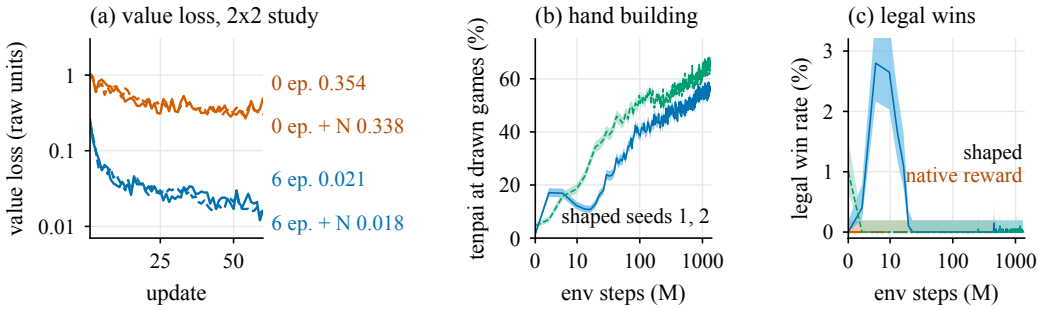


Figure 5: **Optimizer diagnosis and training from scratch.** (a) Value loss of the 2×2 study (ep.: value-only epochs; N: target normalization), one seed, raw units; labels give the last-10 mean. (b,c) From-scratch runs evaluated with shaping off (2,000 games per evaluation, Wilson 95% intervals): tenpai rate at drawn games of the two shaped seeds, and legal win rate of the shaped (blue, green) and native-reward (red) seeds. Steps are linear to 10M and logarithmic above; the point at zero in (c) is shaped seed 2 before training.

per-game quantity	fine-tuned	anchor	difference	95% CI of the difference
win rate	0.1657	0.1630	+0.0027	[-0.0014, +0.0068] (paired)
mean score (points)	0.713	0.384	+0.328	[+0.053, +0.604] (paired)
self-drawn win rate	0.0529	0.0483	+0.0046	[-0.0004, +0.0096] (unpaired)
win-on-discard rate	0.1128	0.1147	-0.0019	[-0.0091, +0.0053] (unpaired)
deal-in rate	0.0983	0.1021	-0.0038	[-0.0106, +0.0030] (unpaired)

Table 3: **Paired gate: the last checkpoint of the single training run against its anchor** (15,000 games, three disjoint blocks of 5,000 walls, seat 0, greedy play). Paired intervals use the per-game differences; the component rates exist only as aggregates, so their intervals treat the arms as independent, likely conservatively (win-rate half-width: unpaired 0.0084, paired 0.0041).

never exceeds 0.0295, below the 0.05 budget (Figure 4a,b). Fixed 50-record windows at the start, middle and end give win rates of 0.166, 0.163 and 0.167 and deal-in rates of 0.105, 0.105 and 0.104, while mean score moves 0.52, 0.42, 0.63. The hard KL threshold therefore does not bind. This does not show that the adaptive penalty is harmless: it may still restrain each update, which only a penalty or budget ablation, not run here, can test.

Value epochs, not target normalization, account for the critic fit. A critic that cannot fit high-variance terminal returns gives noisy advantages, and normalizing the value target is the usual remedy. To separate it from the number of value-fitting steps we ran a same-seed, 60-update 2×2 study (Figure 5a), reporting $\frac{1}{2}(\hat{v}\sigma + \mu - \hat{R})^2$ of the final critic on its update buffer in raw reward units (score/8) for all four arms, so normalized and unnormalized arms share a scale. Six value-only epochs lower the last-10-update mean from 0.354 to 0.021 (about $17\times$); normalization alone moves it from 0.354 to 0.338, and from 0.021 to 0.018 with the value epochs. With one seed, 60 updates and a training-set loss, this shows that reporting only a normalized run would confound target scaling with extra optimization, not that normalization is ineffective or that the better fit improves play.

6.4 Improvement beyond imitation

A checkpoint curve cannot decide the sign. Over 45 checkpoints from 3.6M to 539.1M steps the fine-tuned policy averages a win rate of 0.196 and a score of -2.12 against the field, below the anchor control's 0.233 and +1.53 (Figure 4c), and it beats the tile-efficiency heuristic (0.66) and random play (0.72) comfortably. These numbers are too noisy to carry a claim: at n=40 a win rate of 0.20 has a Wilson 95% interval of about [0.10, 0.35], the control is a single reused n=30 estimate, and the checkpoints lie on one correlated trajectory, so they cannot be pooled. We keep the curve to show why the protocol demands a paired gate.

The paired gate. We therefore evaluated the last checkpoint of the run (545.7M steps, not selected by any evaluation) against the anchor on 15,000 walls in three disjoint blocks of 5,000, the candidate and the anchor each playing seat 0 against three anchors on the same walls (Table 3). The two are tied on win rate (+0.0027, 95% CI [-0.0014, +0.0068]), and the fine-tuned policy scores +0.328 points per game more (95% CI [+0.053, +0.604]; p=0.020, or 0.039 with a Bonferroni correction for the two primary quantities; the component rates are descriptive). The paired games differ on 3,527 walls and are identical on the rest. The score difference is positive in each block (+0.24, +0.52, +0.23). This reverses what the n=40 curve suggested: small checkpoint evaluations with a reused control can get the sign wrong.

Scope. All 15,000 games evaluate one trained policy: the three blocks are evaluation replicates, not training seeds. Under the protocol above, which asks for at least three independently trained seeds, the score gain is therefore *exploratory*. It shows that this run ended slightly above its anchor on score, not that the recipe reliably improves on imitation. A second Task-I attempt, a short run against a growing buffer of past checkpoints plus the anchor (9.9M steps), is too short to be informative: its 5 evaluations give win rates of 0.14–0.20 against an anchor control of 0.16 (100 games each).

Where the score difference comes from. The gate recorded per-arm aggregate rates but not per-game fan totals, so the score difference cannot be decomposed exactly, and the component rates in Table 3 are individually unresolved. Their directions (more self-drawn wins, fewer wins on a discard, fewer deal-ins) are each consistent with a higher score at an unchanged win rate under MCR's payoffs, where a self-drawn win collects $3(8 + f)$ instead of $24 + f$ and dealing in costs $8 + f$ instead of 8, but larger hands are not excluded.

From scratch. Without the anchor, a small network (8 blocks, 64 channels) trained on the native reward reached 524M steps in each of two seeds with a legal win rate of exactly zero in all 168 and 167 evaluations (2,000 games each). With dense shaping by distance to a ready hand at drawn games, evaluated with shaping off so that scores are true MCR values, two seeds learn hand construction: mean distance to a ready hand falls from 2.40 to 0.58 and from 2.24 to 0.48, and the tenpai rate reaches 57% and 64% (Figure 5b). They do not learn to win legally: over 1.31B steps per seed a nonzero legal win rate appears in 23/416 and 20/417 evaluations, the highest is 2.8%, and both seeds' final evaluations are zero (Figure 5c). One nonzero value (0.01, seed 2) occurs at update 0, before training, so an isolated nonzero evaluation is not evidence of learning. Reaching a ready hand is learnable here; turning it into an 8-fan win is not, within this budget.

7 Discussion and limitations

What the environment makes measurable. The workloads separate questions that one curve merges: the optimizer can be repaired without the agent improving, the KL threshold does not bind, and a policy can tie its anchor on win rate while differing on score. Each would be uninterpretable with a wrong scorer or claim logic, which is what the contract tests.

Limitations. The JAX engine's full-corpus record covers events and final scores only; the record on every field (win eligibility, wall counts, request strings, fan names) belongs to the reference engine it mirrors, so on those fields the workloads, which run on the JAX engine, rely on that mirroring. The imitation anchor's weights are not in the anonymous supplement, so others cannot rerun Task I against the same anchor. Per-category fan coverage is unknown, and rare situations have few examples. The only CPU speed baseline is a single-threaded validating checker, and all timings come from one machine per row. The fine-tuning evidence is one game, one algorithm family and one training seed, evaluated only from seat 0 against frozen anchors; the value-loss study is a 60-update diagnostic; the from-scratch runs use two seeds per condition and one network size. Nothing here licenses conclusions about other Mahjong rules, search-based agents, or the strength attainable by richer training systems such as those of Li et al. (2020) or Fu et al. (2022).

Conclusion. In this work, we built an accelerator-native MCR environment and a contract that ties fidelity to the adjudicator's records game by game, with its coverage stated. The reference workloads show a benchmark that is checkable against its adjudicator and hard to learn: from scratch, legal wins stay sporadic (at most 2.8% per evaluation) and end at zero, and fine-tuning an imitation policy for more than half a billion steps leaves its win rate unchanged.

REPRODUCIBILITY STATEMENT

Section 6.2 and Appendix B give every training setting of the reference workloads, and Sections 4 and 5 give the validation and timing protocols. The anonymous supplementary material contains the JAX environment, the reference engine, the replay comparator and the correctness tests. It also contains the training code for the fine-tuning and from-scratch workloads; per-game replay records of all 12,288 games for both engines (the reference engine under the strict comparator, the JAX engine under the event-and-score comparator, with the test script's log of its full replay), per-game records of both engines on the golden subset, and the corpus tag counts; the raw per-update or per-evaluation logs of every run reported here; the timing records; and one script that recomputes every number in the paper from these files and one that redraws the figures. The replay corpus itself is a public test set and is not redistributed; the fine-tuned and anchor network weights are not included. The strict comparator also compares fan names: version 1.4.0 of the open-source fan calculator renames three one-fan patterns, which fails 36 golden games on the name alone, so the supplement pins an earlier version for the reference engine. The JAX comparator compares fan totals, which the renaming leaves unchanged; the full JAX replay ran under version 1.4.0 with JAX 0.4.35 on one CPU core in about 17.3 minutes.

AI USE STATEMENT

We used generative AI tools on the following tasks from ICLR's required-disclosure list: implementing methods (parts of the environment, training, evaluation and analysis code), giving feedback on experimental design, and helping to interpret results. We did *not* use generative AI tools to generate synthetic data reported as measurements, to clean or reformat data sets reported as results, for translation, or for qualitative or thematic data analysis. From the recommended-disclosure list we additionally used generative AI tools to draft and edit parts of this paper, to create and edit figures, to identify and summarise related literature, to format references, and to suggest a structure for the paper.

We have reviewed all AI-assisted work. The environment's fidelity was checked against the adjudicator's records by the replay comparator rather than by inspection; every number reported was regenerated by the released analysis script from the logged outputs and compared with them; and every citation was checked against its publisher, proceedings or arXiv record. We take responsibility for the final content of this work, including all text, claims and artifacts produced with the aid of generative AI.

REFERENCES

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi challenge: A new frontier for AI research. *Artificial Intelligence*, 280:103216, 2020.

Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: A diverse suite of scalable reinforcement learning environments in JAX. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

Haobo Fu, Weiming Liu, Shuang Wu, Yijia Wang, Tao Yang, Kai Li, Junliang Xing, Bin Li, Bo Ma, Qiang Fu, and Wei Yang. Actor-critic policy optimization in a large-scale imperfect-information game. In *International Conference on Learning Representations (ICLR)*, 2022.

Xudong Gong, Dawei Feng, Kele Xu, Yuanzhao Zhai, Chengkang Yao, Weijia Wang, Bo Ding, and Huaimin Wang. Iterative regularized policy optimization with imperfect demonstrations. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 55547–55568, 2024.

Sotetsu Koyamada, Keigo Habara, Nao Goto, Shinri Okano, Soichiro Nishimori, and Shin Ishii. Mjx: A framework for mahjong AI research. In *2022 IEEE Conference on Games (CoG)*, pp. 504–507, 2022.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, et al. Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.

Robert Tjarko Lange. gymnax: A jax-based reinforcement learning environment library. `https://github.com/RobertTLange/gymnax`, 2022.

Junjie Li, Sotetsu Koyamada, Qiwei Ye, Guoqing Liu, Chao Wang, Ruihan Yang, Li Zhao, Tao Qin, Tie-Yan Liu, and Hsiao-Wuen Hon. Suphx: Mastering mahjong with deep reinforcement learning. *arXiv preprint arXiv:2003.13590*, 2020.

Xiali Li, Bo Liu, Zhi Wei, Zhaoqi Wang, and Licheng Wu. Tjong: A transformer-based mahjong AI via hierarchical decision-making and fan backward. *CAAI Transactions on Intelligence Technology*, 9(4):982–995, 2024. doi: 10.1049/cit2.12298.

Yunlong Lu, Wenxin Li, and Wenlong Li. Official international mahjong: A new playground for AI research. *Algorithms*, 16(5):235, 2023.

Soichiro Nishimori, Shinri Okano, Keigo Habara, Sotetsu Koyamada, Eason Yu, and Masashi Sugiyama. Mahjax: A gpu-accelerated mahjong simulator for reinforcement learning in JAX. *arXiv preprint arXiv:2605.20577*, 2026.

Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob Nicolaus Foerster. JaxMARL: Multi-agent RL environments and algorithms in JAX. In *Advances in Neural Information Processing Systems 37: Datasets and Benchmarks Track*, 2024.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

Jiayi Weng, Min Lin, Shengyi Huang, Bo Liu, Denys Makoviichuk, Viktor Makoviychuk, Zichen Liu, Yufan Song, Ting Luo, Yukun Jiang, Zhongwen Xu, and Shuicheng Yan. EnvPool: A highly parallel reinforcement learning environment execution engine. In *Advances in Neural Information Processing Systems 35: Datasets and Benchmarks Track*, pp. 22409–22421, 2022.

Daochen Zha, Kwei-Herng Lai, Songyi Huang, Yuanpu Cao, Keerthana Reddy, Juan Vargas, Alex Nguyen, Ruzhe Wei, Junyu Guo, and Xia Hu. Rlcard: A platform for reinforcement learning in card games. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI), Demonstrations Track*, pp. 5264–5266, 2020.

Haoyu Zhou, Haifeng Zhang, Yushan Zhou, Xinchao Wang, and Wenxin Li. Botzone: An online multi-agent competitive platform for AI education. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, pp. 33–38, 2018.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A MCR VERSUS RIICHI: RULE DIFFERENCES THAT REACH THE CODE

component	Riichi (Tenhou rules, as in Mjx and Mahjax)	MCR (Botzone adjudicated format, this work)
wall and draws	one shared wall, dead wall with dora indicators and kong replacement tiles	four private 34-tile segments read from the back; kong replacement from the drawer's segment; draw when the next drawer's segment is empty
tiles	red fives in the red-rule variant	no red tiles, no flowers
claims	Chi from the left, Pon/Kan over Chi; double ron allowed; furiten restricts wins on discards	Chi from the left, Peng/Gang over Chi; one winner, nearest downstream of the discarder; no furiten
win condition	at least one yaku	at least 8 fan in total
scoring	han and fu, dora, riichi deposits, continuation counters	sum of 81 fan patterns with non-stacking rules; fixed-base payment: $3(8+f)$ self-drawn, $24+f$ on a discard
declarations	riichi	none
game structure	multi-hand match (East–South rounds)	independent single-hand games
validation logs	Tenhou play logs	Botzone judge logs (this work)

Table 4: Every row changes a phase branch of the transition function or the scorer; only the draw–discard–claim skeleton and the design pattern are shared.

B TRAINING ACCOUNTING AND EVALUATION STREAMS

level	hardware	precision	batch / horizon	steps/s
environment only (no policy)	RTX 3060 Laptop	integer	B=64, T=120	316k
			B=512, T=120	2.12M
			B=4096, T=120	6.91M
replay checker (no policy)	same, 1 CPU thread	integer	sequential	247k
rollout: env. + 4-seat policy	RTX 3060 Laptop	fp32	B=512, T=256	2,304
full iteration (rollout, GAE, PPO)	RTX 3060 Laptop	fp32	B=512, T=256	1,482
training, main fine-tuning run	desktop GPU	bf16	B=256, T=256	6,358
training, small net from scratch	desktop GPU	fp32	B=512, T=256	50.7k–54.0k

Table 5: **Throughput at each level.** One step is one phase transition of the engine, or one logged judge event for the replay checker. The rollout and full-iteration rows come from one profiling run; the main-run row is the median over its log; the policy is a 40-block residual network except in the last row (8 blocks, 64 channels).

Training settings. Fine-tuning uses learning rates of 3×10^{-5} (policy) and 3×10^{-4} (critic) with gradient-norm clip 1, and after an accepted update β is multiplied by 1.3 when the buffer KL exceeds $\kappa/2$ and divided by 1.2 when it is below 0.3κ. The from-scratch workloads use the same trainer and

main-run accounting	value	evaluation stream	games/eval	records
training seeds	1	in-training diagnostics	1,000–2,000	500
PPO updates	8,325	checkpoint vs. three-anchor field	40	45
logged evaluation records	500	anchor self-control	30	reused
final training step	545.6M	random / heuristic ladder	30	45
last checkpoint evaluation	539.1M	multi-network ensemble	6	45
median training throughput	6,358 steps/s	paired gate (last checkpoint)	5,000 per block	3 blocks
elapsed time incl. evaluation	30.6 h			

Table 6: Accounting for the released main-run logs. Checkpoint records sample one trajectory densely; they are not independent training seeds. The six-game ensemble check is ladder context only.

loss with the KL term disabled (β=0, no budget), policy learning rate 10^{-4}, entropy coefficient 0.01, B=512, T=256, evaluation every 25 updates on 2,000 games with shaping off, plus one evaluation whenever a run resumed from a checkpoint (hence 416 and 417 shaped and 168 and 167 native evaluations). Native reward: 4,000 updates per seed. Shaped: 10,000 updates per seed; at a drawn game the seat receives $\mathrm{clip}\big(0.5\,(6-d)/6, \pm 0.5\big)$ for shanten distance d, computed once per game on the host, so the environment's dynamics are unchanged. The value-loss study uses the same trainer for 60 updates from the same seed, with normalization on or off and 0 or 6 value-only epochs, all else fixed.

Statistics. Paired intervals in Table 3 are those computed by the evaluator from per-game differences. Unpaired intervals are Wald intervals for a difference of two proportions with 15,000 games per arm. The Wilson interval is for 8 wins in 40 games. The p-value of the paired score difference is recovered from its interval as z=2.33. All are recomputed by the released script.