Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Move Quality Beyond Arena: Pure Distillation for Small Go Networks

Taiki Kojima

koji3744@hotmail.com

Kojima Family Clinic https://orcid.org/0000-0003-2069-6079

Research Article

Keywords: computer Go, knowledge distillation, KataGo, Monte Carlo Tree Search, evaluation methodology, small models

Posted Date: May 19th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9749016/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Move Quality Beyond Arena: Pure Distillation for Small Go Networks

Taiki Kojima[1]

[1]Kojima Family Clinic

May 18, 2026

Abstract

The AlphaZero paradigm dominates computer Go research but is poorly suited to the compute budget available to individual researchers, where 7M-parameter networks trained on a single GPU reliably plateau after a small number of self-play generations. We study an alternative: pure knowledge distillation from a stronger existing engine (KataGo b18c384) into a compact 7M-parameter student, using positions sampled from human (KGS) and KataGo self-play games with no self-play-generated labels. We make four contributions. (1) Pipeline: a fully decoupled distillation pipeline (positions → KataGo labeling → supervised training) that can ingest positions of arbitrary provenance. (2) Move Quality vs. Oracle: an evaluation methodology that compares the student's raw policy distribution directly to a high-visit KataGo analysis on a xed 100-position evaluation set, and that shows a consistent monotonic trend in generation-over-generation improvement (top-1 oracle agreement 0% → 2% → 10% → 16%) in cases where head-to-head arena reports indistinguishable ≈ 0.50 winrates. (3) Label-depth dose-response: increasing KataGo labeling visits from 100 to 4000 yields +60% relative improvement in top-1 oracle agreement while value MAE is essentially unchangeda policy/value asymmetry with implications for label-budget allocation. (4) MCTS optimization: a stack of legal-move, tree-reuse, leaf-batching, and time-control optimizations delivers a 3.4× end-to-end speedup, yielding ∼12,000 simulations within a 30-second per-move budget on consumer hardware. We discuss how this paradigm enables AI-interpreted style transfer for small Go networks, analogous to recent LLM persona work.

Keywords: computer Go, knowledge distillation, KataGo, Monte Carlo Tree Search, evaluation methodology, small models.

1 Introduction

The dominant paradigm in modern computer Go is AlphaZero-style reinforcement learning, in which a network is trained purely on self-play games with MCTS-guided policy targets and game-outcome value targets [7, 8]. This paradigm has been spectacularly successful at the scale of large research labs: AlphaZero, KataGo [10], LeelaZero, and similar projects have produced networks far surpassing human professionals.

Replicating this success at smaller scale, however, has proven surprisingly dicult. Individual researchers working with networks on the order of 7M parameters routinely report plateau behavior after a small number of self-play generations. In our own experience reproducing this paradigm

with a 7M-parameter residual network on 19 × 19 Go, we observed a sequence of 15+ self-play generations in which each generation showed only marginal gains (arena winrates clustered around 0.50, validation losses moving by ∼ 0.1). Because the training data of generation N + 1 is generated by the network at generation N, the data is bounded in quality by the network's current strength. Without sucient compute to compensate through volume, the system saturates well below any meaningful upper bound.

A second observation drove the present study. Despite arena winrates suggesting no progress, qualitative inspection by a strong amateur Go player through the Sabaki GUI consistently rated newer generations as having better shape and playing in a more KataGo-like style. This dissociation between arena metrics and human-judged quality led us to investigate what arena was actually measuringand what it was missing.

A third motivation comes from a broader trend reshaping machine learning practice. In natural language processing, large language models are routinely used as teachers to generate training data for smaller, locally deployable students [1, 9]. A locally ne-tuned model trained on GPT-rendered text in the style of an author does not learn the original author; it learns the teacher's interpretation of that author, ltered through the teacher's understanding. We transpose the same pattern into game AI: KataGo, a publicly available strong Go engine, serves as the teacher; the resulting small Go network is, in the same sense, a KataGo-rendered model of whatever positions it is shown. We return to this analogy in Section 6.

Our approach. We abandon self-play label generation. Positions come from any sourceKGS human games, KataGo-vs-KataGo self-play, or legacy archivesand are labeled by running KataGo b18c384 analysis at a chosen MCTS visit depth. The student network learns to reproduce KataGo's policy distribution, value, and ownership predictions. Position diversity matters for coverage, but provenance does not aect what is learned: the student becomes a small-form KataGo, bounded above by the teacher but freed from the bootstrap-from-weakness problem of self-play.

Contributions.

- Pipeline. An end-to-end pure-distillation pipeline for compact Go networks (Section 3, Figure 1).
- Move Quality vs. Oracle. An evaluation methodology that compares raw student policy against a high-visit KataGo reference on a xed evaluation set and shows a consistent monotonic trend that arena does not detect (Sections 3.4, 5.2).
- Label-depth dose-response. Holding all else constant, increasing labeling depth from 100 to 4000 visits improves top-1 oracle agreement by +60% relative while value MAE is unchanged (Section 5.3).
- MCTS optimization. A stack of four optimizations achieves a 3.4× practical speedup on a single consumer GPU, bringing ∼12,000-simulation searches into a 30-second budget (Section 5.4).

Outline. Section 2 reviews related work. Section 3 describes the pipeline, network, evaluation methodology, and MCTS optimizations. Section 4 gives the experimental setup. Section 5 reports results. Section 6 discusses implications, including AI-interpreted style and the policy/value asymmetry. Section 7 concludes. Low-level engineering details, including position-sampling internals, KataGo labeling implementation, coordinate-axis conventions, and exact hyperparameters, appear in Appendices AC.

2 Related Work

AlphaZero family. AlphaGo [6], AlphaGo Zero [7], and AlphaZero [8] established the self-play paradigm in which a single network outputs a policy and value, and MCTS-derived statistics provide training targets. KataGo [10] accelerates this paradigm with several renements but retains its essential structure, and LeelaZero reproduces it via distributed volunteer compute. Reported training budgets remain far beyond what individual researchers can deploy. We depart from this paradigm at the label-generation stage: rather than producing labels through the student's own play, we treat an existing strong network as a xed oracle and learn to imitate it.

Knowledge distillation. Knowledge distillation [4] trains a small student to match a larger teacher's soft output distribution. The soft targets encode the teacher's relative preferences over outputs and carry more information than one-hot labels. FitNets [5] and Born-Again Networks [3] extend the idea to intermediate-layer matching and self-distillation. In games AI, distillation has typically appeared as a supplementary technique alongside self-playfor example, compressing a trained large network for deployment. We are not aware of a systematic study of distillation as the primary learning signal for a small Go network from training inception.

Evaluation in game AI. Head-to-head play (arena), aggregated through Elo rating systems [2] or online ladders such as CGOS, is the standard evaluation modality. A known weakness: when two engines share systematic late-game failures, arena outcomes can be dominated by these shared failures rather than by the strength dierence under study. Section 5.1 documents an extreme instance. Move-matching evaluation is well-established in human contests but comparatively rare engine-vs-engine; our Move Quality methodology (Section 3.4) applies the move-matching pattern to distilled engines, with the match computed against a strong-engine oracle.

LLM distillation and persona transfer. The closest contemporary analogue comes from the LLM literature. Alpaca [9], Vicuna [1], and WizardLM [11] ne-tune small open models on data generated by a stronger closed-source teacher; the small models absorb the teacher's interpretation of the target distribution rather than the original ground truth. We return to this analogy in Section 6, where it motivates a style-controlled extension of our Go pipeline.

3 Method

3.1 Pipeline Overview

The pure-distillation pipeline consists of three logically separable stages, executed oine (Figure 1):

Stage A: Position acquisition. Sample positions from external SGFs (KGS human games, KataGo self-play, legacy archives) into a feature tensor X. Provenance does not aect what is learned; diversity does.

Stage B: Labeling. Each position is labeled by KataGo b18c384 analysis at a chosen MCTS visit depth, producing policy π, value z, and ownership own targets.

Stage C: Training. Standard supervised multi-task training of the student against the KataGo labels.

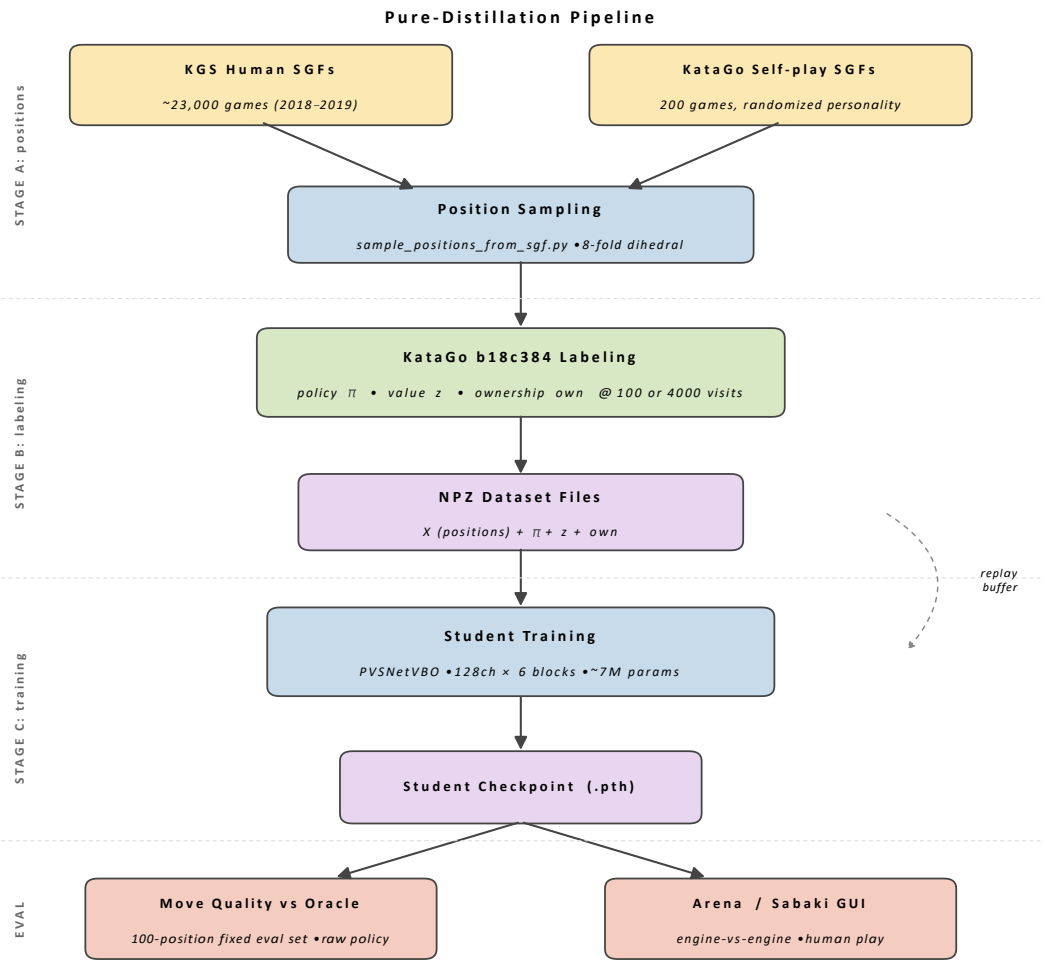


Figure 1: Overview of the pure-distillation pipeline. Positions sampled from KGS and KataGo self-play games are labeled using KataGo policy/value/ownership outputs, converted into compact NPZ datasets, and used to train a 7M-parameter student network. Stage A and Stage B are decoupled, so positions of any provenance can ow through the same labeling and training infrastructure.

Stages A and B are decoupled: positions of any provenance can be ingested into Stage B, and labeled NPZ les can be reused as extra-data replay buers across generations. The student is then evaluated by the Move Quality vs. Oracle protocol (Section 3.4, oine, no MCTS) and optionally by head-to-head arena via our MCTS implementation (Section 3.5).

3.2 Student Network

The student, PVSNetVBO, is a residual convolutional network with four output heads:

- Policy head: per-intersection logits plus a separate PASS head, total dimension $19^2+1 = 362$.
- Value head: scalar in $[-1, +1]$, current-player perspective.
- Score head: scalar normalized to $[-1, +1]$ (disabled in this work, $w_{score} = 0$).
- Ownership head: per-intersection prediction in $[-1, +1]$.

The trunk consists of an initial 3 × 3 convolution followed by six residual blocks of two 3 × 3 convolutions each with batch normalization, yielding approximately 7M trainable parameters. Input features are three 19 × 19 planes: black stones, white stones, and a to-play indicator. Architectural details and exact hyperparameters appear in Appendix C.

3.3 Data Sources and Labeling

We use three position sources, chosen for complementary characteristics: KGS human games (n ≈ 23,000, 20182019), local KataGo self-play (n = 200 with randomized rst move and per-color personality conguration), and a small CGOS archive (n = 16) included for diversity. A lesystem deduplication issue aecting early KGS sampling is noted where relevant in the model lineage (Section 4); position sampling internals, the local KataGo self-play runner, and exact sampling parameters appear in Appendix A.

KataGo labeling is performed via the JSON-RPC analysis engine of KataGo v1.14.0 (OpenCL backend, FP16). KataGo's policy and ownership outputs are reindexed into our internal convention before storage; the coordinate transform and the dihedral-augmentation consistency requirement it implies are illustrated in Appendix B. Other labeling-time details (pipe-buer handling, score-scale normalization) appear in Appendix A.

We label at two depths in this study: 100 visits per position (used through the d14o generation) and 4000 visits per position (used for the label-depth ablation in Section 5.3). On our hardware, 100-visit labeling proceeds at 45 positions per second; 4000-visit labeling at ≈ 0.15 positions per second.

3.4 Move Quality vs. Oracle Evaluation

The Move Quality protocol evaluates a model's policy and value directly against a high-quality KataGo reference on a xed evaluation set, without playing games. The protocol has four steps:

1. Evaluation-set construction. Sample 100 positions (50 KGS, 50 KataGo self-play) with one position per game and min_move = 50 to weight toward midgame and beyond.
2. Oracle labeling. Label all 100 positions with KataGo b18c384 at maxVisits = 4000, producing reference distributions π_{oracle}, values z_{oracle}, and ownership. The labeled le is saved once and reused across all model comparisons in this paper.
3. Model inference. For each model, a single forward pass on the 100 positions (chunks of 64) yields π_{model}. No MCTS : we isolate the network's prior preferences from the search algorithm.
4. Metrics. top-1 accuracy, top-3 accuracy, $\mathrm{KL}(\pi_{\text{oracle}} \| \pi_{\text{model}})$, cross-entropy at the oracle top-1, and value MAE.

The full protocol evaluates 100 positions × N models in approximately 5N seconds (excluding the one-time ∼16-minute oracle labeling). This makes Move Quality cheap to apply repeatedly during developmenta useful practical property in light of the arena limitations documented in Section 5.1.

3.5 MCTS Optimization Stack

Our baseline MCTS is a single-threaded Python translation of the AlphaZero-style PUCT algorithm. Prole measurements (Section 5.4) showed that NN inference at batch = 1 accounts for only ∼37% of search time on our system; the bulk is consumed by Python-level board operations, particularly

a naïve legal-move enumeration that copies the entire board state per candidate intersection. We implemented four optimizations:

Phase 1aNo-copy legal moves. In-place legality check that avoids copying the superko hash set on every candidate. Per-call cost drops from ~ 3.7 ms to ~ 1.2 ms.

Phase 2Tree reuse. The MCTS retains a cached root across consecutive searches; after a played move the cached root descends to the corresponding child, preserving accumulated visits. Disabled automatically when the same instance is shared across two evaluators per color.

Phase 3Leaf batching with virtual loss. Up to B leaves are descended in a single batched pass, evaluated in one NN forward pass, then backed up. Empirically $B = 8$ matches the sequential baseline's visit distribution; $B = 16$ is slightly faster but degrades the distribution.

Phase 4Time control. Per-call max_time and max_visits overrides; the loop checks the deadline between simulations (sequential) or at each batch boundary (batched).

All four optimizations are backward-compatible: passing no overrides and leaving $B = 1$ yields the baseline. Virtual-loss bookkeeping during batched descent, deadline-aware backup, and node-allocation details appear in Appendix A.

4 Experimental Setup

All experiments were conducted on a single workstation: NVIDIA RTX 4000 SFF Ada (20 GB VRAM, OpenCL with FP16 tensor cores), x86_64 CPU, 64 GB RAM, Windows 11 Pro. Software: Python 3.10+, PyTorch 2.x (CUDA), KataGo v1.14.0 (OpenCL), sgfmill for SGF parsing.

Model lineage. We refer throughout to four checkpoints, all sharing the same PVSNetVBO architecture (128×6, $\sim$7M parameters):

Table 1: Model lineage. Consecutive pairs dier in one variable each.

Tag	Init from	Primary training data	Label visits
d14f	d13c (self-play baseline)	10k positions re-labeled from d13c self-play	100
d14n	d14f	10k KGS positions (all KataGo labels)	100
d14o	d14n	30k KataGo self-play positions	100
d14p	d14o	Same 30k positions, re-labeled at 4000 visits	4000

The d14n training data was sampled under a Windows lesystem deduplication issue (Appendix A); we retain the aected data in the lineage to preserve apples-to-apples comparison across the four generations, since the post-x d14o measurement is consistent with the broader trend.

Training hyperparameters. All distillation runs used Adam with cosine learning-rate decay from 3×10^{-4} to 0, batch size 128, 10 epochs, 8-fold dihedral augmentation, $w_{policy} = w_{value} = 1.0$, $w_{score} = 0$, $w_{own} = 0.3$, $\beta_z = 0.7$. Each generation initializes from the previous checkpoint (init_any); value-head weights are not reinitialized. Training time is $\sim$45 minutes per generation on the test hardware. Full settings appear in Appendix C.

Arena conditions. Color-balanced 50-game arenas at sims = 400, $c_{puct} = 0.3$, Dirichlet root noise ($\alpha = 0.03$, $\epsilon = 0.10$) on the rst 5 moves, komi randomized over $\{5.5, 6.5, 7.5\}$, max_moves = 150.

Move Quality conditions. Raw forward pass on the 100-position evaluation set; no MCTS; oracle labels from KataGo b18c384 at 4000 visits, computed once and reused.

Reproducibility. Training and labeling are deterministic given the random seed. Move Quality evaluation is fully deterministic. Arena games are non-deterministic only via Dirichlet noise on the root.

5 Results

5.1 Arena Plateau

Color-balanced 50-game arenas between consecutive distillation generations yielded the outcomes in Table 2. In both comparisons, the winrate is statistically indistinguishable from 0.50 ($|z| < 0.4$).

Table 2: Arena outcomes between successive distillation generations.

Black	White	B-wins	W-wins	Winrate (B)	Maxout	B-color wins
d14m	d14f	26	24	0.520	90%	1/50
d14o	d14n	25	25	0.500	100%	0/50

Two pathologies are apparent. First, nearly every game (90100%) reached the 150-move cap without either player passing. Second, color wins are highly lopsided: across 50 d14o-vs-d14n games, Black won zero. Komi (5.57.5) combined with the networks' inability to recognize endgame closure makes the side nishing ahead at move 150 largely determined by initial conditions rather than by relative model strength. We read this as evidence that, under our setup, arena outcomes are dominated by color/komi eects and endgame-closure failures rather than by the underlying policy quality dierence between models.

5.2 Move Quality vs. Oracle

Table 3 reports Move Quality on the 100-position evaluation set. Figure 2 plots the same data. Random guessing yields top-1 ≈ 0.28%.

Table 3: Move Quality on the 100-position evaluation set. Despite arena (Section 5.1) reporting ≈ 0.50, Move Quality shows a consistent monotonic trend on every policy metric.

Model	top-1	top-3	KL	CE (oracle top-1)	value MAE
d14f (selfplay baseline)	0.000	0.000	8.504	10.352	0.951
d14n (pure distill)	0.020	0.020	4.272	6.043	0.643
d14o (+ KataGo SP positions)	0.100	0.170	3.639	5.265	0.343
d14p (re-label @ 4000 visits)	0.160	0.270	3.470	4.936	0.345

Three observations stand out. First, a monotonic trend appears across all four generations on every policy metric, in contrast with the arena plateau. Second, d14fthe self-play-trained baselinescores zero top-1 matches against the 4000-visit KataGo oracle, suggesting that its policy is approximately orthogonal to KataGo's at this measurement scale; d14p, by contrast, scores 16 matchesapproximately 57× above random chance. Third, a value/policy asymmetry rst becomes visible at the d14o→d14p step: policy metrics improve substantially (top-1 +60% relative) while value MAE is essentially unchanged (0.343 → 0.345). We return to this asymmetry in Section 5.3 and Section 6.

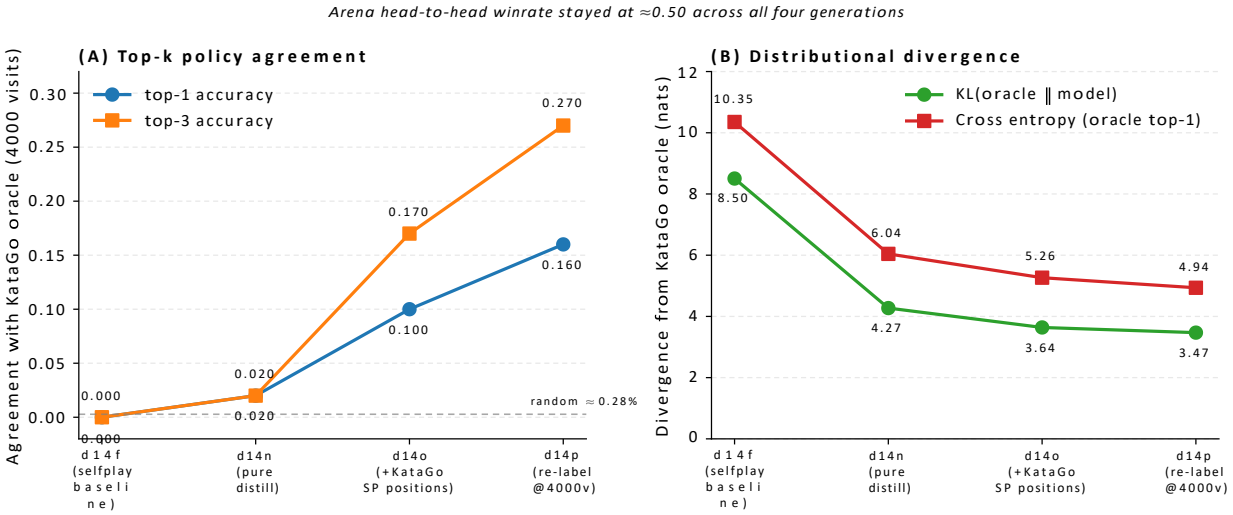


Figure 2: Move Quality metrics across successive distillation generations. Top-k oracle agreement and distributional divergence metrics improve monotonically across the lineage, while arena winrates remain near 0.50: policy metrics continue improving even when arena saturates. Random top-1 chance on a 19 × 19 board with PASS is ≈ 0.28%.

5.3 Label-Depth Ablation

d14o and d14p dier in exactly one variable: the MCTS visit depth used during KataGo labeling of the 30k KataGo-self-play training inle. d14o uses 100 visits per position; d14p uses 4000. All other training settings are identical.

The labeled NPZ le size reects the dierence in label density: d14o's is 24 MB, d14p's is 65 MB. At 100 visits, KataGo's policy distribution is sharply peaked on a handful of moves (most candidates receive zero visits and end up with prior weights only); at 4000 visits, the distribution is meaningfully spread across 3050 moves with graded probabilities. Comparing the last two rows of Table 3:

Metric	d14o (100v)	d14p (4000v)	Rel. Δ
top-1	0.100	0.160	+60%
top-3	0.170	0.270	+59%
KL	3.639	3.470	−4.6%
CE	5.265	4.936	−6.3%
value MAE	0.343	0.345	+0.6%

Policy metrics improve substantially while value calibration appears unchanged. We interpret this as evidence that the policy distribution shape carries depth-dependent informationat 100 visits KataGo's distribution is a compressed argmax-like signal; at 4000 visits it encodes ner relative preferences that the student can absorbwhereas the value output is largely determined by KataGo's game-outcome signal at any visit count and appears to saturate early. This asymmetry has implications for label-budget allocation in future small-network distillation work, which we discuss in Section 6.

5.4 MCTS Throughput

We proled the baseline MCTS implementation under sims = 400 on 19 × 19 positions with d14p. Proling revealed two surprises: NN inference at B = 1 consumed only 37% of search time, while

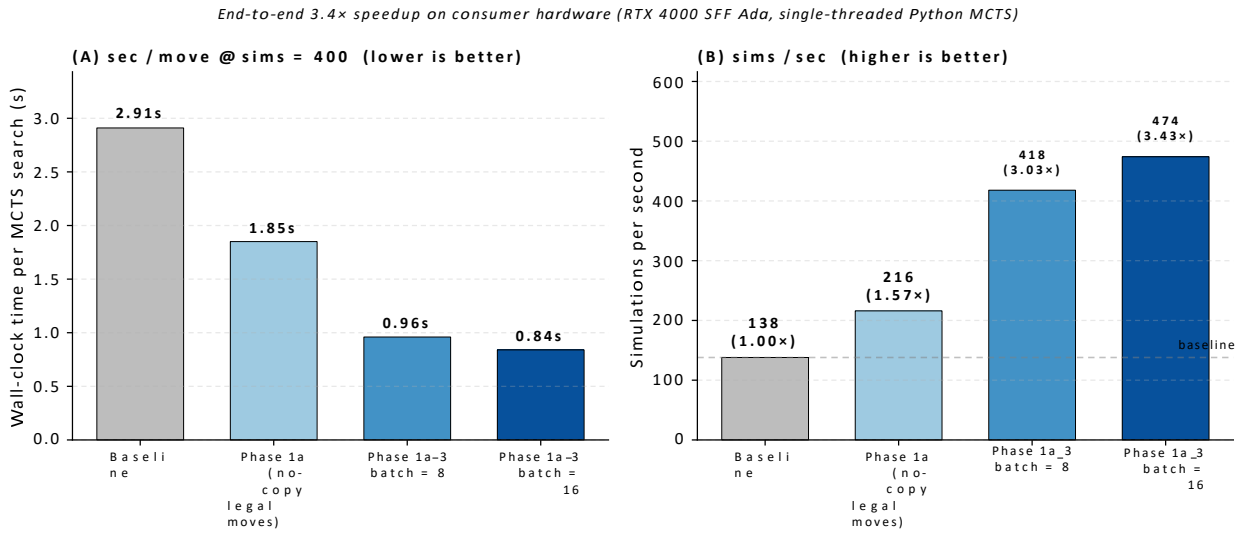


Figure 3: MCTS throughput improvements obtained through no-copy legal-move enumeration (Phase 1a), tree reuse (Phase 2), and batched leaf evaluation (Phase 3). The nal implementation achieves a 3.4× practical speedup on consumer hardware. Raw NN-inference scaling (Appendix A, Figure 4) is faster still the end-to-end speedup is bounded by Python overhead in node allocation and selection.

the legal-move primitive consumed 51%the largest single block. The naïve implementation called board.copy() followed by a trial play for every empty intersection, with each copy dominated by a deep-copy of the monotonically growing superko set.

Table 4 reports per-call wall-clock cost at sims = 400 as Phases 13 are added. Figure 3 plots the same data.

Table 4: MCTS wall-clock time at sims = 400 on consumer hardware. The baseline column is the original implementation prior to Phase 1a.

Conguration	sec/move	sims/sec	Speedup
Baseline (pre-Phase 1a)	2.91	138	1.00×
Phase 1a, B = 1	1.85	216	1.57×
Phase 1a3, B = 8	0.96	418	3.03×
Phase 1a3, B = 16	0.84	474	3.43×

At B = 16, a 30-second per-move time budget produces approximately 14,000 simulations comparable to the new-visit count of modern engines operating with tree reuse and multi-threaded virtual-loss MCTS, even though our implementation remains single-threaded Python. The end-to-end speedup of 3.4× is bounded below the raw NN improvement (Appendix A) because the legal-move primitive, while 3× faster after Phase 1a, still consumes ∼37% of post-optimization wall-clock time, and Python overhead in node allocation and selection scales with simulation count regardless of batch size.

5.5 Qualitative Sabaki Observations

Throughout the distillation lineage, the same human Go player (a strong amateur) played against the network through the Sabaki GUI, unaware of the Move Quality results until the project's conclusion. Reported observations: d14f and earlier showed frequent rst-line plays, empty-area plays, meaningless invasions, and contact-and-die patterns; d14n was marginally smoother but still poor;

d14o was the rst generation in which stones went in productive directions with whole-board awareness and KataGo-like shape; d14p with Phase-3 fast MCTS reduced stone clumping but retained moves that don't follow up and locally meaningless moves. The player's rst identication of KataGo-like shape at d14o coincides exactly with the rst generation crossing 10% top-1 Move Quality, and the residual weakness (moves that don't follow up) matches the quantitative nding that value calibration plateaued while policy improved.

6 Discussion

6.1 Why Arena Failed

The 0.50-winrate plateau across multiple consecutive generations is not an instance of the models are equal. It is an instance of arena measuring something other than the policy dierence under study. Two mechanisms appear to operate jointly. First, all four generations share a common pathology: they cannot reliably close out endgames. Move Quality shows that the networks have learned much about opening and midgame, but value calibration plateaued early; the network does not produce condent this position is won/lost signals, and its policy rarely collapses onto PASS as an obviously preferred move. Games therefore tend to reach the 150-move cap, and the position at move 150 is scored by area. Second, when games are decided by area count at a xed move cap and neither network has a strong preference for closing, the outcome appears to be dominated by komi and color: White's komi compensation favors White on positions of approximately even territorial development. The 50/50 model winrate emerges largely from arithmetic: each network plays 25 games as White (winning most of them) and 25 as Black (losing most of them).

The implication we draw for practice is the following: under shared maxout behavior, arena alone is unlikely to be sucient for evaluating small Go networks, and an auxiliary metric that does not require games to terminate is useful.

6.2 Why Move Quality Worked

Move Quality vs. Oracle sidesteps these failure modes by construction: it does not run games (no termination requirement, no maxout, no komi eect), it is symmetric in color (positions sampled from both colors' perspectives), and it evaluates raw network policy (dierences in MCTS implementation, sims budget, and virtual loss are eliminated). The price of these properties is that Move Quality measures only the network's prior preferences and not its interaction with search; an MCTS-derived variant would conate policy quality with MCTS-stack quality. The raw-policy version is diagnostically cleaner.

6.3 Policy/Value Asymmetry

The label-depth ablation isolates an apparent asymmetry: policy distribution quality is sensitive to oracle visit depth, while value calibration is not. At 100 visits, KataGo's policy is essentially an argmax with a few also-rans; at 4000 visits, it carries graded preferences over the top 3050 moves, and the student can absorb that gradation. Value, by contrast, derives primarily from game-outcome rollouts and appears to saturate quickly. A practical implication is that label budgets may not be best spent uniformly: increasing visit depth seems to provide outsized returns on the policy head, while value supervision may benet from dierent interventionsauxiliary heads, score-margin signals, or self-play ne-tuning on top of the distilled prior. A natural next step is

to investigate whether a hybrid distillationRL stage, with KataGo policy held xed and value retrained against self-play outcomes, can close the value gap without degrading the policy.

6.4 AI-Interpreted Style

The distillation pipeline as constructed has a useful invariant: position provenance is decoupled from the labeling oracle. If positions are drawn from a corpus characteristic of a particular human player or style, the resulting student would not be a model of that playerit would be a model of KataGo's interpretation of those positions, in the same sense that a small LLM ne-tuned on GPT-rendered author text appears to learn the teacher's interpretation of the author rather than the author directly. This parallels the Alpaca/Vicuna pattern in LLM practice [1, 9]. For Go specically, this suggests a class of style-controlled small networks: feed games by a given player through KataGo, and the student could become a KataGo-rendered version of that playerthe student's policy distribution biased toward the kinds of shapes that player tends to produce in the positions KataGo nds in their games, while the labels themselves remain KataGo's. We do not validate this empirically here; we note that the pipeline is constructed to make it a straightforward extension.

6.5 Limitations

Our results have several caveats. Training scale is limited: the student is a single 7M-parameter architecture, the labeling corpus totals ∼50k positions, and we report four generations. We do not have full Elo validation: arena failure motivated Move Quality, but Move Quality itself is a teacher-relative metric and does not directly establish Elo. The compact model has shallow tactical reading; the value head's plateau may reect this. The endgame-closure failure is observed across the lineage and is not addressed by distillation alone. All experiments use a single hardware environment. Future work should validate Move Quality against multiple teachers, broaden the position corpus, and explore the hybrid distillationself-play stage suggested by the policy/value asymmetry.

7 Conclusion

We have presented a pure-distillation pipeline for compact Go networks in which positions of arbitrary provenance are labeled by a strong existing engine and supervised training produces a small student that imitates the teacher. Across four distillation generations, head-to-head arena reports indistinguishable ≈ 0.50 winrates, but our Move Quality vs. Oracle methodology shows a consistent monotonic trend on every policy metric. A label-depth ablation shows that policy quality is sensitive to oracle visit depth while value calibration is nota policy/value asymmetry with implications for label-budget allocation. A stack of MCTS optimizations delivers a 3.4× end-to-end speedup, bringing ∼12,000-simulation searches into a 30-second budget on consumer hardware. The pipeline naturally enables an AI-interpreted style direction for compact Go networks, analogous to recent LLM persona work.

The broader methodological point: when arena saturates because of a shared engine weakness rather than a true equality of strength, a teacher-relative evaluation can recover an underlying signal and provide fast iteration feedback unavailable to arena.

Acknowledgements

Assistant drafting support: Claude (Anthropic).

References

[1] W.-L. Chiang et al. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. lmsys.org blog post, 2023.

[2] A. E. Elo. The Rating of Chessplayers, Past and Present. Arco Pub., 1978.

[3] T. Furlanello, Z. Lipton, M. Tschannen, L. Itti, and A. Anandkumar. Born-again neural networks. ICML, 2018.

[4] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. NeurIPS Deep Learning Workshop, 2015.

[5] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for thin deep nets. arXiv:1412.6550, 2014.

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

[7] D. Silver et al. Mastering the game of Go without human knowledge. Nature, 550:354359, 2017.

[8] D. Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science, 362(6419):11401144, 2018.

[9] R. Taori et al. Stanford Alpaca: An instruction-following LLaMA model. GitHub repository, 2023.

[10] D. J. Wu. Accelerating self-play learning in Go. arXiv:1902.10565, 2019.

[11] C. Xu et al. WizardLM: Empowering large language models to follow complex instructions. arXiv:2304.12244, 2023.

A Implementation Details

This appendix collects engineering details factored out of the main text: position sampling internals, the local KataGo self-play runner, KataGo labeling implementation, MCTS internals, and a lesystem bug whose eects are documented in the d14n/d14o data lineage.

A.1 Position Sampling

From each SGF source, we sample positions via a sample_positions_from_sgf script. For each game we play the moves out with sgfmill, sampling K board states drawn uniformly from move indices in [min_move, total_moves − max_move_from_end]. For most experiments min_move = 30 and max_move_from_end = 10; for the Move Quality evaluation set min_move = 50 to weight toward midgame and beyond.

A.2 KGS Filesystem Bug (Pre-Fix Data Lineage)

Under Windows lesystem semantics, naïve recursive globs for both *.sgf and *.SGF patterns match the same physical le twice. Earlier samples from the KGS source were aected by this duplication; positions were drawn from approximately half the nominally available games. We xed the bug by deduplicating glob results (sorted(set(...))) and report all results post-x. The earlier data including the d14k_kgs_external.npz used in the d14m and d14n trainingwas generated under the bug; we deliberately retain it to preserve a clean apples-to-apples comparison with the post-x d14o data and report this issue here for transparency.

A.3 Local KataGo Self-Play Runner

To complement KGS positions with high-quality strong-AI play, a self-play runner orchestrates two KataGo GTP subprocesses per game. Each game randomly selects a personality conguration (ag-gressive / defensive / normal) for each color via playoutDoublingAdvantage and analysisWideRootNoise, and places a random rst stone from a xed hoshi/3-3/tengen pool. Visits per move are capped at 100 via kata-set-param maxVisits 100. We do not use temperature for move selection during these games; diversity comes from personality and opening rather than stochasticity. Generation throughput on a single RTX 4000 SFF Ada is ≈ 100 s per game.

A.4 KataGo Labeling

We label each sampled position by querying KataGo b18c384 v1.14.0 (OpenCL, FP16) via its JSON-RPC analysis engine. Three implementation issues required care:

Pipe deadlock at scale. Writing all queries before reading any responses caused both the OS pipe buer and KataGo's stdout buer to ll before either was drained. We use a writer thread that submits queries while the main thread drains responses; KataGo's stderr is similarly drained by a third thread.

Axis convention mismatch. Our project indexes board positions as idx = $y \cdot 19 + x$ with $y = 0$ at the bottom of the board (Go row 1). KataGo's policy and ownership arrays use idx = $y_{kata} \cdot 19 + x$ with $y_{kata} = 0$ at the top. We vertically ip the spatial component of KataGo's outputs (policy[:361].reshape(19,19) → flipud → flatten), similarly for ownership, so that the labeled NPZ matches our internal convention. The PASS channel (index 361) is left untouched.

Appendix B illustrates this convention and the augmentation-consistency requirement that follows from it.

Score scale. KataGo's scoreLead is reported in raw points; internally we store score normalized to $[-1, +1]$ by board size. For the present work we set score-related weights to zero during training and emit zeros for the s eld in labeled NPZ les; labeled value z (winrate-derived) and own (ownership) are the labels that drive training.

A.5 MCTS Implementation Notes

Phase 1a: legal-move primitive. The naïve implementation called board.copy() for every empty intersection, with each copy dominated by a deep-copy of the monotonically growing superko set _seen. The Phase 1a implementation saves only the numpy board array ($\approx$ 1 µs vs the $\approx$ 18 µs set-copy), places the candidate stone, evaluates capture and suicide locally, queries _seen without copying it, and restores the board array. Per-call cost drops from $\approx$ 3.7 ms to $\approx$ 1.2 ms on a mid-game 19×19 position.

Phase 3: virtual loss bookkeeping. Each descent in a batched search applies virtual loss along its path: the node's virtual_loss counter increments by one. During PUCT selection, the virtual loss is included both in the denominator and as a pessimistic $Q = -1$ contribution to the numerator, causing subsequent descents in the batch to diverge from already-selected paths. After collection, all non-terminal leaves are evaluated in a single batched NN forward pass; virtual losses are removed, leaves are expanded with their respective NN outputs, and values are backed up. Empirically $B = 8$ provides stable speedup with policy distributions virtually identical to the sequential baseline (top-5 visit counts within ± 1); $B = 16$ is slightly faster but degrades visit distribution as virtual-loss-induced pessimism distorts selection.

Tree reuse safety. We disable tree reuse automatically when the same MCTS instance is used in arena mode against a dierent opponent network for each color, since accumulated Q values would mix evaluations from two dierent evaluators.

Phase 4: deadline-aware loop. search(max_time, max_visits) accepts per-call overrides of the simulation budget. With a deadline set, sequential mode checks time.perf_counter() >= deadline between simulations; batched mode checks at each batch boundary, so the actual wall-clock time may overshoot the deadline by up to one batch's worth of NN inference. Combined with tree reuse (Phase 2), this provides KataGo-style usage: think for N seconds naturally accommodates positions of varying complexity, and inheriting visits across moves means the displayed visit count at any point in the game can substantially exceed the per-call new-simulation budget.

A.6 NN Inference Scaling

Figure 4 reports raw forward-pass throughput on a single RTX 4000 SFF Ada (20 GB), independent of MCTS overhead.

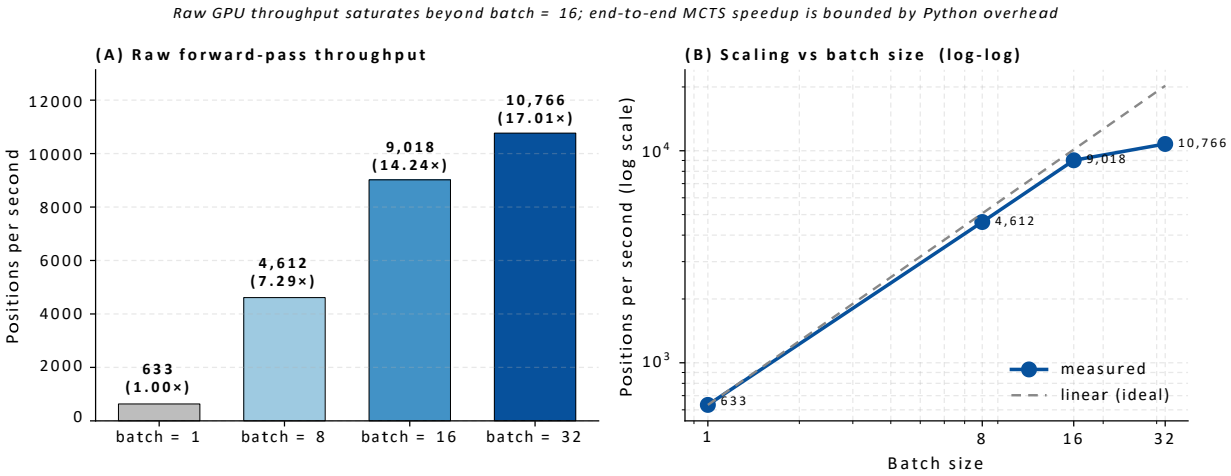


Figure 4: Raw NN-inference throughput on d14p, single RTX 4000 SFF Ada. (A) Bars at batch $\in \{1, 8, 16, 32\}$ with speedup relative to batch 1. (B) Log-log scaling against the linear-ideal reference. Raw GPU throughput scales near-linearly to batch 16 and saturates beyond; end-to-end MCTS gains (Figure 3) are bounded below by Python overhead in node allocation and selection.

B Coordinate Convention and Augmentation Consistency

Figure 5 illustrates the axis-ip convention applied to KataGo's policy and ownership outputs, and the augmentation-consistency requirement that follows from it.

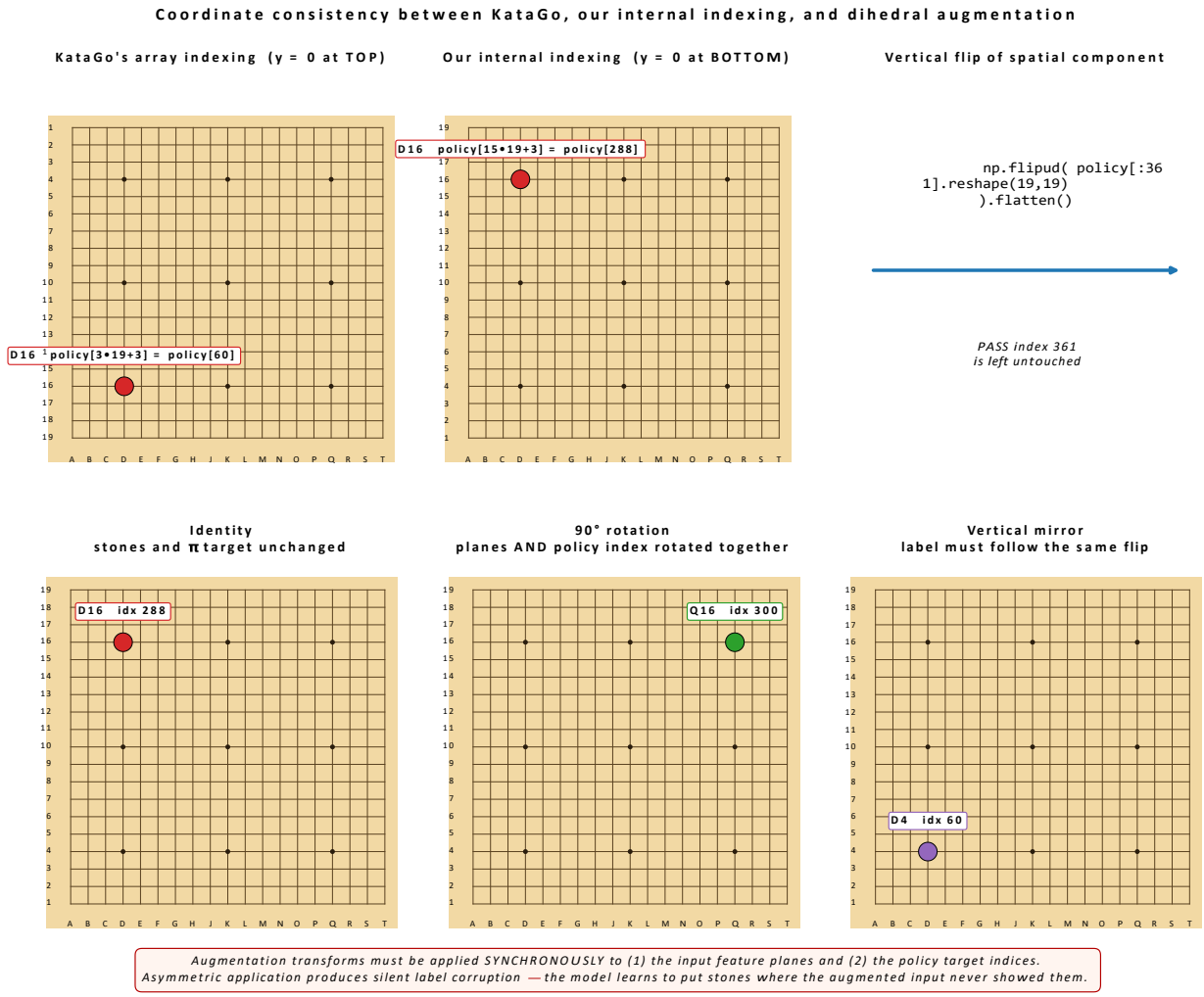


Figure 5: Coordinate consistency during labeling and augmentation. Top: KataGo indexes the board with y = 0 at the top (row label 19); our internal convention places y = 0 at the bottom (row label 1). The point D16 maps to KataGo index 60 and to our internal index 288. The transform is `flipud` on the 19×19 spatial component; the PASS channel (index 361) is untouched. Bottom: dihedral augmentation must transform the input feature planes and the policy target indices in lock step. Identity maps D16 to itself; 90° rotation maps D16 to Q16 (index 300); vertical mirror maps D16 to D4 (index 60). Asymmetric application of the augmentation produces silent label corruption: the model is trained to predict moves at intersections that the augmented input never showed.

C Exact Hyperparameters and Reproducibility

Table 5 reports the full set of hyperparameters used for every training run reported in this paper. Dierences between runs (initialization, primary inle, replay buer composition, labeling visits) are captured by Table 1.

Table 5: Training hyperparameters held constant across all distillation runs reported in Section 5.

Parameter	Value
Architecture	PVSNetVBO, 128 channels × 6 residual blocks
Parameters	∼7 M
Input planes	3 (black, white, to-play), 19 × 19
Optimizer	Adam
Initial learning rate	3×10^{-4}
LR schedule	Cosine decay to 0
Batch size	128
Epochs	10
Augmentation	8-fold dihedral symmetry
w_{policy}	1.0
w_{value}	1.0
w_{score}	0.0 (score head disabled)
w_{own}	0.3
β_z (zscore mix)	0.7
Replay buer	2 previous-generation NPZ les
Initialization	Previous-generation checkpoint via init_any
Training time	∼45 min per generation on RTX 4000 SFF Ada

Arena hyperparameters. sims = 400, c_{puct} = 0.3, Dirichlet noise α = 0.03, ϵ = 0.10 on the rst 5 moves, komi randomized over {5.5, 6.5, 7.5}, max_moves = 150, 50 games per pairing color-balanced.

Move Quality hyperparameters. 100-position xed evaluation set (50 KGS + 50 KataGo self-play), oracle labeling at 4000 visits, raw forward pass on the student (no MCTS), chunks of 64 for memory.

Reproducibility. Training and labeling are deterministic given the random seed (42 for most sampling steps; distinct seeds where ablation comparison requires independent draws). Move Quality evaluation is fully deterministic. Arena games are non-deterministic only via Dirichlet noise on root nodes. The released artifact consists of the four model checkpoints (d14f, d14n, d14o, d14p), the evaluation set (eval_positions_v1_labeled.npz), the labeling scripts, and the training, arena, and Move Quality scripts. KataGo b18c384 is publicly distributed and not redistributed here.