

Mikiri: Learned Stopping and Exact Search Memory on a Frozen Go Engine

Taka Khoo

Independent researcher

`takakhoo@gmail.com`

Technical report, October 2026. A short version is in preparation for IJCAI 2027; not yet peer reviewed.

Abstract

A strong Go player glances at a forced recapture and plays it, then sits for ten minutes over a fight that decides the game. A Go engine gives every move the same search budget. We ask how much stronger a frozen engine becomes from two changes that leave its network and its search untouched: deciding move by move how long to search, and never searching the same position twice. Mikiri wraps KataGo with a short ladder of visit budgets, a learned rule that stops once the winrate still at stake no longer justifies the next rung, and a memory keyed on symmetry-canonical positions. On 6,000 positions, each searched at seven budgets and scored by independent forced searches, the stopping rule matches uniform search at 1.7 to 2.1 times the visits, ahead of ten published stopping rules re-implemented on the same engine. In 19×19 play against KataGo at 200 visits per move, Mikiri at the same mean visits scores 78.8% over 1,000 games with a six-rung ladder (+228 Elo, 95% interval +203 to +254) and 76.6% over 1,000 games with a four-rung one (+206 Elo, +181 to +232). Doubling KataGo’s own budget is worth +237, and the strongest published rule, played the same way, gains +135. Held to 160 visits per move, so that it requests fewer visits than its opponent even when every restarted search is counted in full, Mikiri scores 63.0% (+93 Elo, +72 to +114). Exact positions recur often: a memory keyed on canonical positions serves 21.2% of moves in a 600-game series with no loss of strength, though the network evaluations it saves were already cheap, because the engine caches them. Approximate retrieval, the idea the first version of this project was built on, does not transfer: a similar stored position points to the right move 5.5% of the time, against 43% for the policy network’s own second choice. We also show that the match records reported for that first version were produced by a protocol error, and we release the corrected testbed, dataset, models, and game records.

1 Introduction

Watch a strong player and you will see two kinds of move. Some are played almost before the opponent’s stone has settled: a recapture, a connection, the only move. Others get long, visible thought. The player is not being inconsistent. They are spending attention where the game can still change.

Engines in the AlphaGo line [23, 24] do not make that distinction. They choose a move with a tree search guided by a policy network, which proposes moves, and a value network, which judges positions. The search is given a fixed number of *visits*, which count how many times it descends the tree, and strength is measured by how well it plays at that number. KataGo [28], the strongest engine anyone can download, follows the convention. At 200 visits per move, a forced recapture

and a whole-board fight get 200 visits each, and a position the engine analysed an hour ago in another game is analysed again from nothing.

How much do those two habits cost? This paper measures it. We keep the network and the search fixed and change only how many visits a move receives. The system is called Mikiri, after a Japanese word for the judgment that one has seen enough. It makes two kinds of decision. Before searching, it checks whether it has already searched this position, in any of its eight orientations. While searching, it climbs a short ladder of budgets and asks after each rung whether the decision is settled. Measuring whether that helps turned out to be most of the work, because a search cannot grade its own choices; the testbed we built for it is the first contribution below.

The project began with a different idea. Its first version, DataGo, proposed retrieval-augmented search: store deep analyses, find similar positions with an approximate nearest-neighbour index, and reuse what was found. It reported 9–0–1 and 8–0–2 records against KataGo. When we rebuilt the evaluation we found that those records came from a protocol error that let DataGo place extra stones (Section 3), and that retrieved moves had never reached the board. So we started again from the questions, with a testbed in which every visit is counted and both players are otherwise identical.

What we find.

1. **Search error is rare, concentrated, and partly predictable.** At 200 visits, 71% of positions have no regret at all and the worst tenth hold 84% of it. No single search statistic identifies them well (AUC about 0.74), yet a small model with a simple stopping rule recovers a large share of the waste (Section 6).
2. **A learned stop is worth nearly a doubling of search.** Offline, Mikiri’s stopper matches uniform search at 1.7 to 2.1 times the visits, depending on the ladder. In matches against KataGo at 200 visits it gains +228 Elo at equal visits with a six-rung ladder and +206 with a four-rung one, and +93 Elo when held to 160 visits, where doubling KataGo’s budget gains +237 (Section 7).
3. **It outperforms ten published stopping rules on the same engine.** The strongest of them, the virtual-expansion rule of Ye et al. [32], reaches a multiplier of 1.60 where Mikiri reaches 2.13, a paired difference of 0.54 [0.28, 0.84] (Section 6.5). Played against KataGo at the same 200 visits it gains +135 Elo to Mikiri’s +206 to +228. It also runs fewer network evaluations; at matched evaluations Mikiri leads it by about 60 Elo, and on the prior rule’s own short ladder the two are level (Section 7.4).
4. **The rule matters more than the model.** Dividing predicted regret by the cost of the next rung is what turns a stopping signal into a gain. With a flat threshold, the entropy gate of the first version is worse than no gate, and with the rate rule a prior learned rule draws level with ours at some budgets. Most features can be removed without loss.
5. **Exact memory is an opening book that builds itself, and the engine’s cache already does most of its job.** Positions recur often enough that 21.2% of Mikiri’s moves in a long series need no search, at no cost in strength. Those moves were already cheap in network evaluations, because KataGo caches evaluations across games.
6. **Approximate retrieval does not help a frozen engine.** A stored search transfers when the position is the same and almost never otherwise (Section 8.2).

Everything is released: the testbed, the 6,000-position ladder dataset, the stopping model, every game record, and the audit of the earlier results.¹

2 Related work

Retrieval and memory in game-playing agents. The closest prior work is Humphreys et al. [10], who give a MuZero-style 9×9 Go agent the nearest neighbours of the current position from a database of about 50 million positions. Their network is trained to read those neighbours, and win rate against Pachi rises by several points. Goyal et al. [7] train a retrieval process alongside an Atari agent. Both change the network. Episodic control [2, 21] stores returns in a nearest-neighbour table and helps most early in training, a regime a converged Go network left long ago. Inside a single search, Xiao et al. [30] blend each node’s value with a memory of similar states and report 61% wins at equal time for a 2018 Fuego baseline, and Srinivasan et al. [25] share value estimates across nearby states with a convergence guarantee. We found no published result in which retrieval improves a superhuman Go or chess engine at matched compute without retraining it.

Exact reuse of search. Transposition tables [4] and graph search [5, 12] share statistics between identical positions inside one search. Opening books do the same across games. KataGo’s own books [29] key positions by a symmetry-canonical hash, as we do, and cover small boards with very deep searches. Chaslot et al. [3] choose which book positions deserve analysis with an outer tree search. Reanalyse [22, 31] re-searches stored positions to produce training targets. Our memory is the across-game version of a transposition table for 19×19 play: it is filled by the engine’s own online searches and deepened in proportion to how often a position recurs.

Deciding how long to search. Time management for Monte-Carlo tree search in Go predates neural engines. Huang et al. [9] extend search when the most-visited move is not the highest-valued one, and Baier and Winands [1] find that stopping early when the leader cannot be overtaken, then reinvesting the time, is the most reliable strategy across five games. For AlphaZero-style agents, Lan et al. [16] train small networks to predict whether the current best move will survive more simulations, and Ye et al. [32] stop when the visit distribution has converged, reaching similar strength at about half the simulations on 9×9 Go. Muppidi et al. [20] learn a per-state budget for a frozen planner with reinforcement learning. Hay et al. [8] frame the choice as a metalevel decision problem. Leela Chess Zero ships engineering versions of the same idea [18]. These results use small boards, small networks, or classical playouts. KataGo already adapts search inside the tree, with uncertainty-weighted playouts and variance-scaled exploration [13], so a wrapper can only add value in how many visits a move receives in total. We test that on 19×19 with a modern network and measure the gain in played games.

Lessons from retrieval-augmented language models. Interpolating a frozen language model with nearest neighbours helps [15], with the gain concentrated where neighbours are near-duplicates [6] and on rare inputs [19]. Section 8.2 reports the Go analogue: transfer from a neighbour is reliable only when the neighbour is the same position. One reason is specific to the domain. Lan et al. [17] show that adding two stones that should not matter can change what a strong engine plays, so proximity in a smooth embedding is weak evidence about the best move.

¹<https://github.com/takakhoo/mikiri-beats-katago>

Limits of extra search. Adversarial policies beat KataGo even at very high visit counts [26, 27]. A stopping rule built on the engine’s own search statistics inherits those blind spots: it cannot flag a position the network is confidently wrong about. Our claims concern ordinary play between engines.

3 Testbed

Every number in this paper comes from one harness, built so that the two players differ only in how they spend visits.

Engine. Both players call KataGo v1.18.2 (CUDA backend) through its JSON analysis engine [11]. A query names a position, a visit limit, and optionally the root moves the search may consider. Each query is an independent search: the engine clears its tree afterwards, so no player inherits a tree from an earlier move. We run one search thread per query, which removes virtual-loss effects and makes a visit mean the same thing at every budget, and we get throughput from 64 concurrent queries. The analysis engine widens the root by default (`wideRootNoise` 0.04). We set it to zero, as the engine’s documentation advises for play. Unless stated otherwise the network is `kata1-b18c384nbt-s9996604416-d4316597426`.

Rules and move selection. Games use area scoring, positional superko, no suicide, and komi 7. Under these rules KataGo puts Black’s winning chances from the empty board at 48% with komi 7 and at 37% with komi 7.5, so komi 7 is the fair setting; a tied game counts half a point for each side. Both players choose moves the way KataGo’s GTP front end does by default: they sample in proportion to visits raised to $1/T$, with T falling from 0.5 toward 0.1 with a half-life of 19 moves, among moves holding at least 2% of the visits. A player resigns after three consecutive turns below 3% winrate, and never before move 90. Games that reach two passes are scored on the board by Tromp-Taylor area.

Position identity. The harness keeps its own board with captures, simple ko, and positional superko. Over 40 recorded games (7,247 moves, 381 captures) its stones agreed with KataGo’s own board at every one of 309 checkpoints. A position’s key is a hash of the lexicographically smallest encoding of stones, ko point, and side to move over the eight board symmetries, together with komi and rules. Symmetric positions share a key, and stored moves are mapped back through the query’s own symmetry.

Three measures of compute. A player that searches a position at 50 visits and then at 200 has issued 250 visits of queries. Whether it has done 250 visits of work depends on the engine. With one search thread, KataGo’s search is deterministic given its cached network evaluations, and the 50-visit tree is a prefix of the 200-visit tree. We measured this directly on fresh engines: over six midgame positions, a 200-visit search followed by a 400-visit search evaluated 366 to 396 network rows, against 353 to 395 rows for the 400-visit search alone. Restarting at a higher budget therefore costs about what extending the tree in place would. We report three numbers for every match and leave the choice to the reader:

- **continuation visits:** the size of the deepest search run for a move, which is what an in-engine implementation would pay;
- **restart visits:** the sum of every search requested, an upper bound;

- **network rows:** evaluations each player’s own engine process actually executed, read from its log at exit.

Each player runs in its own KataGo process so the third number is attributable. Budgets are matched on continuation visits.

Match formats. A *series* is repeated play from the empty board with alternating colours, which is how an engine is used in practice and is the setting in which positions recur. Consecutive games form colour-balanced pairs, and intervals are bootstrap percentiles over pairs. Elo differences are $-400 \log_{10}(1/s - 1)$ for score s .

What changed from the first version. The first version of this project, called DataGo, reported 9–0–1 and 8–0–2 records against KataGo. Those matches drove one KataGo process through GTP and analysed positions with `kata-genmove_analyze`, a command that also plays the move it generates. Every deep search therefore added a black stone before White replied, and retrieved moves were never sent to the engine. We reproduce the behaviour with the original controller in the repository’s `legacy/AUDIT.md`: three black stones after one black turn. None of the earlier numbers are used here.

4 Measuring what more search buys

Before building a gate we need to know what it could gain, position by position. We collect that in a *ladder dataset*.

Positions. We played 300 self-play games with the test network, each at a per-game visit budget drawn log-uniformly from 30 to 200, and sampled 20 positions from each game uniformly over move numbers after the fourth. This gives 6,000 positions from the distribution a low-budget player actually meets, from first corner moves to late endgame.

Ladder. Each position is searched at 50, 100, 200, 400, 800, 1,600 and 3,200 visits. We record the engine’s chosen move and summary statistics at every rung.

An independent yardstick. A search cannot grade itself: the value it reports for its own choice is the very estimate that might be wrong. For every distinct move that any rung chose, and for the top two moves of the deepest rung, we run a separate search restricted to that move at the root (1,600 visits). Its winrate, $q(s, a)$, is our value for playing a in s . All moves are scored the same way, by a search that did not choose them, so a comparison between two stopping policies is not biased toward either. The *regret* of stopping at rung k is

$$r_k(s) = \max_{a \in C(s)} q(s, a) - q(s, a_k(s)), \quad (1)$$

where $a_k(s)$ is the move chosen at rung k and $C(s)$ is the set of scored moves. Regret is in winrate units, so it is automatically small in positions that are already decided and large where the game can still turn.

How Mikiri decides one move

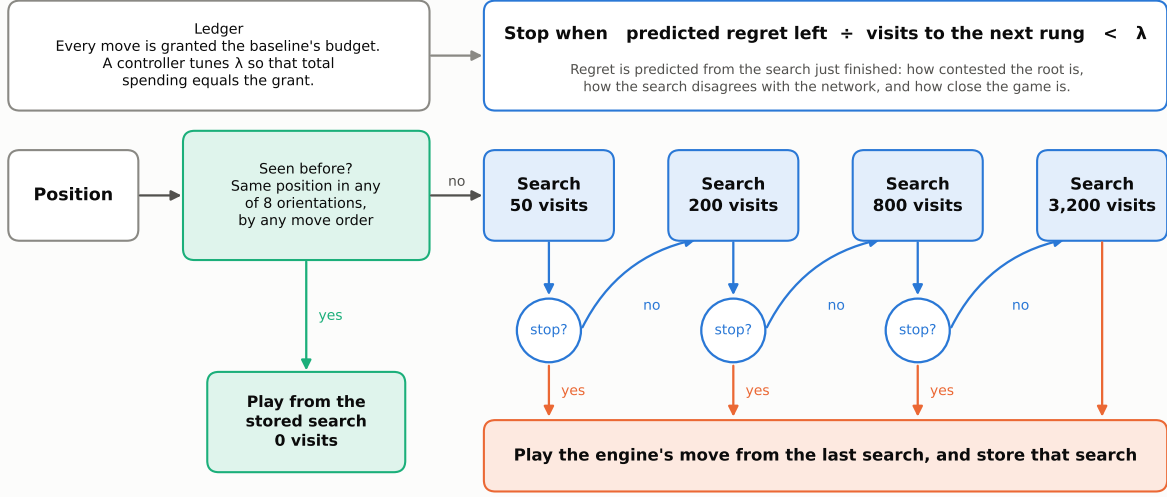


Figure 1: How Mikiri decides one move.

Equivalent visits. The mean regret of always stopping at rung k traces a curve $R(v)$ against visits. For any other policy with mean regret $\bar{r}$ we report the uniform budget v^* with $R(v^*) = \bar{r}$, interpolating R linearly in $\log v$. The ratio of v^* to the policy’s own mean cost is its *compute multiplier*: how many times more visits uniform search needs to give up as little.

5 Mikiri

Mikiri wraps a fixed engine and changes only how many visits each move receives. It has three parts, shown in Figure 1.

5.1 A visit ladder with a learned stop

Mikiri searches a position at the first rung of a short ladder, for example 50 visits, and asks a model whether the decision is settled. If it is, Mikiri plays. If not, it searches at the next rung (200, then 800, then 3,200) and asks again.

The model is a gradient-boosted regression tree ensemble that predicts the regret remaining at the current rung, $\hat{r}_j(x)$. Its inputs come from the search that just finished, at no extra engine cost:

- how contested the root is: visit share of the top two moves, visit entropy, number of moves above 5% of visits, the margin between the best move’s lower confidence bound and the best rival’s winrate, and whether the most-visited move is also the highest-valued one;
- how the search disagrees with the network: prior of the chosen move, whether it is the top-prior move, the KL divergence from prior to visit distribution, and the gap between the raw network winrate and the searched winrate;
- what is at stake: root winrate, its distance from 50%, score lead, and the spread of score among the leading moves;
- KataGo’s own raw-network predictions of its short-term winrate and score error;

- game phase and the logarithm of visits;
- from the second rung on, what changed since the previous rung: whether the best move changed, how many times it has changed so far, and the change in winrate, top-move share, and confidence margin.

Mikiri stops at rung j when

$$\frac{\hat{r}_j(x)}{v_{j+1} - v_j} < \lambda. \quad (2)$$

The rule comes from a constrained problem. Let a stopping policy π stop position s at rung $\pi(s)$. We want

$$\min_{\pi} \mathbb{E}[r_{\pi(s)}(s)] \quad \text{subject to} \quad \mathbb{E}[v_{\pi(s)}] \leq B. \quad (3)$$

With a multiplier λ on the constraint the problem separates over positions, each minimising $r_k(s) + \lambda v_k$. At rung j the visits already spent cannot be recovered, and continuing by one rung is worthwhile exactly when $\mathbb{E}[r_j - r_{j+1} \mid x] > \lambda(v_{j+1} - v_j)$. The reduction $r_j - r_{j+1}$ is at most r_j , and replacing it with a prediction of the regret remaining gives the rule above. Section 6.6 tests the alternatives this derivation suggests, including predicting the reduction itself, and finds that remaining regret is the best target in practice.

The left side is the regret still on the table per visit needed to reach the next rung. A single λ then trades winrate against visits at the same rate everywhere on the ladder: going from 800 to 3,200 visits requires four times the stake that going from 200 to 800 does. Without this scaling a long ladder spends most of its budget on a handful of positions that no amount of search resolves (Section 6).

The model is trained on the ladder dataset with all non-final rungs pooled. Every offline number we report is cross-fitted: games are split into five folds, and each position is scored by a model that never saw any position from its game. The deployed model is trained on all games and stored as JSON trees evaluated in NumPy.

5.2 Exact search memory

Every search Mikiri finishes within the first 80 moves of a game is stored under the position’s symmetry-canonical key. When a later game reaches the same position, in any of its eight orientations and by any move order, Mikiri plays from the stored search and spends nothing. The stored result is a search of exactly this position, at the budget Mikiri’s own stopping rule chose for it, so a hit gives up nothing in quality.

Entries that keep being hit are worth more than their first search. After each game, if the ledger (below) shows a surplus, Mikiri re-searches the most frequently hit entries at a larger budget and replaces them.

5.3 A ledger

To compare fairly with an engine that spends b visits on every move, Mikiri is granted b visits for each move it makes and may never spend more than it has been granted. Memory hits and early stops leave part of the grant unspent. A controller adjusts λ after every game so that cumulative spending tracks the cumulative grant, which turns visits saved on known or easy positions into deeper searches on hard ones. Post-game deepening draws on the same ledger. Over a series, Mikiri’s total compute, online and offline, is at most the baseline’s.

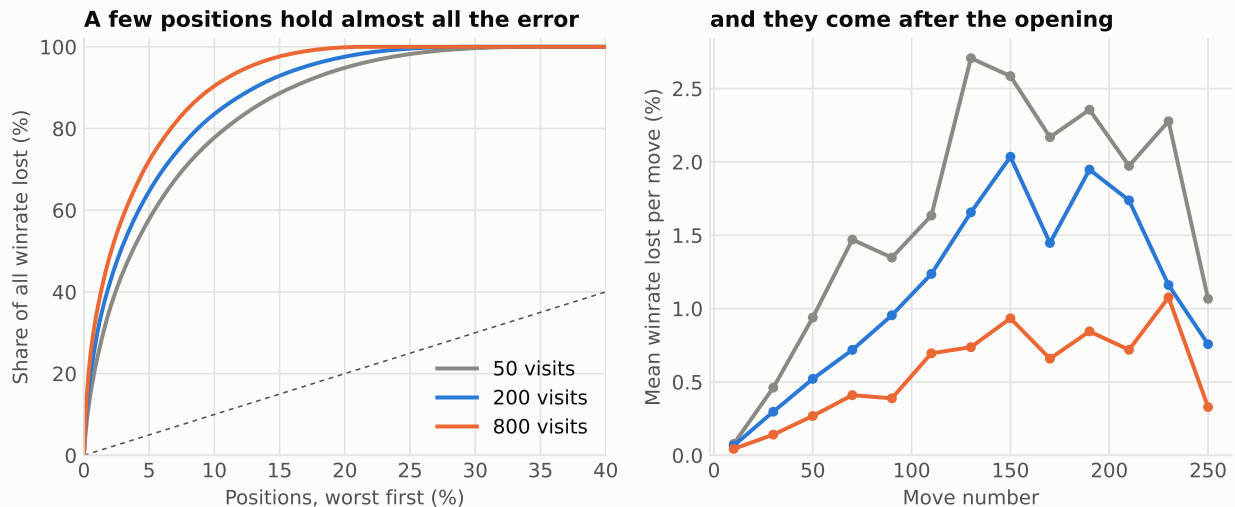


Figure 2: Left: cumulative share of regret held by the worst positions at three budgets. Right: mean regret by move number.

6 What the ladder dataset shows

6.1 Regret is rare and concentrated

Table 1 gives the price of a uniform budget. At 200 visits the engine gives up 0.99% of winrate per move on average, and doubling the budget removes roughly a third of what remains. The average hides the shape. At 200 visits 71% of positions carry no regret at all: the 200-visit move is the best move we scored. The worst tenth of positions hold 84% of the total. Regret also depends on when in the game the position occurs. At 200 visits it averages 0.19% per move over the first 40 moves, 0.74% from move 40 to 100, and 1.4 to 1.6% after move 100. The opening is nearly free of search error at these budgets, which will matter for how we interpret memory.

Figure 2 shows both facts. Figure 3 shows what one of the costly positions looks like: at 50 and at 200 visits the engine chooses a move that an independent search rates at 11%, and at 800 visits it finds one rated at 62%. The stopper’s score is ten and then six times its threshold at the first two rungs, and a fifth of it at the third.

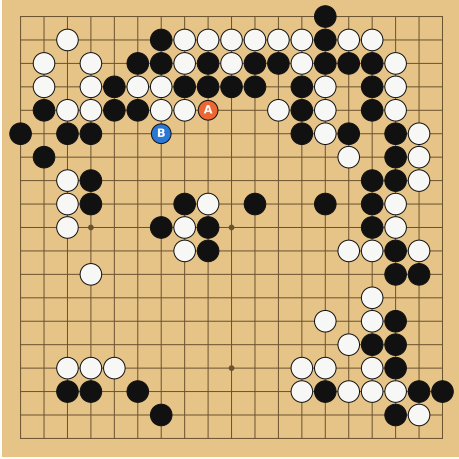
A rule that knew each position’s regret and stopped at the first rung with none would match uniform search at twelve times the visits at a 200-visit budget (last column of Table 3). That is the room available to any stopping rule on this ladder.

6.2 No single signal finds it

Table 2 asks how well individual search statistics rank positions by whether more than 0.5% of winrate remains on the table. The measures of how contested the root is (top move’s visit share, the margin between the best move’s lower confidence bound and its best rival) reach an AUC of about 0.74. KataGo’s own prediction of its short-term winrate error, a signal it uses inside the tree to weight playouts, reaches 0.58 to 0.60 as a predictor of whether the root decision will improve. The classical time-management cue, that the most-visited move is not the highest-valued one [9], is weaker still at 0.59 to 0.61. Contested positions are common, and most of them are contested between moves of nearly equal value, where being wrong costs nothing.

A real position where thinking longer changed the move

Move 123, Black to play



What each rung of the ladder saw

Visits	Engine's move	Search says (winrate)	Independent check	Stopper: stake / cost	Decision
50	A	J15 37.7%	11.0%	$10.2 \times \lambda$	keep thinking
200	A	J15 43.5%	11.0%	$5.6 \times \lambda$	keep thinking
800	B	G14 69.0%	61.6%	$0.2 \times \lambda$	stop and play
3,200	B	G14 53.9%	61.6%	last rung	play

A is what a fixed 200-visit search plays. An independent 1,600-visit search of each move puts B 50.6 points of winrate ahead of A. The stopper's stake estimate stayed above its threshold λ at 50 and at 200 visits, so Mikiri went on to 800 and found B.

Figure 3: A position from the ladder dataset where the stopper keeps searching and the move changes. The threshold λ shown is the one that gives a mean cost of 200 visits. The model here was trained on all games, including this one.

Visits	Mean regret (% winrate)	Positions with no regret (%)	Regret held by worst 10% (%)	Move differs from 3,200-visit move (%)
50	1.47	66	78	33
100	1.19	68	81	30
200	0.99	71	84	27
400	0.70	74	86	22
800	0.48	78	90	17
1,600	0.30	82	94	11
3,200	0.17	86	98	0

Table 1: Uniform search on the ladder dataset (6,000 positions). Regret is measured against the best move scored by an independent forced search.

6.3 A learned stop on a ladder beats uniform search by 1.7 to 1.9 times

Table 3 and Figure 4 give the main offline result. With the 50, 200, 800, 3,200 ladder and a mean cost of 200 visits, the learned stopper gives up 0.75% of winrate per move. Uniform search needs 358 visits to do as well, a multiplier of 1.80. The multiplier stays between 1.69 and 1.88 for mean budgets from 100 to 800 visits. If every restarted search is charged in full, so that reaching the 800-visit rung costs $50 + 200 + 800$, the multiplier is 1.29 to 1.51.

6.4 The ladder and the rate rule do most of the work

Table 4 runs hand-written stopping signals through the same ladder. Two things stand out. First, a flat threshold is harmful for most signals: v1's entropy gate with a flat threshold reaches a multiplier of 0.63 at 200 visits, which is worse than not gating at all, and random stopping is far worse. Lan et al. [16] made the same observation about random stopping. Second, dividing the score by the visits needed to reach the next rung turns the same signals into gains. The LCB margin alone, with no learning, reaches 1.45 at 200 visits.

Signal	50	200	800
Top move’s visit share	0.75	0.74	0.74
LCB margin	0.74	0.73	0.73
v1 entropy gate	0.73	0.73	0.74
KL(visits, prior)	0.65	0.63	0.61
Most-visited is not best-valued	0.60	0.59	0.61
KataGo’s predicted winrate error	0.58	0.58	0.60
Searched minus raw winrate	0.57	0.56	0.56

Table 2: AUC for ranking positions by remaining regret above 0.5%, from the search at 50, 200, and 800 visits.

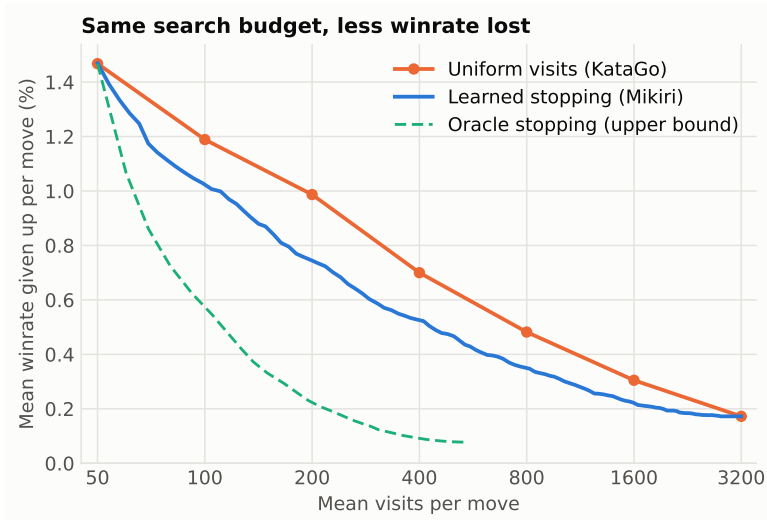


Figure 4: Mean regret against mean visits for uniform search, the learned stopper, and an oracle that knows the true regret.

6.5 Against published stopping rules

How does this compare with what has been published? We re-implemented ten stopping rules from their papers and source code and ran them on the same data, where every rule sees the same root statistics and is scored by the same forced values. To give each rule its natural setting we use the full doubling ladder, so a rule may stop at 50, 100, 200, and so on up to 3,200 visits. Table 5 gives the results.

The rules fall into four groups. *BEHIND* and *UNST* search longer when the player is losing or when the most-visited move is not the highest-valued one [9]. *STOP* ends a search once the runner-up cannot overtake the leader in the visits that remain, or in a fraction p of them, and *CLOSE* extends it when the top two moves are close in visits [1]. *Smart pruning* is Leela Chess Zero’s form of *STOP*, which ships with a stop on the per-visit KL gain of the visit distribution [18]. Two rules come with a statistical rationale: the *VOI* rule stops when a bound on the value of further samples falls below their cost [8], and the *BAI* rule stops when the leader’s confidence interval separates from every rival’s [14]. Three are learned. The *state-only* rule predicts regret from what is known before any search, in the spirit of Muppidi et al. [20]. For *DS-MCTS* [16] we train a classifier on root statistics to the authors’ label $U(s, n)$, in place of their board-input networks. For *V-MCTS* [32] we run the authors’ virtual expansion on KataGo’s root priors, visits,

Mean visits	Uniform regret (%)	Stopper regret (%)	Equivalent uniform visits	Multiplier continuation	restart	Oracle multiplier
100	1.19	1.03	173	1.76	1.51	5.5
150	1.07	0.87	265	1.79	1.47	9.6
200	0.99	0.75	358	1.80	1.44	12.1
300	0.82	0.60	544	1.88	1.47	10.9
400	0.70	0.53	692	1.75	1.36	8.1
600	0.57	0.41	1072	1.80	1.38	5.7
800	0.48	0.35	1332	1.69	1.29	5.7

Table 3: Learned stopping on the 50/200/800/3,200 ladder, cross-fitted by game. *Equivalent uniform visits* is the fixed budget with the same mean regret. The oracle stops with knowledge of the true regret.

Stopping signal	Flat threshold		Rate rule	
	200 visits	400 visits	200 visits	400 visits
Random stopping	0.44	0.29	0.71	0.69
Most-visited is not best-valued	0.55	0.60	0.58	0.77
v1 entropy gate	0.63	0.76	1.16	1.28
LCB margin	1.13	0.86	1.45	1.00
Learned stopper	1.41	1.22	1.82	1.77

Table 4: Compute multipliers (continuation cost) for stopping signals on the 50/200/800/3,200 ladder. Values below 1 are worse than uniform search.

and winrates with their published $r = 0.2$ and $\epsilon = 0.1$. The last two are adaptations to a different engine, and we do not claim they reproduce the authors’ systems.

Every rule in the table is charged continuation cost, the size of the deepest search, which is what a rule living inside the engine pays. Section 3 showed that a restarted search costs about the same in network evaluations. A reader who prefers to charge Mikiri for every restart should compare its four-rung figure of 1.44 at 200 visits with the 1.60 of V-MCTS, which stops inside a single search and has no restarts to charge; the matches in Section 7 settle the question in network evaluations.

Three things stand out. First, the rules with a stopping guarantee are far worse than uniform search here (0.34 and 0.77 at 200 visits). They stop on confidence in the leading move, and that confidence says little about whether the game is still in doubt. Second, V-MCTS is the strongest published rule, at 1.60, and Mikiri at 2.13 exceeds it by 0.54 [0.28, 0.84] at 200 visits and by 0.35 [0.09, 0.65] at 400, as paired differences over the same resampled games. Third, the rate rule improves every signal it can be applied to. It lifts the DS-MCTS classifier from 1.48 to 1.75, and at 400 visits that combination ties Mikiri (difference 0.03 [-0.16, 0.27]). Mikiri’s own model with a flat threshold reaches only 1.40. The price on the next rung matters more than which learned signal it is applied to.

6.6 What the model should predict

The rate rule scores a position by its remaining regret, an upper bound on what continuing can recover. The derivation in Section 5 names other quantities more directly: the gain from the next rung, $r_j - r_{j+1}$; the gain from the whole ladder, $r_j - r_m$; and the best recovery per visit over any continuation, $\max_{k>j} (r_j - r_k) / (v_k - v_j)$. Table 6 tries each one twice: as an oracle that reads the

Rule	As published, 200	As published, 400	With rate rule, 200
VOI stop [8]	0.34 [0.30, 0.40]	0.24 [0.20, 0.31]	1.05 [0.86, 1.21]
BAI stop [14]	0.77 [0.63, 1.01]	0.59 [0.50, 0.71]	1.18 [1.09, 1.29]
BEHIND [9]	0.89 at 174	0.89 at 348	–
STOP, $p = 0.45$ [1]	1.25 at 134	1.19 at 512	–
Smart pruning [18]	1.17 at 159	1.21 at 308	–
CLOSE [1]	1.15 at 229	1.11 at 453	–
UNST [9]	1.21 at 239	1.17 at 475	–
KLD gain [18]	1.32 [1.18, 1.49]	1.23 [1.02, 1.42]	–
State only [20]	1.18 [0.97, 1.40]	1.02 [0.89, 1.23]	1.54 [1.35, 1.74]
DS-MCTS [16]	1.48 [1.28, 1.73]	1.48 [1.29, 1.73]	1.75 [1.60, 1.92]
V-MCTS [32]	1.60 [1.44, 1.76]	1.54 [1.36, 1.70]	–
LCB margin	1.14 [0.95, 1.36]	0.82 [0.70, 0.97]	1.45 [1.24, 1.70]
Mikiri	1.40 [1.21, 1.63]	1.35 [1.15, 1.61]	2.13 [1.88, 2.41]

Table 5: Compute multipliers on the doubling ladder, with 95% bootstrap intervals over games. Rules with a threshold are swept and shown at their best point for each mean budget. Rules without one are shown at the configuration nearest the budget, with its mean cost in visits. The last column applies the rate rule to the same signal where that is possible.

true value, and as the target of the same model, cross-fitted in the same way. As a ceiling it also gives the hindsight optimum, which picks for each position the rung minimising $r_k + \lambda v_k$ with every rung known.

With perfect information the best-gain-per-visit index reproduces the hindsight optimum (14.11 against 14.16), so it is the right quantity in principle. It is also the hardest of the four to learn: fitted from search statistics it reaches 1.63 to 1.87, behind remaining regret under the rate rule at 1.83 to 2.14. The gain from the next rung is the weakest choice on both counts. As an oracle it is myopic, because it stops whenever one more rung does not help even if two more would, and as a target it is noisy, because a difference of two regrets has more variance than either. The gain from the whole ladder matches remaining regret on short ladders and falls slightly behind on the longest.

The three rows for remaining regret isolate the price. With no price the multiplier is 1.41 to 1.56. Dividing by the visits to the next rung gives 1.83 to 2.14. Dividing by the visits to the top of the ladder, which charges for a commitment the rule never makes, is worse than no price at all. The rate rule works because it pairs a target that search statistics can predict with the one cost that is known exactly.

6.7 Prior rules’ statistics add nothing as inputs

The published rules each reduce the root statistics to one number. If those numbers carried information that Mikiri’s 30 features miss, feeding them to the regressor should help. On the doubling ladder we added, one at a time and then together, the KLD gain, the VOI bound, the BAI gap, and the virtual-expansion distance of V-MCTS. The multiplier at 200 visits moved from 2.14 to between 2.07 and 2.17, and no paired difference at any budget was distinguishable from zero in the direction of a gain (the largest was +0.11, interval -0.00 to $+0.26$, for the BAI gap at 400 visits; all

Scored	Divided by	Oracle,	Learned, 200 visits		
		100 visits	4	6	7 rungs
r_j	1 (flat)	5.99	1.41	1.56	1.51
r_j (rate rule)	$v_{j+1} - v_j$	8.69	1.83	1.98	2.14
r_j	$v_m - v_j$	5.86	1.35	1.50	1.39
$r_j - r_{j+1}$	$v_{j+1} - v_j$	4.96	1.50	1.54	1.44
$r_j - r_m$	$v_{j+1} - v_j$	9.07	1.79	1.98	2.04
$\max_{k>j} \frac{r_j - r_k}{v_k - v_j}$	built in	14.11	1.63	1.87	1.77
Hindsight optimum		14.16	–	–	–

Table 6: Compute multipliers for different quantities to score at rung j and prices to divide them by (m is the top rung). The oracle column reads the true quantity on the four-rung ladder at a mean cost of 100 visits (at 200 the oracles reach the top of the measurable range). The learned columns fit each quantity with the same model and report a mean cost of 200 visits on three ladders, averaged over three fold assignments.

four together gave -0.10 at 200). The rules differ in what they do with the root statistics, and the statistics themselves are exhausted. This agrees with Section 9: most of the regret that survives sits on moves the search barely visited, where no root statistic can see it.

6.8 The model needs little

Appendix A reports ablations and training diagnostics. Removing the trajectory features, removing KataGo’s raw error predictions, or keeping only seven features changes the multiplier at 200 visits by at most 0.09. A model of 80 depth-2 trees does nearly as well as one of 400 depth-3 trees. Trained on 300 labeled positions the stopper already reaches 1.50, against 1.82 with 4,800. Two choices matter. Fitting the square root of regret instead of regret raises the multiplier from 1.74 to 1.82, because a few very large blunders otherwise dominate the fit. Replacing the rate rule with a flat threshold lowers it to 1.41.

7 Matches

All matches in this section are played against KataGo at a fixed budget, from the empty board, with alternating colours and KataGo’s default move sampling for both players unless noted. Table 7 lists every match we ran.

7.1 What a visit is worth

To read the other results we first need the exchange rate between compute and strength in this setting. KataGo against itself at 200 visits scores 50.2% over 400 games ($+2$ Elo, -30 to $+35$), which checks that the harness has no side bias. Against the 200-visit engine, KataGo at 100 visits rates -220 Elo, at 283 visits $+108$, and at 400 visits $+237$. A doubling is worth about 230 Elo here. In network evaluations the uniform engine runs 40, 76, 98, and 132 per move at the four budgets: fewer than its visits, because consecutive searches share cached evaluations.

Player	Opponent	Games	W-L-D	Elo (95% interval)	Visits	Restart	Evals
KataGo 100 visits	KataGo 200	400	77-301-22	-220 (-262 to -181)	100	100	40/71
KataGo 200 visits	KataGo 200	400	185-183-32	+2 (-30 to +35)	200	200	76/76
KataGo 283 visits	KataGo 200	400	247-126-27	+108 (+76 to +143)	283	283	98/73
KataGo 400 visits	KataGo 200	400	310-73-17	+237 (+198 to +280)	400	400	132/73
Mikiri (stopper), 100-visit grant	KataGo 100	400	262-119-19	+130 (+95 to +167)	100	117	50/40
DS-MCTS-style rule (Lan et al. 2021) in the same player, 200-visit grant	KataGo 200	600	395-157-48	+146 (+117 to +176)	200	351	105/77
Mikiri (stopper + memory), 200-visit grant	KataGo 200	1200	908-219-73	+227 (+204 to +250)	201	255	109/76
Mikiri (stopper), 200-visit grant, six-rung ladder	KataGo 400	600	268-271-61	-2 (-27 to +24)	200	303	102/143
Mikiri (stopper), 200-visit grant, six-rung ladder	KataGo 200	1000	750-175-75	+228 (+203 to +254)	201	304	102/78
Mikiri (stopper), 175-visit grant	KataGo 200	600	432-129-39	+193 (+163 to +225)	175	217	89/76
Mikiri (stopper), 200-visit grant	KataGo 200	1000	728-196-76	+206 (+181 to +232)	200	251	99/74
Mikiri (stopper), 160-visit grant	KataGo 200	1000	586-325-89	+93 (+72 to +114)	160	197	80/74
Mikiri (memory), 200-visit grant	KataGo 200	600	276-292-32	-9 (-37 to +19)	158	158	69/69
Mikiri (stopper), 200-visit grant, paired openings, no sampling	KataGo 200	800	561-183-56	+178 (+153 to +205)	200	250	115/96
Mikiri (hand-written LCB rule), 200-visit grant	KataGo 200	600	349-201-50	+88 (+61 to +115)	200	250	100/75
Mikiri (stopper), 200-visit grant, same ladder as V-MCTS	KataGo 200	600	404-160-36	+150 (+121 to +180)	200	350	92/74
V-MCTS rule (Ye et al. 2022) in the same player, 200-visit grant	KataGo 200	600	395-173-32	+135 (+108 to +164)	200	350	87/72

Table 7: All matches. *Visits* is the mean size of the deepest search per move, *Restart* counts every search requested, and *Evals* is network evaluations per move for the player and for its opponent, read from each engine’s log.

7.2 The stopping rule at equal visits and below

With the ladder 50, 200, 800, 3,200 and the controller holding its mean at the opponent’s budget, Mikiri scores 76.6% over 1,000 games (728–196–76), a gain of +206 Elo (95% interval +181 to +232), at 200 visits per move. Counting every restarted search it requested 251 visits per move, and it ran 99 network evaluations per move to KataGo’s 74.

A longer ladder does at least as well. With six rungs (50, 200, 400, 800, 1,600, 3,200), the best ladder offline by continuation cost, Mikiri scores 78.8% over 1,000 games (750–175–75), a gain of +228 Elo (+203 to +254), at 201 visits per move. KataGo gains +237 from doubling its budget, so the rule recovers nearly all of a doubling.² The finer ladder restarts more often, so it requested 304 visits per move with restarts counted, and it ran 102 evaluations per move to KataGo’s 78. We also played the six-rung ladder at a 200-visit grant directly against KataGo at 400 visits. Over 600 games the result is level: 268–271–61, –2 Elo (–27 to +24), with Mikiri running 102 network evaluations per move to KataGo’s 143. Half the visits and about 70% of the evaluations are enough to hold the stronger setting to a draw, as the two matches against the 200-visit engine imply (+228 against +237).

The two ladders’ intervals overlap, so the match does not separate them, though the ladder dataset ranks the six-rung ladder higher (Table 8). Unless we say otherwise, the analyses below use the four-rung match, which has the most games and a full trace.

Visits understate Mikiri’s cost in one respect. A fixed-budget engine re-meets much of its previous tree on the next move and finds those evaluations cached. Mikiri’s searches vary in size, so less carries over, and it runs more network evaluations per visit. We therefore reran the match with Mikiri granted fewer visits. At a grant of 160 it averaged 160 visits per move, 197 with every restarted search counted, and 80 network evaluations to KataGo’s 74. It scored 63.0% over 1,000 games (586–325–89), a gain of +93 Elo (+72 to +114). At a grant of 140 it averaged ?? visits, ?? restart visits, and ?? evaluations to KataGo’s ??, and gained ?? Elo (?? to ??) over ?? games.

Figure 5 places these results against the uniform curve under all three measures of compute. Every Mikiri run lies above the curve under each. The figure also shows the prior rules of Section 7.4. The comparison at equal evaluations is direct: Mikiri in the main match and KataGo at 283 visits run 99 and 98 evaluations per move and gain +206 and +108 Elo.

²After 600 games this match stood at +263. The first 600 games ran above the long-run rate, and the figure here is for all 1,000.

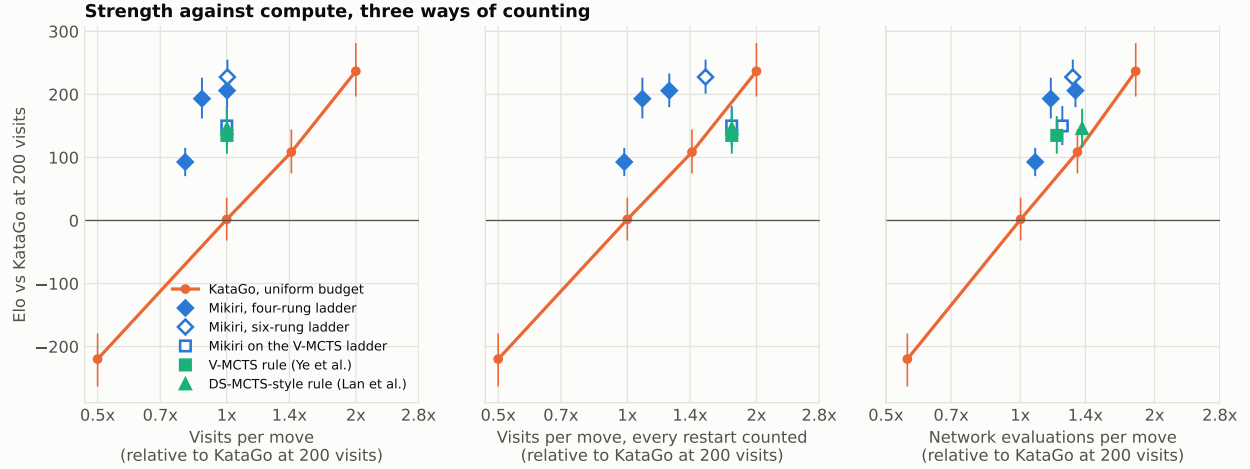


Figure 5: Elo against KataGo at 200 visits, as a function of compute relative to that baseline, counted three ways. The line is KataGo at uniform budgets. Bars are 95% intervals.

Rule	Mean visits per move	Offline regret (% winrate)	Equivalent uniform visits	Predicted Elo	Measured Elo (95% interval)
Mikiri, 4 rungs	160	0.82	297	+126	+93 (+72 to +114)
Mikiri, 4 rungs	175	0.79	325	+160	+193 (+163 to +225)
Mikiri, 4 rungs	200	0.74	360	+197	+206 (+181 to +232)
Mikiri, 6 rungs	201	0.71	387	+225	+228 (+203 to +254)
Mikiri, short ladder	200	0.75	358	+196	+150 (+121 to +180)
V-MCTS	200	0.80	312	+145	+135 (+108 to +164)
DS-MCTS-style	200	0.82	299	+128	+146 (+117 to +176)

Table 8: Elo forecast from the offline regret analysis, against the Elo measured in play, for stopping rules against KataGo at 200 visits.

7.3 The offline analysis predicted the result

The ladder dataset gives a forecast that can be checked. For each run we take the mean visits Mikiri actually used, read the cross-fitted regret at that cost from Table 3, convert it to equivalent uniform visits, and read the Elo of that uniform budget from the measured curve. Table 8 compares the forecast with the match.

For the four-rung match at 200 visits the forecast is within nine Elo of the result, and for the six-rung ladder within three. At 160 visits it is about 30 Elo too high, outside the measured interval. For the two prior rules of Section 7.4, forecast from their own calibration on the ladder dataset, it is ten Elo too high and 18 too low, both inside the intervals. Its one clear miss is Mikiri’s rule on the short ladder of Section 7.4, which it rates at +196 and which gained +150. That ladder stops at 400 visits, so holding a mean of 200 leaves it close to its ceiling, where a small shift between dataset and games costs more. The forecast orders Mikiri’s three main configurations correctly and puts both prior rules below them, as measured. It has the two prior rules 17 Elo apart in one order and the matches have them 11 apart in the other, a difference neither can resolve. Regret measured on self-play positions is a usable guide to match strength, good to a few tens of Elo, which makes the ladder dataset a cheap way to choose among stopping rules before playing any games.

7.4 Prior rules in play

Section 6.5 compares ten published stopping rules offline. We packaged the two strongest so that they run in the same player, under the same controller, at the same 200-visit budget. The V-MCTS rule [32], with the authors’ constants and their ladder of checkpoints up to 400 visits, gains +135 Elo (+108 to +164) over 600 games at 200 visits per move. The DS-MCTS-style classifier [16] gains +146 Elo (+117 to +176) over 600 games.

The DS-MCTS-style rule uses the full doubling ladder and ran 105 evaluations per move to KataGo’s 77, slightly more than Mikiri’s six-rung ladder (102), for about 80 Elo less. Mikiri is ahead of it on every measure of compute.

At equal visits Mikiri is also ahead of V-MCTS: +206 and +228 against +135, with intervals that do not overlap. Network evaluations tell a closer story. V-MCTS ran 87 evaluations per move to KataGo’s 72, and Mikiri’s four-rung ladder ran 99. Mikiri’s ladder reaches 3,200 visits, and its deepest searches meet fewer cached positions than searches capped at 400. Interpolating between Mikiri’s matches at 160 and 200 visits (80 and 99 evaluations, +93 and +206 Elo) puts it at about +135 at 87 evaluations, level with V-MCTS. Two further matches test this directly. In the first, Mikiri’s stopper runs on V-MCTS’s own ladder (50, 100, 200, 400), so that the two rules differ in nothing else. It gains +150 Elo (+121 to +180) over 600 games at 92 evaluations per move, against +135 at 87 for V-MCTS. The intervals overlap almost entirely: on this ladder the two rules cannot be told apart, although the ladder dataset rated Mikiri’s rule higher (1.80 to 1.60). In the second, the four-rung ladder is granted 175 visits to bring its evaluations close to those of V-MCTS: it gains +193 Elo (+163 to +225) over 600 games at 89 evaluations per move, against V-MCTS’s +135 (+108 to +164) at 87. The intervals touch at one end.

The picture is therefore this. Mikiri’s advantage over V-MCTS at equal visits is real, and it comes from ladders that reach 3,200 visits, which the rate rule makes affordable and which V-MCTS does not exploit (offline, its best multiplier over all maximum budgets is 1.55 to 1.60). On V-MCTS’s own short ladder the two rules are level. Those deep searches cost more evaluations per visit, which closes part of the gap: at matched evaluations Mikiri is ahead by about 60 Elo, with less certainty than its lead per visit.

One feature of these matches we cannot explain from the data: between grants of 160 and 175 visits the four-rung ladder gains 100 Elo (+93 to +193, intervals that do not overlap), and between 175 and 200 only 13 more. The ladder dataset forecasts a smoother curve (+126, +160, +197). The two runs use different seeds and nothing else differs in their configuration.

7.5 In-game ablations

Two hand-written rules were played on the same ladder at the same budget. The LCB margin with the rate rule, which reaches 1.45 offline with no learning, gains +88 Elo (+61 to +115) over 600 games. The entropy gate of the first version with a flat threshold, 0.63 offline, gains ?? Elo (?? to ??) over ?? games.

7.6 Without move sampling

KataGo’s default play samples among well-visited moves early in the game. A search that stops when one move dominates has sharper visit counts than a fixed-budget search, so sampling could favour Mikiri for reasons unrelated to search quality. To rule this out we sampled 400 balanced openings of 12 moves, played each twice with colours swapped, and had both players always play their engine’s top move. Mikiri scores 73.6% over 800 games (561–183–56), +178 Elo (+153 to +205), at 200 visits per move.

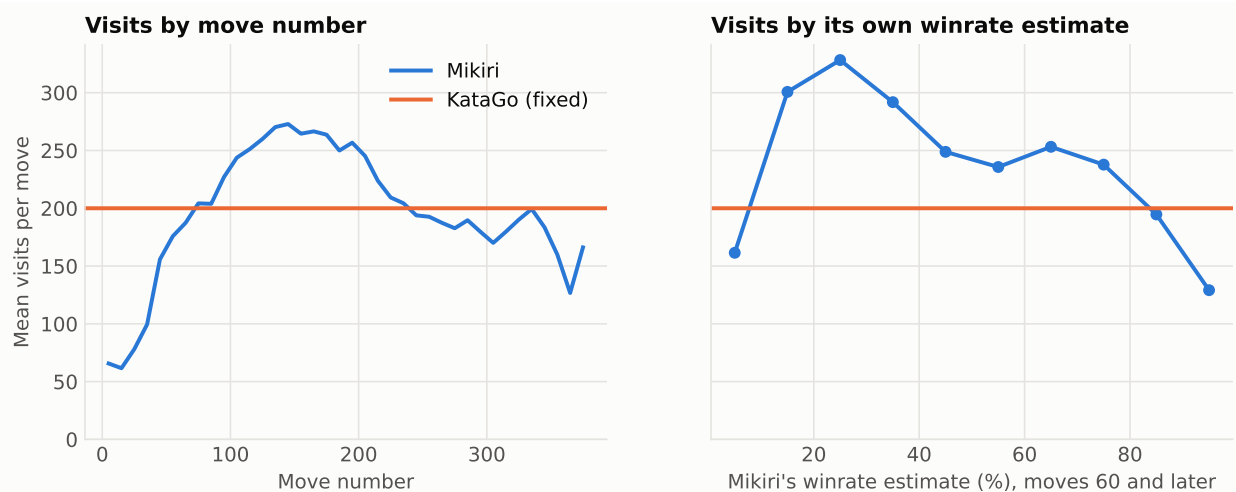


Figure 6: Mean visits per Mikiri move in the main match, by move number and, for moves 60 and later, by Mikiri’s own winrate estimate.

7.7 Where the visits go

Figure 6 shows Mikiri’s spending in the main match. It uses about a third of the budget over the first 20 moves, where Section 6 found almost no regret to recover, and about 270 visits per move around move 140. After move 60 it also spends by stakes: about 130 visits per move when it rates its own winrate above 90%, about 160 when below 10%, and 190 to 330 in between, with the peak when it is behind but not lost. Neither behaviour was programmed. Both follow from predicting regret in winrate units. Of all its moves, 47% stopped at 50 visits, 42% at 200, 11% at 800, and 0.2% at 3,200.

The gain does not depend on colour. In the main match Mikiri scored 76.2% as Black and 77.0% as White. Figure 7 shows when it arrives: Mikiri’s mean winrate estimate stays at 50% for the first 70 moves, where it is spending a third of the budget, and climbs steadily through the middle game.

7.8 Memory

With memory alone and no stopper, Mikiri is KataGo with an opening book that fills itself. Over 600 games it served 21.2% of its moves from memory (23.1% in the second half of the series, from 8,940 stored positions), averaged 158 visits per move, and scored 48.7% (−9 Elo, −37 to +19). Strength is unchanged, as it should be when a hit returns the search that would have been run, and a fifth of the visits are saved.

The saving does not carry over to network evaluations. Mikiri ran 69 evaluations per move and KataGo 69. KataGo keeps a cache of network evaluations for as long as its process lives, so a baseline that plays hundreds of games in one process has already paid for the positions that recur: its opening searches cost visits but few new evaluations. Exact memory removes search work that the evaluation cache had already made cheap. It would save evaluations for an engine that restarts between games, and it saves the tree-search overhead in any case, but in this harness its effect on the dominant cost is nil.

The full system combines memory with the stopper and lets the controller reinvest what memory saves. Over 1,200 games it scores 78.7% (908–219–73), +227 Elo (+204 to +250), at 201 visits and 109 evaluations per move against 76, with 20.8% of moves served from a memory that grew

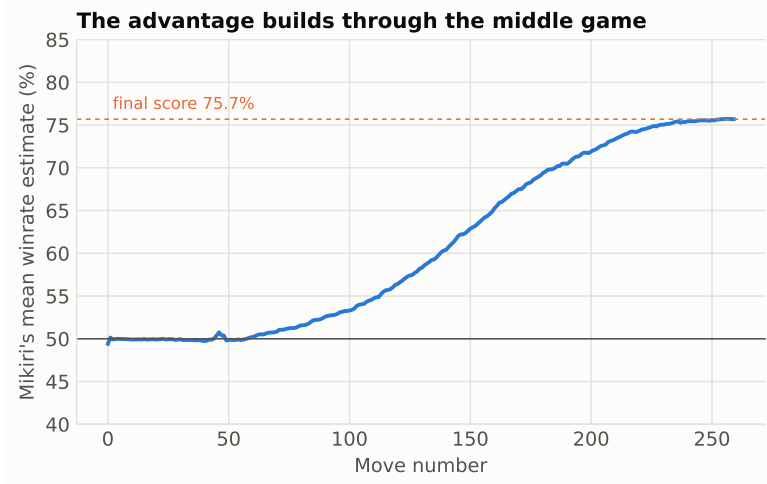


Figure 7: Mikiri’s mean winrate estimate by move number in the main match. Finished games are carried forward at their result.

to 20,087 positions. The stopper alone gained +206 (+181 to +232) at 99 evaluations per move. The intervals overlap, and the reinvested visits are new searches that the evaluation cache does not cover, which is why evaluations rise. Memory adds little on top of the stopper in this setting.

7.9 Other budgets, another network, other boards

The same model, trained once on 19×19 positions from the b18 network, was run without retraining in further settings.

- **Budgets.** Against KataGo at 100, 400, and 800 visits, at matched visits, Mikiri gains +130 (+95 to +167), ?? (?? to ??), and ?? (?? to ??) Elo. At 100 visits the first rung of the ladder is already half the budget, which leaves the rule less to reallocate than at 200.
- **A stronger network.** On the 28-block network, with both players using it, Mikiri gains ?? (?? to ??) Elo.
- **Smaller boards.** On 13×13 and 9×9 it gains ?? (?? to ??) and ?? (?? to ??) Elo.

8 What a frozen engine can and cannot retrieve

8.1 Exact positions recur often

How much could an exact-match memory ever serve? We replayed the 300 self-play games behind the ladder dataset in order and asked, for each position, whether its symmetry-canonical key had already occurred in an earlier game. Figure 8 shows the answer for the second half of the games, when the memory holds 150 to 300 games. A position at move 10 had been seen before 86% of the time, at move 20 51% of the time, and at move 30 30% of the time. Games stayed on known ground for 24 moves on average, and one for 108. Over whole games, 11.5% of all positions were repeats.

These games used a higher early move temperature (0.8) than KataGo’s default (0.5), so they are more varied than default play. The reuse rate is large for two reasons: a strong engine’s opening

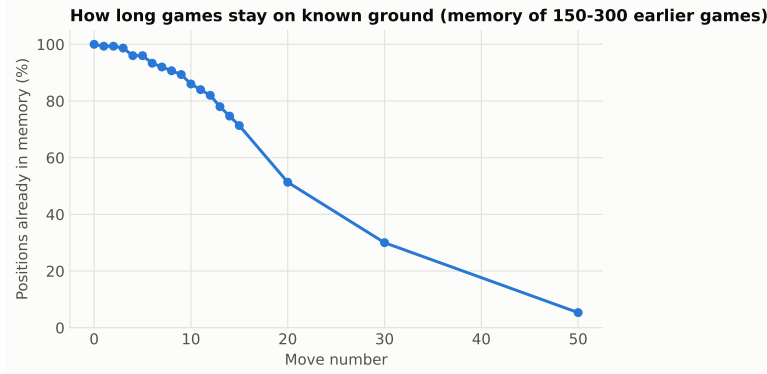


Figure 8: Share of positions whose canonical key had already occurred in an earlier game, by move number, with a memory of 150 to 300 earlier games.

choices are concentrated on a few lines, and canonicalising over the eight symmetries and over move order merges what look like different games.

8.2 Near matches do not transfer

The first version of this project, DataGo, was designed around approximate retrieval: find stored positions that resemble the current one and reuse their analysis. We tested the idea directly. Each position in the ladder dataset is embedded with KataGo’s raw policy and ownership maps, compared against every position from other games in all eight orientations, and matched to its nearest neighbours by cosine similarity. A neighbour’s 3,200-visit move is mapped through the aligning symmetry and offered as a proposal.

Proposals matter in the 1,605 positions (27%) where the 200-visit move differs from the 3,200-visit move. There the nearest neighbour’s move is the 3,200-visit move 5.5% of the time, and one of five neighbours has it 7.8% of the time. The policy network’s highest-prior alternative to the 200-visit move is right 43% of the time, and one of its top five 85%. An index of stored searches is a much worse source of candidate moves than the network the engine already carries.

Figure 9 shows why. A neighbour’s best move is this position’s best move in 96% of cases when the similarity exceeds 0.999, which in practice means the same position. Agreement is 86% between 0.97 and 0.999, 27% between 0.90 and 0.97, 11% between 0.8 and 0.9, and under 4% below that. After move 20 the median nearest-neighbour similarity is 0.75 or lower. Retrieval is useful exactly where the exact key already finds the position, and nowhere else.

The same holds for deciding when to think. The similarity-weighted regret of a position’s five nearest neighbours predicts its own regret with an AUC of 0.55. The search’s own confidence margin reaches 0.73.

We do not claim that approximate retrieval cannot help Go engines. Humphreys et al. [10] show that it can when the network is trained to read its neighbours and the index holds tens of millions of positions. Our index holds 6,000, and our network is frozen. Within those limits the result is unambiguous, and it matches what is known about Go: Lan et al. [17] change a strong engine’s move by adding two stones that should not matter.

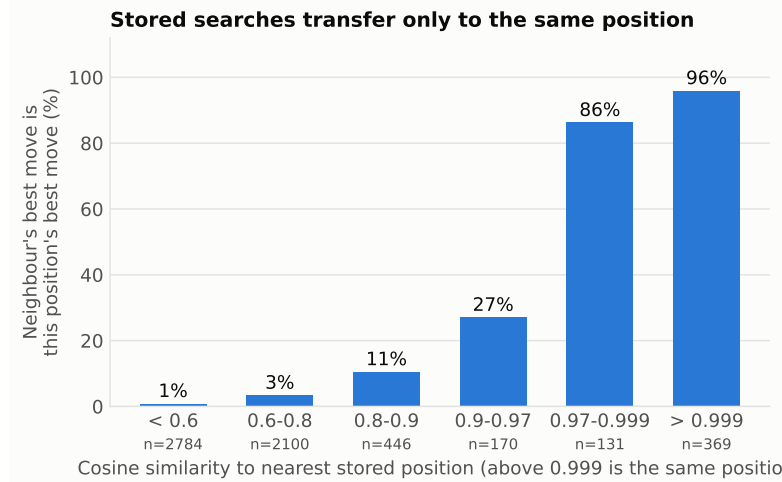


Figure 9: How often the nearest stored position’s 3,200-visit move is also this position’s, by embedding similarity. Numbers under the bars are position counts.

9 Limits

One engine family, low budgets. All results use KataGo networks at mean budgets of 100 to 800 visits per move. Elo per doubling of visits is steep in this range (about 230 at 200 visits in our matches), which is what makes reallocation valuable. At tournament budgets of tens of thousands of visits the curve is flatter, errors are rarer, and we have not measured what remains to be gained.

The yardstick is a search. Regret is defined by 1,600-visit forced searches, which are themselves imperfect. They are applied identically to every move and to every policy, so comparisons between policies are fair, but the absolute regret numbers inherit the engine’s blind spots. The match results do not depend on the yardstick.

Most of the headroom is out of reach of search statistics. The oracle multiplier at 200 visits is 12; the learned stopper reaches 1.8. Among positions with more than 0.5% regret at 200 visits, 57% of the regret sits on a best move that had received under 10% of the visits, and 12% on one that had received under 1%. A stopping rule that reads the search cannot see a move the search has not looked at. Closing this gap needs a different source of candidates. Section 8.2 shows that a similarity index is not one.

Memory is an opening book. Exact reuse is confined to the first few dozen moves, where search error at these budgets is already small. It saves visits, and in a long-running engine process it saves almost no network evaluations, since the engine’s own cache covers the same positions. Against opponents with a wider opening repertoire than KataGo’s, hit rates will be lower than ours.

The opponent is cooperative. Adversarial policies defeat KataGo at any budget we could afford [26, 27]. Mikiri reads the same network and the same search, and would stop early with full confidence in exactly the positions those attacks construct.

No direct comparison with AlphaGo. AlphaGo and AlphaGo Zero [23, 24] are not available to play. KataGo’s public networks are among the strongest Go engines anyone can run, so they are the baseline here. What we show is an improvement over that baseline at equal compute.

10 Conclusion

A frozen Go engine leaves most of a doubling of search on the table by spending visits uniformly. A small learned rule recovers it, and the part of the rule that matters most is the price it puts on continuing: divide the stake by the cost of the next rung, and signals that were worse than no gate become gains. The same engine cannot be helped by analyses of similar positions, and an exact memory of its own past searches saves visits that its evaluation cache had already made cheap.

We think the more general lesson is about measurement. A search cannot grade itself. Once a yardstick independent of the search exists, questions that looked settled, such as whether a modern engine has anything left to gain from time management, turn out to have numbers attached, and the numbers can be checked against games before the games are played. The testbed, the dataset, every model and every game record are released so that others can check ours.

Acknowledgments

The first version of this project, DataGo, was a team project with Benjamin Huh, Jason Peng, Olir Eswaramoorthy, David Roos, and Victor Lun Pun, whose questions about uncertainty gating and reuse of deep searches this work takes up. Its code and report are preserved in the repository under `legacy/`. Experiments ran on the Dartmouth LISP Lab’s machines. KataGo and its networks are the work of David J. Wu and the KataGo community.

References

- [1] Hendrik Baier and Mark H. M. Winands. Time management for Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):301–314, 2016.
- [2] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z. Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.
- [3] Guillaume M. J-B. Chaslot, Jean-Baptiste Hooek, Julien Perez, Arpad Rimmel, Olivier Teytaud, and Mark H. M. Winands. Meta Monte-Carlo tree search for automatic opening book generation. In *IJCAI Workshop on General Intelligence in Game Playing Agents*, 2009.
- [4] Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and move groups in Monte Carlo tree search. In *IEEE Symposium on Computational Intelligence and Games*, pages 389–395, 2008.
- [5] Johannes Czech, Patrick Korus, and Kristian Kersting. Improving AlphaZero using Monte-Carlo graph search. In *International Conference on Automated Planning and Scheduling*, pages 103–111, 2021. arXiv:2012.11045.
- [6] Andrew Drozdov, Shufan Wang, Razieh Rahimi, Andrew McCallum, Hamed Zamani, and Mohit Iyyer. You can’t pick your neighbors, or can you? When and how to rely on retrieval in the kNN-LM. In *Findings of EMNLP*, 2022. arXiv:2210.15859.

- [7] Anirudh Goyal, Abram L. Friesen, Andrea Banino, Theophane Weber, Nan Rosemary Ke, Adrià Puigdomènech Badia, Arthur Guez, Mehdi Mirza, Peter C. Humphreys, Ksenia Konyushkova, Laurent Sifre, Michal Valko, Simon Osindero, Timothy Lillicrap, Nicolas Heess, and Charles Blundell. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pages 7740–7765, 2022. arXiv:2202.08417.
- [8] Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In *Uncertainty in Artificial Intelligence*, 2012. arXiv:1207.5879.
- [9] Shih-Chieh Huang, Rémi Coulom, and Shun-Shii Lin. Time management for Monte-Carlo tree search applied to the game of Go. In *International Conference on Technologies and Applications of Artificial Intelligence*, pages 462–466, 2010.
- [10] Peter C. Humphreys, Arthur Guez, Olivier Tieleman, Laurent Sifre, Théophane Weber, and Timothy Lillicrap. Large-scale retrieval for reinforcement learning. In *Advances in Neural Information Processing Systems*, 2022. arXiv:2206.05314.
- [11] KataGo project. KataGo parallel analysis engine. https://github.com/lightvector/KataGo/blob/master/docs/Analysis_Engine.md, 2026. v1.18.2, accessed 2026-10-01.
- [12] KataGo project. Monte-Carlo graph search from first principles. <https://github.com/lightvector/KataGo/blob/master/docs/GraphSearch.md>, 2026. Accessed 2026-10-01.
- [13] KataGo project. Other methods implemented in KataGo. <https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md>, 2026. Accessed 2026-10-01.
- [14] Emilie Kaufmann and Wouter M. Koolen. Monte-Carlo tree search by best arm identification. In *Advances in Neural Information Processing Systems*, 2017. arXiv:1706.02986.
- [15] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*, 2020. arXiv:1911.00172.
- [16] Li-Cheng Lan, Meng-Yu Tsai, Ti-Rong Wu, I-Chen Wu, and Cho-Jui Hsieh. Learning to stop: Dynamic simulation Monte-Carlo tree search. In *AAAI Conference on Artificial Intelligence*, 2021. arXiv:2012.07910.
- [17] Li-Cheng Lan, Huan Zhang, Ti-Rong Wu, Meng-Yu Tsai, I-Chen Wu, and Cho-Jui Hsieh. Are AlphaZero-like agents robust to adversarial perturbations? In *Advances in Neural Information Processing Systems*, 2022. arXiv:2211.03769.
- [18] Leela Chess Zero contributors. Lc0 options. <https://github.com/LeelaChessZero/lc0/wiki/Lc0-options>, 2026. Accessed 2026-10-01.
- [19] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Annual Meeting of the Association for Computational Linguistics*, 2023. arXiv:2212.10511.
- [20] Aneesh Muppidi, Firas Darwish, Dylan Cope, João F. Henriques, and Jakob Nicolaus Foerster. Finding the time to think: Learning planning budgets in real-time RL. *arXiv preprint arXiv:2606.26463*, 2026.

- [21] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *International Conference on Machine Learning*, pages 2827–2836, 2017. arXiv:1703.01988.
- [22] Julian Schrittwieser, Thomas Hubert, Amol Mandhane, Mohammadamin Barekatain, Ioannis Antonoglou, and David Silver. Online and offline reinforcement learning by planning with a learned model. In *Advances in Neural Information Processing Systems*, 2021. arXiv:2104.06294.
- [23] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.
- [24] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.
- [25] Sriram Srinivasan, Erik Talvitie, and Michael Bowling. Improving exploration in UCT using local manifolds. In *AAAI Conference on Artificial Intelligence*, 2015. doi: 10.1609/aaai.v29i1.9660.
- [26] Tom Tseng, Euan McLean, Kellin Pelrine, Tony T. Wang, and Adam Gleave. Can Go AIs be adversarially robust? In *AAAI Conference on Artificial Intelligence*, 2025. arXiv:2406.12843.
- [27] Tony T. Wang, Adam Gleave, Tom Tseng, Kellin Pelrine, Nora Belrose, Joseph Miller, Michael D. Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, and Stuart Russell. Adversarial policies beat superhuman Go AIs. In *International Conference on Machine Learning*, pages 35655–35739, 2023. arXiv:2211.00241.
- [28] David J. Wu. Accelerating self-play learning in Go. *arXiv preprint arXiv:1902.10565*, 2019.
- [29] David J. Wu. KataGo opening books. <https://katagobooks.org/>, 2026. Accessed 2026-10-01.
- [30] Chenjun Xiao, Jincheng Mei, and Martin Müller. Memory-augmented Monte Carlo tree search. In *AAAI Conference on Artificial Intelligence*, 2018. doi: 10.1609/aaai.v32i1.11531.
- [31] Chunyu Xuan, Yazhe Niu, Yuan Pu, Shuai Hu, Yu Liu, and Jing Yang. ReZero: Boosting MCTS-based algorithms by backward-view and entire-buffer reanalyze. *arXiv preprint arXiv:2404.16364*, 2024.
- [32] Weirui Ye, Pieter Abbeel, and Yang Gao. Spending thinking time wisely: Accelerating MCTS with virtual expansions. In *Advances in Neural Information Processing Systems*, 2022. arXiv:2210.12628.

A Training the stopper

The stopper is a gradient-boosted regression tree ensemble (150 trees of depth 3, learning rate 0.05, row subsampling 0.8, at least 30 samples per leaf) fitted to the square root of regret, with all non-final rungs of the ladder pooled. Its prediction is squared before use. Figure 10 shows the fit and its dependence on data. Held-out error stops improving at about 100 trees.

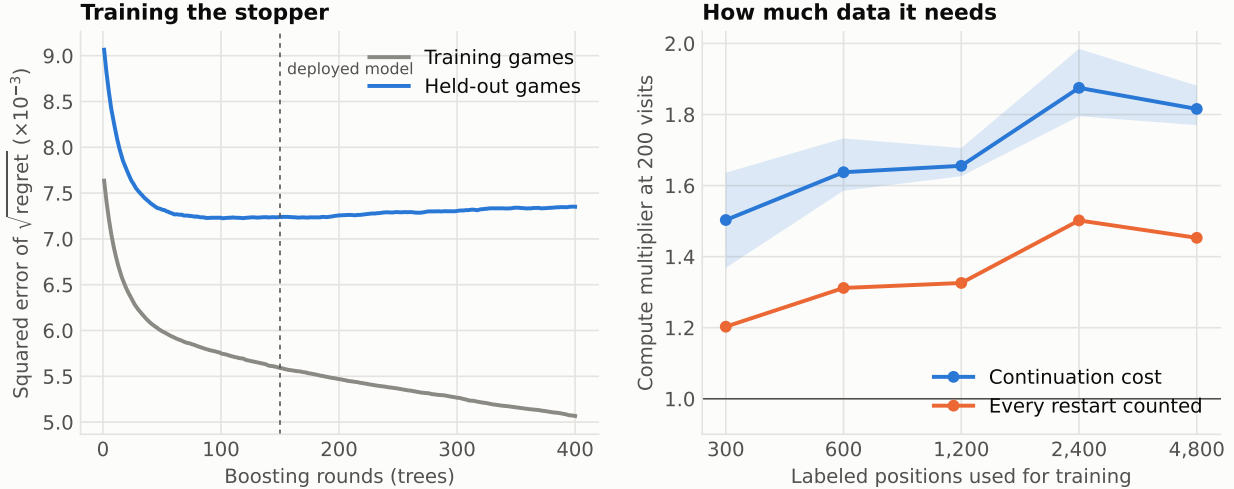


Figure 10: Left: squared error on training and held-out games as trees are added. Right: compute multiplier at a 200-visit budget against the number of labeled positions used for training. The band is the range over three draws of training games.

Figure 11 shows that the score orders held-out positions correctly: realized regret rises from 0.02% of winrate in the lowest decile of predicted stake to 4.2% in the highest. The squared prediction underestimates the level of regret, since it estimates a typical value on the square-root scale. This does not affect play, because only the ordering and a controller-tuned threshold are used. The model relies mostly on one feature, the product of how undecided the game is and how far the best rival’s winrate exceeds the chosen move’s lower confidence bound.

Table 9 lists the ablations. Each entry is the compute multiplier by continuation cost, with the multiplier under restart accounting in parentheses, averaged over three assignments of games to folds.

B Reproducing the results

The repository contains the ladder dataset, the trained models, every game record, and the scripts that produce each table and figure from them. The dataset took 1.5 GPU-hours to label on one RTX 6000 Ada, and a 1,000-game match at 200 visits takes about the same. Training the stopper takes minutes on a laptop CPU. The test suite runs the full pipeline against a small stand-in engine and needs no GPU.

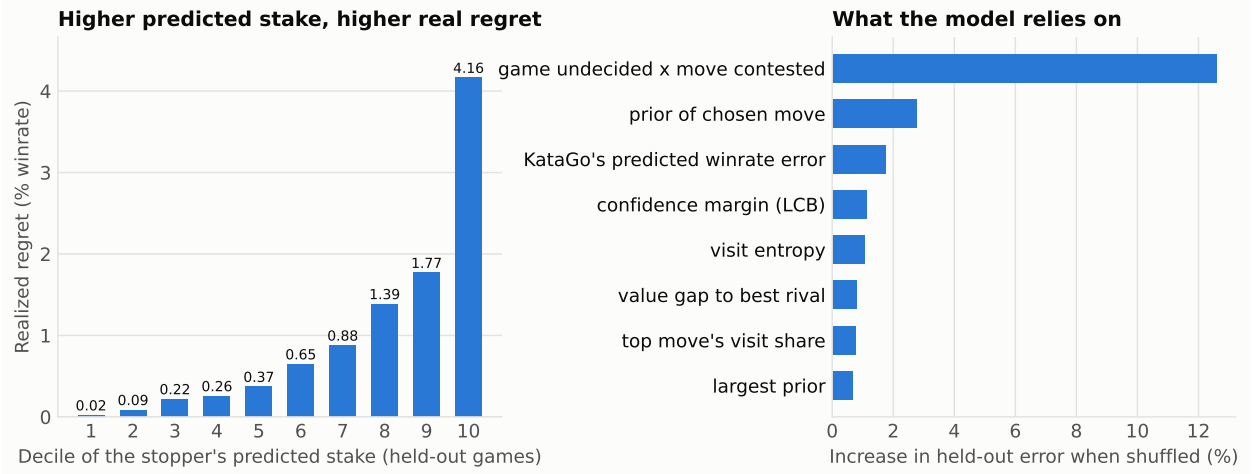


Figure 11: Left: realized regret by decile of the stopper’s score, on held-out games. Right: increase in held-out error when one feature is shuffled.

Variant	100	200	400	800
Full model	1.81 (1.56)	1.82 (1.46)	1.77 (1.37)	1.64 (1.25)
Regret target (no square root)	1.69 (1.45)	1.74 (1.39)	1.64 (1.27)	1.49 (1.13)
Flat threshold (no rate rule)	1.34 (1.16)	1.41 (1.13)	1.22 (0.94)	1.31 (1.00)
No trajectory features	1.80 (1.55)	1.82 (1.46)	1.75 (1.35)	1.63 (1.24)
No KataGo raw error features	1.76 (1.51)	1.87 (1.50)	1.70 (1.31)	1.62 (1.23)
Seven core features	1.70 (1.46)	1.73 (1.39)	1.65 (1.27)	1.62 (1.23)
LCB margin only	1.44 (1.24)	1.48 (1.18)	1.36 (1.05)	1.07 (0.81)
80 trees of depth 2	1.76 (1.51)	1.79 (1.43)	1.67 (1.29)	1.60 (1.22)
400 trees of depth 3	1.74 (1.50)	1.83 (1.46)	1.61 (1.25)	1.58 (1.21)
Ladder 50, 200, 800	1.86 (1.60)	1.84 (1.47)	1.68 (1.30)	1.00 (0.76)
Ladder 100, 400, 1600	–	1.58 (1.36)	1.55 (1.24)	1.50 (1.16)
Ladder 50, 200, 400, 800, 1600, 3200	2.05 (1.62)	1.98 (1.30)	1.97 (1.15)	1.85 (1.01)
Ladder 50, 100, 200, 400, 800, 1600, 3200	2.04 (1.37)	2.14 (1.22)	1.94 (1.04)	1.86 (0.96)

Table 9: Stopper ablations at mean budgets of 100, 200, 400, and 800 visits.