WHERE TO THINK, WHAT TO EXPLORE: HIERARCHICAL AGENTIC REASONING AND EXPLORATION FOR CODING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Coding agents work by interacting with a repository over many turns, so the compute they spend at inference time is as much a design choice as the model itself. That compute can go into reasoning inside a decision, more turns, more candidate actions, or parallel subagents, and these choices have not been compared under one accounting. We measure the four dimensions (reasoning depth D, interaction horizon I, action width W, subagent fan-out S) and their interactions under one coding agent, model, benchmark, serving stack, and cost accounting. Scaled alone they saturate for different reasons: depth has a finite operating point, horizon reaches diminishing returns, width produces candidate diversity that naive selection fails to exploit, and delegation uptake limits passive subagents. They also interact: reasoning changes both the value of later interaction and the cost of parallel branching, and the depth of the step that reduces a fan-out to one action sets what that fan-out is worth. These measurements motivate HARE, Hierarchical Agentic Reasoning and Exploration, which treats D and I as a serial pair and W and S as a parallel pair. HARE reasons only at the turns that write to the repository or synthesize gathered evidence, and folds each fan-out back into one committed action with a single reduction step. On SWE-bench-Pro tasks with a Qwen3.8-27B model, HARE matches dense reasoning at 15% less wall-clock and raises the best measured resolve rate from 56.7 to 60.0%. Whole-trajectory sampling and sequential refinement stay below the frontier HARE reaches, at higher cost.

1 INTRODUCTION

Large language model agents solve tasks through sequential interaction with an environment: software engineering Jimenez et al. (2024); Yang et al. (2024), web navigation Zhou et al. (2024), computer use Xie et al. (2024), and broader decision-making Liu et al. (2024). They repeatedly observe, reason, act, and incorporate feedback over multi-step trajectories Yao et al. (2023b); Jimenez et al. (2024); Zhou et al. (2024), so their performance depends on where the inference-time computation goes as much as on the model. Additional test-time compute helps through longer or more deliberate reasoning Yao et al. (2023a); Snell et al. (2025), repeated sampling of candidates Wang et al. (2023), search over intermediate candidates Yao et al. (2023a), and collaboration across agents Wu et al. (2024).

An interactive agent can spend that compute along four dimensions. Reasoning depth D is how much computation a decision spends, interaction horizon I how long the agent keeps interacting, action width W how many candidate actions it considers from a state, and subagent fan-out S how many concurrent contexts it explores. Recent agentic scaling methods combine sequential refinement with sampling and selection among whole trajectories Ehrlich et al. (2025); Li et al. (2026); Kim et al. (2026); Han et al. (2026). They allocate compute at the level of complete trajectories rather than within decisions, and none characterizes the interactions among all four of (D, I, W, S). We ask: *given additional test-time compute, how should an agent allocate it across (D, I, W, S) to improve accuracy and efficiency?*

We study this question under one coding agent, model, benchmark, serving stack, and cost accounting, varying each dimension alone and then in combination. The dimensions scale differently. Depth has

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

a finite operating point, horizon reaches diminishing returns, width produces candidate diversity that naive selection fails to exploit, and delegation uptake limits passive subagents. They also interact: reasoning changes both the usefulness of later interaction and the cost of parallel branching, and the depth of the step that reduces a fan-out to one action sets what that fan-out is worth. These interactions expose a hierarchy. Depth and horizon are *serial* dimensions and width and fan-out *parallel* ones; within the serial pair, horizon determines *where* computation can be spent and depth *how much*, and within the parallel pair, subagents determine *what contexts to explore* and width *what actions to consider*.

We propose **HARE**, *Hierarchical Agentic Reasoning and Exploration*, allocates the two pairs hierarchically rather than uniformly. It concentrates reasoning on the sparse subset of turns that write to the repository and on the step that synthesizes gathered evidence. Parallel expansion fires only at selected turns, and one reduction step, whose depth the experiments settle, folds each fan-out back into a single committed action.

Our main contributions are:

- We characterize the four dimensions of agentic test-time scaling under one agent, model, benchmark, serving stack, and cost accounting, and show that they saturate for different reasons and interact.
- We identify the serial–parallel structure of agentic compute, a serial pair (D, I) and a parallel pair (W, S), and measure the three interactions that decide how to allocate across them.
- We introduce **HARE**, which combines sparse serial reasoning, parallel exploration at selected turns, and a single reduction step per fan-out, under one placement rule: at most one reasoning step per turn, and never in the judge that picks among action proposals.
- We evaluate HARE on SWE-bench-Pro tasks with a Qwen3.8-27B model. It matches dense reasoning at 15% less wall-clock, ties that accuracy with no reasoning at all, and raises the best measured resolve rate from 56.7 to 60.0%, while whole-trajectory sampling and sequential refinement stay below it at higher cost.

2 Related work

Test-time scaling for language models. Additional inference compute can improve language-model performance through longer or adaptively allocated reasoning, repeated sampling and self-consistency, verifier-guided selection, and search over intermediate candidates (Snell et al., 2025; Muennighoff et al., 2025; Wang et al., 2023; Brown et al., 2024; Lightman et al., 2024; Yao et al., 2023a). These methods motivate reasoning depth and width, but interactive agents introduce a persistent environment state and an additional sequential interaction horizon.

Agentic test-time scaling. Recent work studies test-time scaling for agents through parallel rollouts, sequential revision or reuse, trajectory selection, and budget-aware combinations of sequential and parallel exploration (Ehrlich et al., 2025; Zhu et al., 2025; Li et al., 2026; Kim et al., 2026; Han et al., 2026). These methods primarily allocate additional compute at the level of complete trajectories or trajectory summaries. HARE studies a finer-grained allocation problem inside the trajectory: how reasoning depth D, interaction horizon I, action width W, and subagent fan-out S scale and interact at individual decision points.

Adaptive reasoning and multi-agent systems. Adaptive reasoning methods route between thinking and non-thinking modes or learn when to expose explicit reasoning (Zhang et al., 2025; Liang et al., 2025; Jung et al., 2026; Mia & Amini, 2026). Multi-agent frameworks such as AutoGen, MetaGPT, ChatDev, CAMEL, and OpenHands organize cooperating agents with fixed or dynamic roles (Wu et al., 2024; Hong et al., 2024; Qian et al., 2024; Li et al., 2023; Wang et al., 2024). HARE places these mechanisms in a common scaling space: depth and horizon are serial dimensions, while width and subagents are parallel dimensions with distinct roles in action and context exploration.

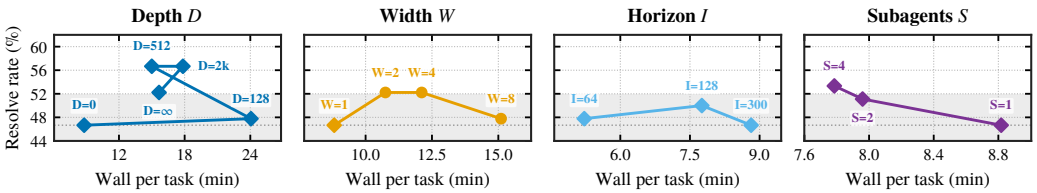


Figure 1: Scaling one dimension at a time from the non-reasoning baseline. Only depth improves accuracy beyond the noise floor; width, horizon, and passive subagent access do not.

3 Motivation: How Agentic Test-Time Compute Scales

3.1 Scaling Problem

A turn is one LLM call followed by execution of the emitted tool calls. With agent state x_t, committed action a_t, and environment transition E,

$$x_{t+1} = E(x_t, a_t), \qquad t = 1, \ldots, I. \tag{1}$$

A trajectory ends with a final message or at the horizon I, after which the official SWE-bench-Pro harness grades the repository diff. The serial pair (D, I) spends computation along the trajectory: I sets the decision points and D_t the reasoning at decision point t. The parallel pair (W, S) spends it on fan-out: W creates alternative actions from one state, S concurrent contexts that investigate different aspects of the task. An allocation policy is

$$\pi(x_t) \to (\text{continue}_t, D_t, W_t, S_t), \tag{2}$$

and we maximize $\text{ResolveRate}(\pi)$ subject to $T_{\text{wall}}(\pi) \leq B$. Sweeping B traces the resolve-rate–wall-clock frontier we report, and we report modelled GPU-busy time separately because concurrency hides hardware work behind wall-clock.

3.2 Experimental Setup

We study Hermes (Nous Research, 2025) with its default prompts and tools and Qwen3.8-27B in FP8 served by vLLM 0.19 (Kwon et al., 2023) on one H200. Qwen exposes reasoning as a per-request switch (Yang et al., 2025); we cap hidden reasoning at D tokens with native budget control (Muennighoff et al., 2025), $D = 0$ denoting no reasoning and $D = \infty$ an unbounded chain. Each task runs in an isolated, network-disabled container on four CPU cores under a two-hour cap. The benchmark is a repository-balanced 90-task subset of SWE-bench-Pro (Deng et al., 2025).

The first campaign varies $D \in \{0, 128, 512, 2048, \infty\}$, $W \in \{1, 2, 4, 8\}$, $I \in \{64, 128, 300\}$, and passive subagent access $S \in \{1, 2, 4\}$, holding agent, model, benchmark, serving stack, and accounting fixed. The second campaign runs the policies of Section 5 and changes only the GPUs, a second H200 cluster (Appendix K). Paired comparisons use exact McNemar tests (Appendix G). The second cluster holds five runs of the non-reasoning baseline: a matched re-run, three seeds that form the $K = 4$ pool of Section 5.1, and one seed matched to its replicates. The matched re-run resolves 52.2% at 10.4 minutes against 46.7% at 8.8 minutes on the first cluster. We compare wall-clock across clusters unadjusted and pair every second-campaign setting against both baselines (Table 6).

We estimate GPU-busy time as $T_{\text{gpu}} = \sum_r [p(\text{ctx}_r) + c_d(n_r, \text{ctx}_r) \cdot \text{decodeSteps}_r]$ over requests r, with p the calibrated prefill cost and c_d the decode-step cost of an n-way request; a microbenchmark replaying real prompts from 4k to 128k tokens at $n \in \{1, 2, 4, 8\}$ calibrates both (Appendix H).

3.3 Scaling Each Dimension Independently

Figure 1 scales one dimension at a time from the non-reasoning baseline ($D = 0$, $W = 1$, $I = 300$: 46.7% at 8.8 minutes per task), and Table 1 gives wall-clock and modelled GPU-busy time.

Depth has a finite operating point. Only depth gains beyond the noise floor, and more depth is not monotonically better. A 512-token budget resolves 56.7% at 15.0 minutes, against 46.7% at

Table 1: Resolve rate (%) of the measured settings, with mean wall-clock and modelled GPU-busy time in grey (minutes). Full token and call accounting is in Table 3.

Knob	Value	No reasoning (D=0)	Reasoning (D=∞)
		Rate (%) (wall / GPU-busy, min)	
Baseline	W=1, I=300	46.7 (8.8 / 5.5)	52.2 (15.7 / 14.2)
Depth	D=128	–	47.8 (24.1 / 10.5)
	D=512	–	56.7 (15.0 / 12.9)
	D=2048	–	56.7 (17.8 / 15.8)
Width	W=2	52.2 (10.7 / 6.6)	52.2 (26.7 / 25.0)
	W=4	52.2 (12.1 / 8.1)	54.4 (35.0 / 31.3)
	W=8	47.8 (15.1 / 10.2)	53.3 (42.6 / 39.7)
Horizon	I=64	47.8 (5.2 / 3.2)	50.0 (12.3 / 11.9)
	I=128	50.0 (7.8 / 4.2)	53.3 (21.5 / 13.8)
Subagents	S=2	51.1 (8.0 / 4.8)	–
	S=4	53.3 (7.8 / 4.9)	–

8.8 minutes without reasoning and 52.2% at 15.7 minutes with an unbounded chain; paired against no reasoning it wins 10 tasks and loses 1 ($p = 0.01$). A 2,048-token budget resolves 56.7% at 17.8 minutes. The 128-token budget is inefficient: 47.8% after 24.1 minutes and 102 LLM calls per task, consistent with chains truncated before completion and an agent compensating with more interactions.

Width creates diversity that pick-1 does not exploit. At $D = 0$, $W = 2$, 4, and 8 resolve 52.2, 52.2, and 47.8% at 10.7, 12.1, and 15.1 minutes, while the width hook produces 1.9, 3.6, and 5.9 distinct actions per turn. Pick-1 (Appendix B) commits the most frequent action and breaks ties on branch 0, so it selects branch 0 on 98% of steps at $W = 2$, 78% at $W = 4$, and 63% at $W = 8$. These settings measure action generation plus pick-1, not oracle use of width.

Horizon reaches diminishing returns. At $D = 0$ a 128-turn cap resolves 50.0% at 7.8 minutes and a 64-turn cap 47.8% at 5.2 minutes, against 46.7% at 8.8 minutes for 300 turns. Beyond roughly 128 turns, long failing trajectories dominate the extra sequential work: they keep reading, editing, and retesting without converging.

Passive subagent access is uptake-limited. With passive access to two child agents at $D = 0$ the parent resolves 51.1% at 8.0 minutes, with four 53.3% at 7.8 minutes. Only 14 of 90 runs delegate at $S = 2$ (20 children) and 11 of 90 at $S = 4$ (17 children); an $S = 2$ child makes 6 LLM calls and 7 commands before returning a summary. The setting measures permission to delegate, not forced fan-out.

Observation 1: single-axis scaling saturates for different reasons. Depth has a finite operating point, horizon reaches diminishing returns, width produces candidate diversity that naive selection fails to exploit, and delegation uptake limits passive subagents. The four dimensions are not interchangeable ways of buying compute.

3.4 Interactions between the dimensions

Depth and horizon are coupled. At $D = \infty$ a 64-turn cap resolves 50.0%, 2.2 points below the 300-turn setting, while at $D = 0$ the same truncation costs nothing (47.8 against 46.7%); the 128-turn cap stays within noise in both modes. Both modes issue a similar number of commands before the first edit (28 with reasoning, 32 without), but afterwards a non-reasoning run averages 15 edits over 86 turns and a reasoning run 7 edits over 55 turns. Failing runs stretch to 99 turns and 18 edits without reasoning and stop near 51 turns and 6 edits with it. Depth is therefore a vector $(D_1, \ldots, D_I)$ over the trajectory rather than a scalar.

Reasoning inside a branch is expensive. Reasoning inside every width branch is the worst composition measured: at $D = \infty$, $W = 2$, 4, and 8 resolve 52.2, 54.4, and 53.3% at 26.7, 35.0, and 42.6 minutes, against 52.2% at 15.7 minutes for $W = 1$. Accuracy stays within noise while $W = 8$

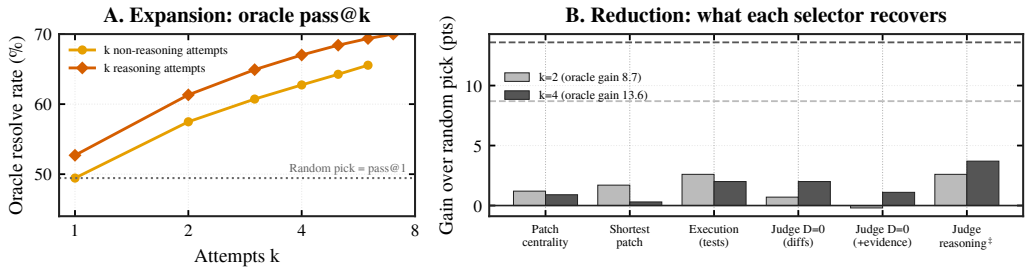


Figure 2: Expansion and reduction. A: the oracle over k pooled attempts of one mode rises with k (pass@k, unbiased estimator). B: every selector's gain over the random pick on its own subsets at $k = 2$ and $k = 4$; the dashed lines are the pooled oracle gain. A judge that reasons recovers a third of its own oracle gap at $k = 4$ (‡, measured on its own subsets); the non-reasoning judge and execution-based selection recover about a sixth.

raises wall-clock by 2.7× and output tokens by 8.4×, against 1.7× at $D = 0$, because every proposal carries its own serial chain (Appendix F, Figures 11 and 12). Reasoning therefore belongs after a fan-out, not inside it.

3.5 What a fan-out is worth

Parallel sampling collects solutions that pick-1 discards. Pooling the six first-campaign non-reasoning settings (W=1, 2, 4, 8 pick-1 and I=64, 128; 25 random k-subsets per task) raises oracle coverage to 57.1% at pass@2 and 62.9% at pass@4, while a random pick stays near 49%. Mechanical selectors, which rank candidates by edit count, patch length, or patch centrality, stay within a point of the random pick at $k = 4$; execution-based selection over repository tests and a non-reasoning judge over candidate diffs each recover 2.0 points, a sixth of their oracle gap, and a reasoning judge 3.7, a third of its own (Table 2, Figure 2, Appendix C; every selector is scored against the random pick on its own subsets). The coverage is real and hard to collect, and its value depends on the step that reduces the candidates to one action.

Observation 2: the measurements prescribe a shape to test. Spend depth at selected turns, expand in parallel with cheap candidates instead of reasoning branches, and put the reasoning, if any, in the single step that reduces the fan-out. Section 4 builds that policy and Section 5 tests it.

4 HARE

Observation 2 of Section 3 prescribes a policy: sparse depth, cheap parallel candidates, and the reasoning, if any, in the step that reduces them. Motivated by the observations, we propose HARE (Figure 3), Hierarchical Agentic Reasoning and Exploration. It reserves a 512-token budget for the turns that write to the repository, expands in parallel only at selected turns, and ends every fan-out in one *reduction step* that returns a single committed action. Each decision follows one measurement. A horizon cut costs accuracy only when the turns reason, and reasoning shortens the trajectory (Section 3.4), so depth belongs at a sparse subset of turns; the write turns are a cheap trigger for that subset, and Section 5.2 shows a random-sparse control matching it. Reasoning inside every width branch costs wall-clock and tokens for no accuracy (Section 3.4), so the parallel pair expands with cheap candidates and reduces once. A fan-out is worth what its reduction step recovers (Section 3.5), and a reasoning parent almost never delegates on its own (Appendix I), so HARE forces the context fan-out and reduces it in one step. We do not measure width or subagent fan-out crossed with the horizon.

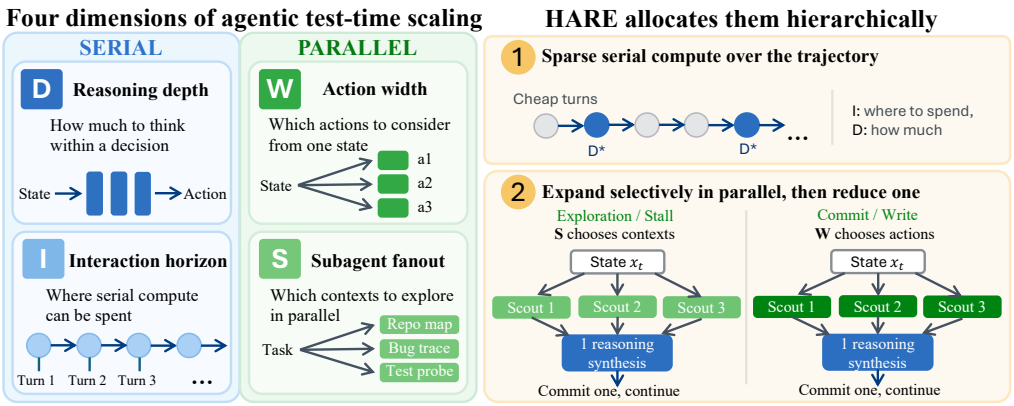


Figure 3: Left: the four dimensions of agentic test-time scaling. Depth D controls how much computation a decision spends; horizon I controls where serial computation can be spent along the trajectory; width W controls which actions the agent considers from a state; fan-out S controls which contexts it explores in parallel. Right: HARE allocates them hierarchically: sparse serial reasoning over the trajectory and, at selected turns, parallel expansion through subagents or action proposals, followed by one reduction step before the next action, a reasoning synthesis over subagent evidence or a cheap judge over action proposals.

4.1 Serial allocation: sparse depth over the horizon

A dense policy spends the same budget on every turn, $D_t = D$; HARE reserves the budget for selected turns,

$$D_t = \begin{cases} D^*, & t \in \mathcal{C}, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The serial instantiation is training-free. The hook first drafts every turn with reasoning disabled; if the draft would modify the repository, it discards the draft and regenerates the same request with $D^* = 512$, and read-only drafts commit directly. We call this write-triggered instantiation *HARE-Edit*. Two controls separate the value of sparse serial reasoning from the trigger: *HARE-Rand* spends the same budget on a random 16% of turns against HARE-Edit's 13%, and *HARE-Ends* also reasons on the first and final turns. One level outward, I controls how many decision points exist and $(D_1, \ldots, D_I)$ how the policy distributes reasoning over them, so the serial problem is to decide whether to continue and, if so, whether the current decision warrants deeper reasoning.

4.2 Parallel allocation and reduction

Each fan-out ends in one reduction step of depth D_r, the *reducer*: a *judge* that picks one of W candidate actions, or a *synthesis* that folds S subagent reports into the next action. Selective width generates W inexpensive non-reasoning proposals from one state; scouting spawns S read-only subagents, the *scouts*, with distinct investigation objectives and collects their compact summaries:

$$\{a_t^{(1)}, \ldots, a_t^{(W)}\}_{D=0} \to \text{Judge}_{D_r} \to a_t, \qquad \{e_t^{(1)}, \ldots, e_t^{(S)}\}_{D=0} \to \text{Synthesize}_{D_r} \to a_t. \tag{4}$$

Read-only scouts avoid conflicting repository mutations, and Equation 4 fixes where the reasoning may sit. One rule governs the two reducers, and Section 5.3 tests it: at most one reasoning step per turn, and never in the judge that picks among action proposals. Section 5.3 measures $D_r \in \{0, 512\}$ for each reducer and confirms the assignment of Figure 3. *HARE-Width* is selective $W = 4$ at write turns with a non-reasoning judge ($D_r = 0$); *HARE-Scout* is $S = 4$ scouts with a 512-token synthesis per round, in three variants (one round after the first tool turn, one round at a random turn, repeated rounds). We report the other depth of each reducer as its *reasoning-judge variant* and *non-reasoning-synthesis variant*. The resulting policy spends nothing extra on routine turns, $(D, W, S) = (0, 1, 1)$, fans out contexts at scout turns and actions at write turns, and reduces once after each fan-out.

Triggers and controls. Selective width reuses HARE-Edit's write trigger: the hook re-samples a draft that would modify the repository as $W = 4$ non-reasoning proposals from the same state and the judge picks one. *Uniform width* applies the same expansion and judge on every turn and *random width* on a seeded 15% of turns; they separate selectivity from the reducer. Scouting is positional: after the first tool turn the parent blocks while four read-only scouts pursue fixed objectives (the faulty code, the tests and specification, the call and data flow, analogous code), each with 25 turns and a 15-minute cap. The parent's next request synthesizes their summaries. HARE-Scout (repeated) fires a round on every stall of three read-only turns (at most four rounds); HARE-Scout (random turn) fires once at a seeded random turn and tests the position. The scout hook refuses scout writes. The parent keeps its delegation tool, as in the passive settings; its own children are ordinary subagents, counted in cost and outside the read-only gate (Appendix I).

Compositions and baselines. Two compositions put the components on one trajectory: HARE-Edit with HARE-Scout, and the *judge stack*, which stacks the 512-token judge of selective width on one scout round, at $I = 300$ and $I = 128$. The judge stack tests whether a misplaced reducer propagates through a policy. The placement rule prescribes two further compositions (Section 5.1): HARE-Width with HARE-Scout (repeated), reasoning only at the syntheses, and HARE-Edit with HARE-Scout (repeated). Two controlled prior-work policies use the same agent, model, and tasks. Whole-trajectory sampling runs $K = 4$ independent non-reasoning trajectories from distinct seeds and selects one offline with a random pick, the candidates' own tests, or an offline trajectory judge without reasoning or with a 2,048-token budget (Appendix D). Sequential refinement, our controlled analogue of trajectory-reuse methods (Kim et al., 2026; Han et al., 2026), runs a second non-reasoning trajectory whose task statement carries a model-written summary (at most 250 words: final message, patch, last test run) of the task's first-campaign non-reasoning baseline run. The summarizer never sees the verdict.

5 Experiments

Section 5.1 places every policy on one accuracy–latency plane, and Sections 5.2 and 5.3 ablate the serial and the parallel half.

5.1 The accuracy–latency frontier

Figure 4 draws resolve rate against wall-clock for the HARE instantiations, the two compositions, and the two prior-work policies; Figure 13 in the appendix repeats it against GPU-minutes.

HARE spans the frontier. HARE-Width resolves 58.9% at 11.6 minutes and HARE-Scout (repeated) 60.0% at 14.9 minutes, the highest resolve rate measured. HARE-Edit reaches the same 58.9% as HARE-Width at 12.7 minutes, and dense $D = 512$ 56.7% at 15.0 minutes. The frontier runs through passive $S = 4$, HARE-Scout (random turn), HARE-Width, and HARE-Scout (repeated), with HARE-Edit just behind it. Every point on it above the baseline spends at most one reasoning step per turn: at the scout synthesis, at the write turn, or nowhere.

A misplaced reducer survives composition. The judge stack, which gives the width fan-out a 512-token judge and keeps the scout synthesis, resolves 51.1% at 13.3 minutes, 8 points below HARE-Edit ($p = 0.09$) and 6 below dense reasoning; at $I = 128$ it resolves 52.2%. HARE-Edit with HARE-Scout, which keeps both reducers as the rule prescribes, resolves 57.8% at 12.5 minutes, equal to HARE-Edit alone. The two compositions that pair an instantiation with repeated scouts behave the same way. HARE-Width with HARE-Scout (repeated) resolves 61.9% at 16.4 minutes against 63.1% for either component alone on those tasks, and HARE-Edit with HARE-Scout (repeated) 57.6% at 18.9 minutes against 61.2% for HARE-Edit. Neither difference separates ($p \geq 0.5$), and the tasks they have yet to grade are ones their components fail, and Figure 4 draws them hollow and outside its frontier line. Composition neither repairs a misplaced reducer nor adds to a correct one, and a second correct reducer buys wall-clock rather than accuracy.

Whole-trajectory sampling collects solutions that no selector recovers. Four non-reasoning seeds resolve between 46.7 and 52.2% and pairwise flip 16–23% of tasks, the single-seed noise floor of every comparison here. On the SWE-Bench-Pro tasks graded under every seed their oracle reaches 60.2% at $K = 2$ and 67.4% at $K = 4$, against 51.6% for a random pick over the same subsets. No training-free

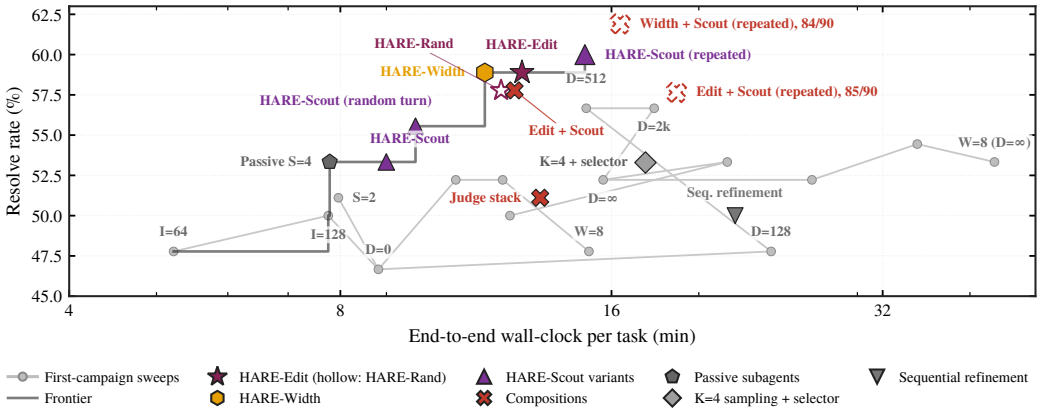


Figure 4: Accuracy–latency frontier (the GPU-minute view is Figure 13). Grey: the first-campaign sweeps of Figure 1, labelled at their ends; coloured: the HARE instantiations, the compositions, and the prior-work policies. The step line is the frontier over every setting graded on SWE-Bench-Pro tasks; the variants and controls of Figure 5 lie below it and are not drawn. The two hollow crosses are the repeated-scout compositions at their graded tasks, which the frontier line excludes because the tasks they have yet to grade are ones their components fail. HARE-Edit (58.9%) and HARE-Rand (57.8%) sit just behind it, and the diamond is $K = 4$ sampling with execution-based selection at the per-task maximum of the four walls.

selector collects much of it: the shortest non-empty patch resolves 57.3%, execution-based selection 53.3%, and the three offline judges fall below the random pick (Appendix C). Sampling also costs 17.5 minutes and 20.0 GPU-minutes per task, more wall-clock and more hardware work than any HARE instantiation. Sequential refinement resolves 50.0% within noise of its own baseline, at 21.9 minutes the slowest point measured.

A second seed leaves one to three points of every gain. Re-run under a new seed on the second cluster, the baseline, HARE-Edit, HARE-Width, and HARE-Scout (repeated) resolve 51.1, 54.4, 52.2, and 52.8%. Paired against their seed-mate baseline, the three policies gain 3.3, 1.1, and 1.1 points, against 6.7, 6.7, and 7.8 for the first seed of each. No second-seed comparison separates at $p < 0.05$, so Figure 4 is a single-seed ordering.

5.2 Serial ablation: sparse against dense depth

Sparse depth buys the accuracy of dense depth for less time. HARE-Edit resolves 58.9% at 12.7 minutes and 9.4 GPU-minutes while reasoning on 13% of its turns. It gains 12.2 points over the non-reasoning baseline ($p = 0.007$) for 3.9 more minutes per task, and matches dense $D = 512$ at 15% less wall-clock. The write trigger is not what earns that: HARE-Rand reaches 57.8% at 12.1 minutes and HARE-Ends 53.3% (Appendix J, Table 8), so sparse full-depth reasoning carries the gain rather than the choice of turns.

Sparse depth needs the long horizon. At $I = 128$ HARE-Edit resolves 47.8%, 11 points below $I = 300$ ($p = 0.013$) and within noise of no reasoning at that horizon; at $I = 64$ it resolves 43.3% ($p = 0.001$). The cut removes the turns the trigger fires on, since the first edit falls at turn 36 on average and the write turns follow it. The 512-token budget is the operating point under sparsity too: 128 tokens at the selected turns resolve 52.2%, and an unbounded budget 57.8% at 20.5 minutes, the same accuracy at 1.6× the wall-clock. Figure 5 collects the three composition questions.

5.3 Parallel ablation: the depth of the reducer

The depth of the reducer decides what an action fan-out is worth. HARE-Width, whose judge does not reason, resolves 58.9% at 11.6 minutes and 7.4 GPU-minutes, 12.2 points above the baseline ($p = 0.019$). The same judge with a 512-token budget resolves 47.8%, 11 points lower ($p = 0.006$) and 9 below dense reasoning. That judge does not abstain: both judges see 3.9 distinct proposals

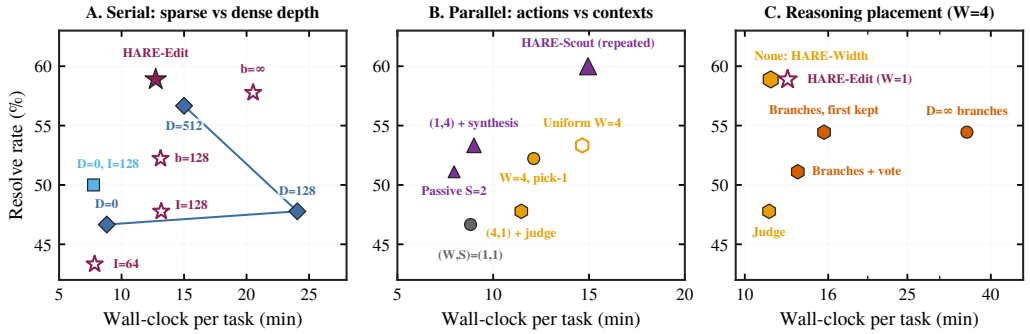


Figure 5: Composition ablation on the resolve-rate–wall-clock plane; hollow markers are variants of the named setting. A: dense against sparse depth over the horizon (stars: HARE-Edit at budget b or horizon I). B: matched fan-out, actions (W) against contexts (S), each with one 512-token reducer. C: reasoning placement under $W = 4$: none (HARE-Width), in the judge, or in every branch.

per expanded turn and rarely keep branch 0, so the reasoning judge spends 68 seconds per task to commit worse proposals. Every other reasoning placement on the same fan-out also loses: a majority vote over four 512-token proposals at 51.1%, committing the first of them at 54.4%, uniform width at 53.3%, and random width at 50.0%.

Context fan-out runs the other way. HARE-Scout (repeated) resolves 60.0% ($p = 0.004$) at 14.9 minutes and 13.6 GPU-minutes, with 3 rounds, 15 scouts, and 356 seconds of parent waiting per task. One round resolves 53.3% and one round at a random turn 55.6%, so the number of rounds matters ($p = 0.07$) and its position does not. Removing the reasoning synthesis drops the same scouts to 48.9%, so their evidence alone does not carry the gain (Appendix I). Matched at four branches and one 512-token reducer, contexts therefore lead actions, 53.3 against 47.8%, and the order flips under a non-reasoning reducer, 48.9 against 58.9%. Against the same-cluster baseline re-run, HARE-Width gains 6.7 points and HARE-Scout (repeated) 7.8; the tests above use the first-campaign baseline, the lowest of the six non-reasoning runs, and Table 6 carries both references.

Observation 3: the reducer decides the outcome of parallel work. One reasoning step after a context fan-out moves accuracy from 48.9 to 53.3% ($p = 0.29$), and to 60.0% with repeated rounds. One reasoning step after an action fan-out costs 11 points against a non-reasoning judge. At most one reasoning step per turn, and never in the judge that picks among action proposals.

6 Conclusion

Agentic test-time compute has four dimensions that are not interchangeable: scaled alone, depth has a finite operating point, horizon reaches diminishing returns, width produces candidate diversity that naive selection fails to exploit, and delegation uptake limits passive subagents. They interact in cost, since depth and horizon extend the critical path while width and subagents trade hardware work for concurrency, and in decision quality, since the depth of a reduction step sets what a fan-out is worth. The interactions expose a hierarchy between a serial pair and a parallel pair: horizon decides where computation goes and depth how much; subagents decide what contexts to explore and width what actions to consider. HARE turns that hierarchy into a runtime policy under one placement rule: at most one reasoning step per turn, and never in the judge that picks among action proposals. Its instantiations reach the measured frontier: HARE-Edit matches dense reasoning at 15% less wall-clock, HARE-Width ties it with no reasoning at all, and HARE-Scout raises the best measured resolve rate to 60.0%. Whole-trajectory sampling and sequential refinement stay below them at higher cost. Learning the policy, with the trigger and the reducer depth as its outputs, is the next step.

AI use statement

We used a generative AI coding assistant to draft the experiment orchestration, analysis and figure scripts, and to draft and edit the prose of this paper. We did not use generative AI to design the

experiments, to produce or alter any experimental result, or to generate citations without verification. Every number in the paper is produced by the released scripts from the released per-trajectory artefacts. We take responsibility for the final content of this work, including text, claims and artefacts produced with the aid of generative AI.

Ethics statement

The benchmark tasks are open-source repositories with permissive licences; every agent container ran without network access, so no external system was contacted by an agent. The study involves no human subjects. The campaign consumed about 1 400 H200 GPU-hours; the measurements are intended to reduce that cost for others.

Reproducibility statement

Section 3.2 and Appendix A give the agent, model, serving flags, task selection rule and per-task cap; Appendix B gives the width mechanism, the pick-1 rule and the merge prompts; Appendix C gives the selector definitions; Appendix H gives the cost model, its fits and the microbenchmark; Appendix I gives the subagent mechanism; Appendix G gives every paired test. The supplementary material contains the 27-setting specification, the 146 task identifiers with their baseline outcomes, the run ledger, the per-trajectory artefacts (transcripts, per-request token and latency records, tool-call logs) and the scripts that produce every table and figure.

References

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Xiang Deng, Jeff Yang, Chongyi Dushi, Tejas Goel, et al. SWE-Bench Pro: Long-horizon software engineering tasks for coding agents. *arXiv preprint arXiv:2509.16941*, 2025.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. CodeMonkeys: Scaling test-time compute for software engineering. *arXiv preprint arXiv:2501.14723*, 2025.

Rujun Han, Samira Daruki, Tomas Pfister, Chen-Yu Lee, Jiefeng Chen, Ke Jiang, Yanfei Chen, I-Hung Hsu, Jin Miao, Ethan Liang, Tengxiao Liu, Zifeng Wang, and Jun Yan. Cost-effective agent test-time scaling. Google Research, 2026. URL `https://research.google/pubs/cost-effective-agent-test-time-scaling/`.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL `https://arxiv.org/abs/2310.06770`.

Dongwon Jung, Peng Shi, Yi Zhang, Junshan Zhang, and Muhao Chen. Adaptive latent agentic reasoning. *arXiv preprint arXiv:2606.02871*, 2026.

Joongwon Kim, Wannan Yang, Kelvin Niu, Hongming Zhang, Yun Zhu, Eryk Helenowski, Ruan Silva, Zhengxing Chen, Srinivasan Iyer, Manzil Zaheer, Daniel Fried, Hannaneh Hajishirzi, Sanjeev Arora, Gabriel Synnaeve, Ruslan Salakhutdinov, and Anirudh Goyal. Scaling test-time compute for agentic coding. *arXiv preprint arXiv:2604.16529*, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *SOSP*, 2023.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Xiaochuan Li, Ryan Ming, Pranav Setlur, Abhijay Paladugu, Andy Tang, Hao Kang, Shuai Shao, Rong Jin, and Chenyan Xiong. Benchmark test-time scaling of general LLM agents. *arXiv preprint arXiv:2602.18998*, 2026.

Guosheng Liang, Jiaqi Zhong, Yong Zhang, et al. ThinkSwitcher: When to think hard, when to think fast. *arXiv preprint arXiv:2505.14183*, 2025.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *ICLR*, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

Md Jueal Mia and M. Hadi Amini. Don't overthink, don't underthink: Toward adaptive reasoning in agentic AI. *arXiv preprint arXiv:2608.26442*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

Nous Research. Hermes agent. https://github.com/NousResearch/hermes-agent, 2025. version used in this work.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.03314.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. OpenHands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2203.11171.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-1650. URL https://arxiv.org/abs/2404.07972.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, 2024. doi: 10.52202/079017-1601. URL https://arxiv.org/abs/2405.15793.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023a. URL https://arxiv.org/abs/2305.10601.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023b. URL https://arxiv.org/abs/2210.03629.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. *arXiv preprint arXiv:2505.13417*, 2025.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for LLM agents. *arXiv preprint arXiv:2506.12928*, 2025.

APPENDIX

- Sec. A: Setup Details
  - Task Selection
  - Serving and Jobs
  - Grading
- Sec. B: Width Mechanism
- Sec. C: Selectors
  - Execution-based Selection
  - Offline Selection Results
  - Four-seed Pool
- Sec. D: Judge Prompt
- Sec. E: Additional Results
- Sec. F: Interaction and Cost Measurements
- Sec. G: Paired Tests
- Sec. H: Cost Model
  - Per-request Fits
  - Decomposition
  - Microbenchmark
- Sec. I: Subagent Mechanism
- Sec. J: HARE Serial Implementation and Regeneration Statistics
- Sec. K: Second Campaign: Composition and Baseline Settings
  - Task Subset
  - Cluster
  - Settings

A SETUP DETAILS

Task selection. SWE-bench-Pro has 731 public tasks over eleven repositories. We first ran the agent on 250 tasks in both modes; from those we drew 146 (20% of 731) with a per-repository quota as even as availability allows (tutanota 4, NodeBB 12, teleport, ansible, openlibrary and navidrome 15 each, the rest 14), stratified within each repository on the pair of pilot outcomes, preferring tasks whose pilot wall-clock was under two hours in both modes. The subset resolves 56.2% (reasoning) and 48.6% (non-reasoning) in the pilot against 55.6% and 48.8% on the 250.

Serving and jobs. Qwen3.8-27B-FP8 on one H200 under vLLM 0.19 with prefix caching and the native `thinking_token_budget`; one job per setting on an eight-core allocation (four cores for the server, four for the agent process and its container); a per-task cap of two hours; a fixed seed base per setting with the request index added by a proxy, so branches of one request are distinct and reproducible. A one-token probe before every task detected a job whose engine stopped answering; we re-ran its tasks.

Grading. The official SWE-bench-Pro harness on the submitted diff; a task killed at the cap counts as unresolved.

B WIDTH MECHANISM

At every turn the request carries $n = W$; the hook parses each choice into its tool calls and keys it by the canonical command (or FINAL when the draft calls no tool). *Pick-1* commits the majority key; ties go to the majority verb signature, then to a tool call over a final answer, then to a registered tool over an unregistered one, then to the lowest index (branch 0). For measurement, the hook executes the alternatives that a static allowlist proves read-only on a shadow of the container state (own working directory and snapshot) concurrently with the committed command, and the reported wall-clock includes their cost. Three merge policies decide what the agent sees: *pick-1* hides the alternatives; *raw* appends every executed alternative's command and observation to the committed command's output; *compressed* asks the same model, with reasoning off and a 512-token limit, for a summary of the alternatives and appends that. The sampled diversity under pick-1 at $D = 0$ was 1.9, 3.6 and 5.9 distinct actions per turn at $W = 2$, 4 and 8, flat along the trajectory (3.7 / 3.6 / 3.4 by thirds at $W = 4$).

C SELECTORS

Execution-based selection. For every candidate: reset the task image to the base commit, apply the patch, run the test files the patch adds or modifies (own tests), then apply each other candidate's test-side files on top and run those (cross-candidate tests). Runners per repository: `go test` per touched package (flipt, vuls, teleport, navidrome), `pytest` (ansible, openlibrary, qutebrowser under an offscreen display), `mocha` (NodeBB), `jest` (element-web, webclients) and tutanota's whole suite; 900 s per invocation, 30 min per candidate, no network. The selector uses no hidden or gold test. Selection rules: most passes over own and cross tests, most cross passes, most own passes, fewest fails, highest pass rate. Random: one candidate uniformly. Fewest / most edits and fewest turns: from the transcript. Shortest wall: the attempt with the lowest wall-clock. Last test passed: the exit code of the last command matching a test runner. Patch centrality: the candidate with the highest mean similarity to the others, by Jaccard over added and removed lines or over touched files. Shortest / longest non-empty patch: by diff length.

Offline selection results. The six non-reasoning settings of the first campaign give up to six attempts per task with known outcomes, so we score every selector on the 90 tasks without new agent runs: for each task and $k \in \{2, 4\}$ we draw 25 random k-subsets, apply the selector, and grade the selected patch from the ledger (Table 2). The non-reasoning judge does not select: 49.8% at $k = 2$ with execution evidence and 50.7% without, against 50.0% for a random pick on the same 450-call subsets, and 52.9 / 53.8% at $k = 4$ against 51.8% (Table 2 reports every selector's gain against the random pick on its own subsets; its random and oracle rows are the mechanical run's); the two input variants agree on 84% of the pairs and 61% of the quadruples, so the judge is consistent and wrong,

and the diffs-only variant picks the first candidate in 53% of the pairs and 35% of the quadruples, a mild first-position preference. Execution-based selection recovers the same 2.0 points at $k = 4$ (530 of 530 non-empty patches applied, 32 s per candidate, 4.8 CPU-hours; the table shows the best of its five rules, most cross-candidate passes): resolved candidates pass more tests on average (162 against 98), but the tasks carry the difference, since only 35 of the 90 tasks have a unique top scorer and 32 tasks have no runnable test in any candidate. A judge that reasons before answering (2 048-token budget, 32 s per call) recovers 2.6 of the 7.8 points between random and oracle at $k = 2$ and 3.7 of the 11.5 points at $k = 4$, over 270 calls each.

Four-seed pool. Over the $K = 4$ seed pool, execution-based selection over the seeds' own and cross-candidate tests reaches 53.3% at $K = 4$ (random 50.8%, oracle 66.7%; a runnable test exists for 56 of the 90 tasks) for about 100 CPU-seconds per task, and the offline judges reach 52.8% from the diffs, 48.3% with execution evidence and 51.7% with 24 seconds of reasoning per call, against 53.9% for the single random draw they scored and an oracle of 67.4% on the same 89 quadruples. Over the four-seed pool of Section 5.1, the candidates' own test outcomes, edit counts, and patch centrality stay at the random pick at $K = 4$; at $K = 2$ the offline judges gain 3 to 4 points over a random draw (51.7, 51.3, and 52.1% for the diffs, evidence, and reasoning variants against 47.9% on 267 pairs, oracle 59.9%), a third of the pass@2 gap, which the fourth candidate erases.

Table 2: Offline selection over the first campaign's non-reasoning settings: resolve rate (%) of the selected patch and, in parentheses, its gain over the random pick on the same subsets. The random and oracle rows are the 25-subset mechanical run (48.4 / 49.3 and 57.1 / 62.9 at k=2 / 4); execution-based selection draws its own subsets (random 49.6 / 49.9), as do the judges (50.0 / 51.8) and the reasoning judge ‡, which scores 3 subsets per task at 32 s per call against its own random and oracle bounds of 45.9 / 53.7 at $k = 2$ and 50.0 / 61.5 at $k = 4$.

Selector	$k = 2$	$k = 4$
Random (= pass@1)	48.4	49.3
Mechanical selectors (Appendix C)		
Patch centrality (line-set agreement)	49.6 (+1.2)	50.2 (+0.9)
Shortest non-empty patch	50.1 (+1.7)	49.6 (+0.3)
Execution-based		
Repository + cross-candidate tests, most cross passes	52.2 (+2.6)	51.9 (+2.0)
Non-reasoning judge		
Diffs only	50.7 (+0.7)	53.8 (+2.0)
Diffs + execution evidence	49.8 (-0.2)	52.9 (+1.1)
Reasoning judge		
Diffs + execution evidence‡	48.5 (+2.6)	53.7 (+3.7)
Oracle (= pass@k)	57.1	62.9

D JUDGE PROMPT

System prompt: *You are a senior software engineer reviewing candidate patches for a repository task. Hidden tests will judge the patch. Prefer the candidate that implements exactly the requested interface and behaviour (function names, parameters, return values, error handling, response formats) with minimal, correct changes, and penalise candidates that add behaviour the task did not ask for, leave the task incomplete, or break existing behaviour. Decide from the diffs; test output, when given, is the candidate's own and may be incomplete.* The user message carries the task statement, then each candidate as `CANDIDATE` i with its diff and, in the evidence variant, the last test command it ran with the tail of its output, and ends with *reply with only one line: ANSWER:* n. The non-reasoning judge answers in at most 160 tokens; the reasoning judge has a 2 048-token budget. Candidate order is random per call.

E ADDITIONAL RESULTS

Figure 6 places every dimension on one accuracy–wall-clock plane, Figure 7 ranks the settings by resolved tasks per H200-hour, Figure 8 gives pass@k and per-task coverage, Figures 9 and 10 give the trajectory shape and where the output tokens go, and Table 4 lists every offline selector.

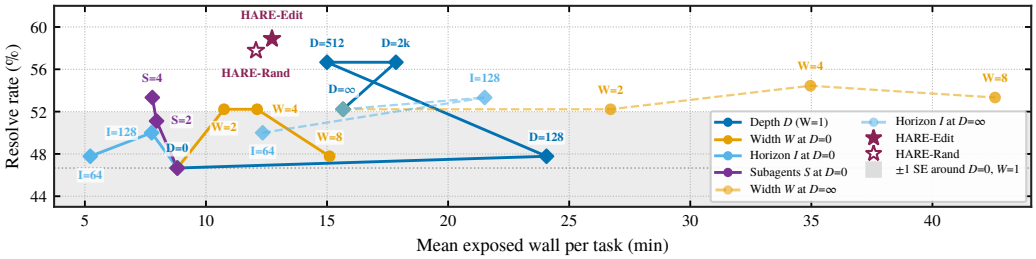


Figure 6: Resolve rate against mean wall-clock per task for every dimension scaled alone from the non-reasoning baseline (solid), the same dimensions on top of reasoning (dashed), and HARE-Edit and its two controls (stars). The grey band is one standard error around the baseline.

F INTERACTION AND COST MEASUREMENTS

Figure 11 draws the depth-by-width and depth-by-horizon interactions of Section 3.4 on one plane. A short horizon costs more when reasoning makes later turns productive (Section 3.4).

Figure 12 decomposes wall-clock into prefill, decode, tool execution, hook overhead, and residual server time, beside the modelled GPU-busy time. In the microbenchmark one decode step costs roughly 9–12 ms at $n = 1$ and about 11 ms at $n = 8$ for moderate contexts; depth adds sequential decode steps and horizon repeats model and tool work on the critical path, so parallelism can be attractive on wall-clock while consuming more GPU work.

G PAIRED TESTS

Every setting ran once, except the four re-run under a second seed in Section 5.1. Tables 5 and 6 compare each setting with the $W = 1$ baseline of its mode and with its designated comparators on the same 90 tasks: tasks won (resolved by the setting and not the baseline), lost, the net, the exact two-sided McNemar p on the discordant pairs, the share of tasks whose verdict flips, and, for settings with runs killed at the two-hour cap, the rate with those tasks re-run under an 18-hour cap; the suffixes pick1, raw and comp name the merge policy of Appendix B. The noise floor is the flip rate between runs that differ only in the seed or in a knob with no effect: the four seeds of the non-reasoning baseline flip 16–23% of the tasks pairwise, $D = 0$ against its 128-turn cap 14%, unbounded reasoning against its 128-turn cap or the 2 048 budget 17–18%, and HARE-Edit against HARE-Rand 8%. Against the non-reasoning baseline, the 512 budget, HARE-Edit, HARE-Edit with an unbounded budget, HARE-Rand, HARE-Edit + HARE-Scout, HARE-Width, and HARE-Scout in its repeated and random-turn variants separate at $p < 0.05$; against the unbounded chain no setting does, and against the same-cluster re-run of the non-reasoning baseline (52.2%) no second-campaign setting does.

H COST MODEL

Per-request fits. For every request of the campaign the profiler records the prompt tokens, the completion tokens summed over the $n = W$ choices, and the blocking wall-clock of the call; the server exposes no prefill or first-token time. Per width and mode we fit $\text{wall_ms} = a + c_p \cdot \text{prompt} + b \cdot \text{out}$ by least squares on requests with at least one output token and under ten minutes, once with the aggregate output tokens (b_{out}, per token over all branches) and once with the decode steps of the

Table 3: Every setting with its cost columns: resolve rate, mean wall-clock and modelled GPU-busy time per task, output tokens summed over sampled candidates, LLM calls, and resolved tasks per H200 wall-hour (res/h). Setting names: D = reasoning budget (0, 128, 512, 2048, full), W = width under pick-1, I = horizon, and HARE-Edit with its two controls HARE-Ends and HARE-Rand.

Setting	D	W	I	n	Rate (%)	Wall (min)	GPU-busy (min)	Out tok (k)	Calls	Res/h
Baselines										
D=0	0	1	300	90	46.7	8.8	5.5	24	91	3.18
D=∞	∞	1	300	90	52.2	15.7	14.2	61	64	2.00
D=0 (second cluster)	0	1	300	90	52.2	10.4	4.8	22	86	3.00
Depth D (W=1)										
D=128	128	1	300	90	47.8	24.1	10.5	40	102	1.19
D=512	512	1	300	90	56.7	15.0	12.9	52	90	2.27
D=2048	2048	1	300	90	56.7	17.8	15.8	66	84	1.91
Width at $D = 0$										
D=0, W=2	0	2	300	90	52.2	10.7	6.6	46	94	2.92
D=0, W=4	0	4	300	90	52.2	12.1	8.1	90	96	2.59
D=0, W=8	0	8	300	90	47.8	15.1	10.2	174	107	1.90
Width at $D = \infty$										
D=∞, W=2	∞	2	300	90	52.2	26.7	25.0	150	80	1.17
D=∞, W=4	∞	4	300	90	54.4	35.0	31.3	277	77	0.93
D=∞, W=8	∞	8	300	90	53.3	42.6	39.7	514	71	0.75
Horizon I										
D=0, I=64	0	1	64	90	47.8	5.2	3.2	15	55	5.49
D=0, I=128	0	1	128	90	50.0	7.8	4.2	19	75	3.87
D=∞, I=64	∞	1	64	90	50.0	12.3	11.9	52	52	2.43
D=∞, I=128	∞	1	128	90	53.3	21.5	13.8	59	64	1.49
Subagents S (up to S parallel delegated children, W=1)										
Passive S=2	0	1	300	90	51.1	8.0	4.8	22	82	3.85
Passive S=4	0	1	300	90	53.3	7.8	4.9	22	86	4.11
S=2, D=∞ (second cluster)	∞	1	300	90	54.4	18.0	15.4	65	73	1.81
HARE (Section 5.2; tokens include the drafts of regenerated turns)										
HARE-Edit	0/512	1	300	90	58.9	12.7	9.4	44	104	2.78
HARE-Ends	0/512	1	300	90	53.3	14.7	11.2	52	122	2.18
HARE-Rand	0/512	1	300	90	57.8	12.1	7.7	36	99	2.87
HARE-Edit, I=128	0/512	1	128	90	47.8	13.2	8.0	38	95	2.18
HARE-Edit, I=64	0/512	1	64	90	43.3	7.8	5.1	24	63	3.31
HARE-Edit, budget 128	0/512	1	300	90	52.2	13.1	8.4	39	110	2.39
HARE-Edit, unbounded budget	0/512	1	300	90	57.8	20.5	11.2	53	107	1.69
HARE-Edit, seed 2	0/512	1	300	90	54.4	11.5	7.6	36	97	2.85
Selective width at write turns (second cluster, 90 tasks)										
Sel. W=4, branch-0 control	0	1	300	90	54.4	15.6	11.4	83	111	2.09
HARE-Width	0	1	300	90	58.9	11.6	7.4	50	118	3.05
Sel. W=4, 512-token judge	0	1	300	90	47.8	11.5	7.6	48	101	2.50
4 $\times$ D=512 + vote	0	1	300	90	51.1	13.5	9.3	67	103	2.27
Uniform W=4, 512-token judge	0	1	300	90	53.3	14.6	10.4	87	117	2.19
Random-turn W=4, 512-token judge	0	1	300	90	50.0	12.8	7.9	46	110	2.35
HARE-Width, seed 2	0	1	300	90	52.2	12.4	6.4	41	99	2.52
Forced read-only scouts (second cluster, 90 tasks)										
Scouts, non-reasoning synthesis	0	1	300	90	48.9	8.4	6.3	29	144	3.48
HARE-Scout	0	1	300	90	53.3	9.0	6.6	30	146	3.56
HARE-Scout (repeated)	0	1	300	90	60.0	14.9	13.6	63	361	2.41
HARE-Scout (random turn)	0	1	300	90	55.6	9.7	7.0	32	153	3.44
HARE compositions (second cluster, 90 tasks)										
HARE-Edit + HARE-Scout	0	1	300	90	57.8	12.5	10.2	48	162	2.78
Judge stack, I=300	0	1	300	90	51.1	13.3	12.1	59	163	2.30
Judge stack, I=128	0	1	128	90	52.2	11.5	12.0	58	158	2.72
Seed replicates of D=0 (second cluster, 90 tasks)										
D=0, seed 2	0	1	300	90	51.1	8.2	4.7	21	86	3.72
D=0, seed 3	0	1	300	90	48.9	11.7	5.2	23	91	2.51
D=0, seed 4	0	1	300	90	50.0	10.2	5.3	23	96	2.95
D=0, seed 5	0	1	300	90	51.1	8.3	4.3	19	83	3.71
Sequential refinement, PDR-style (second cluster, 90 tasks)										
Sequential refinement	0	1	300	90	50.0	11.5	5.6	25	94	2.61

longest branch (c_d, per step; the longest branch's share of the aggregate comes from the width hook's per-branch length estimate). Table 7 gives the fits. At $W = 1$ both regressors coincide and $b = 12.8$ ms per token with reasoning and 11.9 without; as W grows, b_{out} per aggregate token falls (12.8, 9.1, 6.6, 4.1 ms with reasoning) while c_d per step stays at 11.7–12.8 ms, the signature of a memory-bound decode that batches the n branches. Without reasoning c_d drifts from 11.9 to 7.5 ms with a falling R^2 (0.51 at $W = 8$): the short non-reasoning answers make the longest-branch estimate, which comes from character counts, least reliable there. The fits use the grid settings only (no HARE

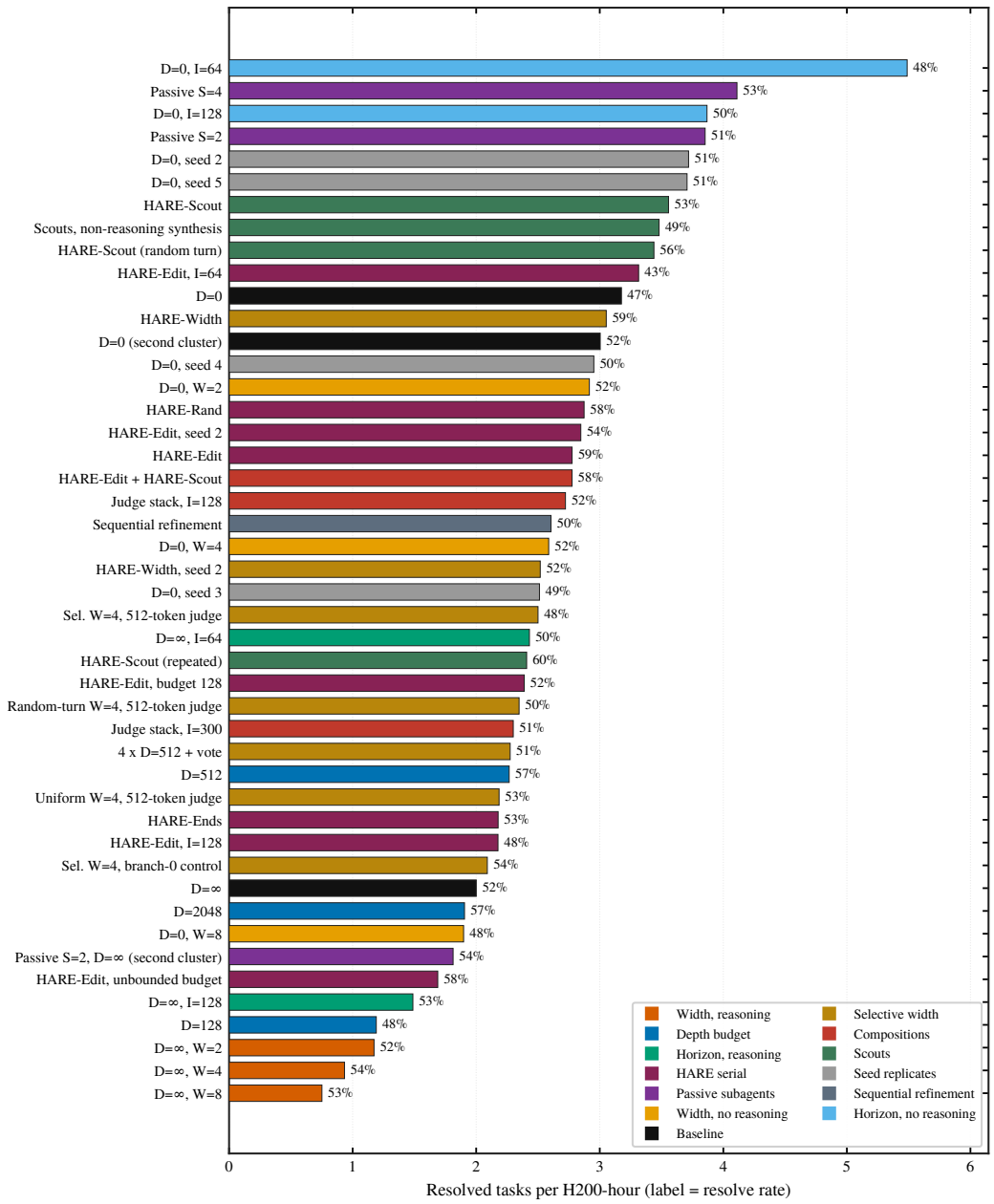


Figure 7: Left: resolved tasks per H200 wall-hour for every setting, sorted; the label above each bar is its resolve rate (suffix nr = no reasoning, r = reasoning). Right: resolve rate against wall-clock with the frontier (colour = block, marker size = W).

drafts, no children). When the calibrated prefill and decode of a run exceed its measured LLM time, we scale both down to it, so the decomposition of Figure 12 never exceeds the measured wall-clock.

Decomposition. Per run, we split the exposed wall-clock into prefill (the warm per-request prefill at the request's context length), decode ($c_d(W, \mathrm{ctx})$ times the decode steps), the remaining LLM time, the committed tool commands (at $W > 1$ from the width hook's per-turn timing, since the tool log also holds the shadow-executed alternatives), the width hook's overhead (the turn's exposed time beyond the committed command, including the compressed-merge call), and the rest. The modelled

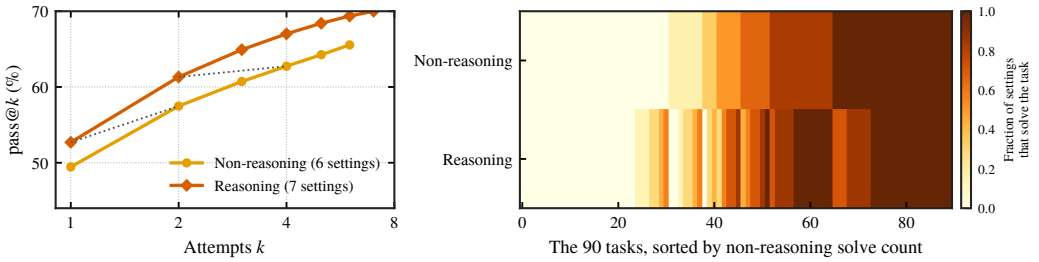


Figure 8: Left: pass@k over the settings of each mode (non-reasoning: D=0 at W=1, 2, 4, 8 and I=64, 128; reasoning: D=128, 512, 2048, ∞ at W=1, I=64, 128, and W=2); dotted lines connect reasoning@k to non-reasoning@$2k$, which coincide. Right: for each of the 90 tasks, sorted by non-reasoning solve count, the fraction of settings of each mode that solve it; no non-reasoning setting solves 31 tasks, all six solve 25, and 34 are coin flips.

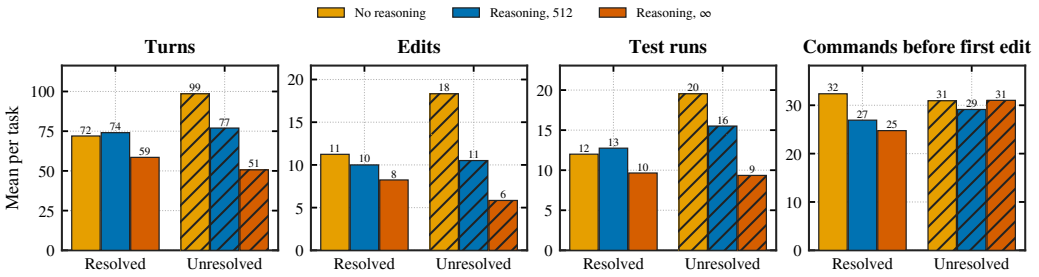


Figure 9: Trajectory shape per task for resolved and unresolved runs: turns, edits, test runs and commands before the first edit. Exploration is identical across modes; reasoning halves the number of edits and ends failing runs early.

GPU-busy time is prefill plus decode; the serial bound of Figure 12 charges $c_d(1, \mathrm{ctx})$ for every aggregate output token instead.

Microbenchmark. On the campaign's own server (one H200, the serving flags of Appendix A) we replay 24 real agent prompts rebuilt token-exactly from campaign trajectories, three per context length of 4k, 16k, 64k and 128k tokens and per mode, at $n \in \{1, 2, 4, 8\}$ with 256 pinned decode tokens and streaming, plus two and four concurrent independent requests (the subagent regime); 200 calls, 10 minutes, no preemption. Cold prefill costs 0.066–0.097 ms per token (271 ms at 4k, 12.3 s at 128k); the warm prefill of one request that appends an observation of about 2k tokens to a cached prefix costs 163, 253, 434 and 713 ms at the four context lengths (prefix-cache hit ratio 0.67–0.98), and this per-request cost is what the decomposition charges. A decode step costs 10.3, 10.5, 9.0 and 11.8 ms at $n = 1$ and 11.3, 11.3, 16.3 and 23.0 ms at $n = 8$ over the four context lengths, so eight branches batch for free up to 16k tokens and for twice the step time at 128k; aggregate throughput rises from 97 to 700 tokens per second at 4k and from 84 to 346 at 128k. Time to first token is 92–479 ms at $n = 1$ and 182–748 ms at $n = 8$. Two and four concurrent requests decode at 10.5–14 and 11.4–18 ms per step. The reasoning switch changes none of these numbers (11.8 against 11.9 ms at $n = 1$, 22.6 against 23.0 at $n = 8$, 128k). The decomposition charges every request its warm prefill at its context length and every decode step $c_d(W, \mathrm{ctx})$ interpolated in context.

I SUBAGENT MECHANISM

The parent agent gains one tool, `delegate_task`, that spawns up to S child agents from a goal and a context string. A child is a fresh conversation of the same model on the same server with the same reasoning switch and budget as the parent, the terminal tool, and its own turn budget of 50; children of one call run in parallel in the parent's container and working directory, and the parent blocks

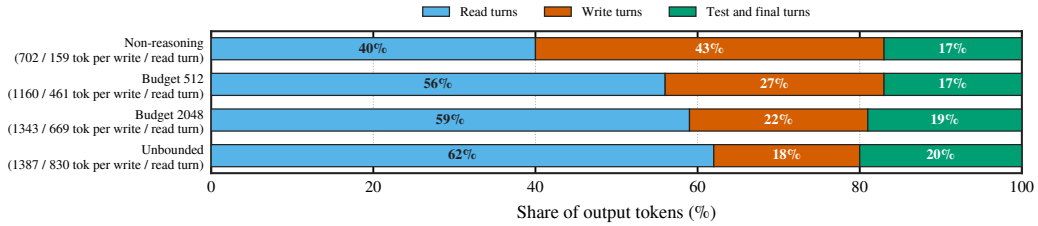


Figure 10: Share of output tokens by turn type, with the mean output tokens per write turn and per read turn in the row labels. With reasoning on, most tokens are spent before commands that only read the repository.

Table 4: Mechanical selectors over random k-subsets of the six non-reasoning baselines per task: resolve rate (%) of the selected patch against the random and oracle bounds.

Selector	k=1	k=2	k=3	k=4	k=6
Random (= pass@1)	49.8	48.4	49.9	49.3	49.6
Oracle (= pass@k)	49.8	57.1	61.1	62.9	65.6
Fewest edits	49.8	47.1	46.1	46.7	43.3
Fewest turns	49.8	48.3	48.6	49.8	50.0
Shortest wall	49.8	48.6	48.2	48.5	47.2
Most edits	49.8	51.5	51.5	52.1	51.1
Last test passed	49.8	49.5	48.8	49.2	47.8
Patch centrality (text)	49.8	49.6	51.3	50.2	50.0
Patch centrality (files)	49.8	49.6	51.2	50.2	49.3
Non-empty, shortest patch	49.8	50.1	49.1	49.6	48.9
Non-empty, longest patch	49.8	50.1	49.8	48.3	47.9

until all return a summary, which enters its context as the tool result. A child cannot delegate. The profiler logs every child request and command next to the parent's, tagged by role, and the per-child call count, duration and token use come back in the tool result. The task prompt carries one added sentence: *You may delegate up to S parallel subagents with the delegate_task tool for independent exploration or test runs; they share your container and repository and report back a summary.* A two-task pilot without that sentence produced no delegation in either mode, so the sentence is part of every subagent setting.

Delegation statistics ($D = 0$, 90 runs). At $S = 2$, 14 runs (16%) delegate, 11 once and 3 twice (20 children, 14 calls with one child and 3 with two); the first delegation comes at parent turn 26 on average. Children make 125 LLM calls and 136 commands over the 90 runs (6.2 and 6.8 per child), all 20 children exit with a completed summary, and the parent waits 7 seconds per task. At $S = 4$, 11 runs (12%) delegate once each (17 children, 7 calls with one child and 4 with more); children make 360 LLM calls and 466 commands (21 and 27 per child), 15 children complete and 2 hit their turn budget, and the parent waits 16 seconds per task. No child request carries a reasoning chain in either setting. With reasoning ($S = 2$, $D = \infty$), 2 of 90 runs delegate.

J HARE SERIAL IMPLEMENTATION AND REGENERATION STATISTICS

Algorithm 1 states the write-triggered rule and Table 8 lists the serial instantiation and its two controls end to end.

The rule is a wrapper around the chat-completion call of the agent. The hook parses the draft into its tool calls; a turn is a write turn when any command matches an in-place edit (`sed -i`), a shell redirection into a file, `tee`, `patch`, `git apply`, or a move, copy, delete or touch. The regenerated request differs from the draft only in the reasoning switch and the 512-token budget; the draft of a regenerated turn is never executed. Table 9 gives the per-task regeneration statistics from the event

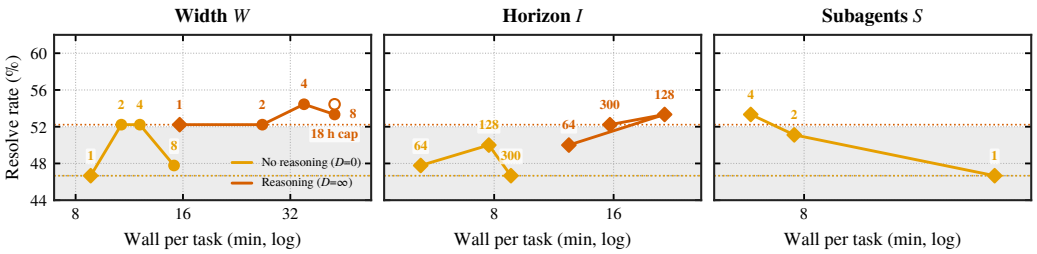


Figure 11: Measured interactions between depth and width or horizon. Width becomes far more expensive when every branch carries reasoning, while a short horizon is more damaging when reasoning makes later turns productive.

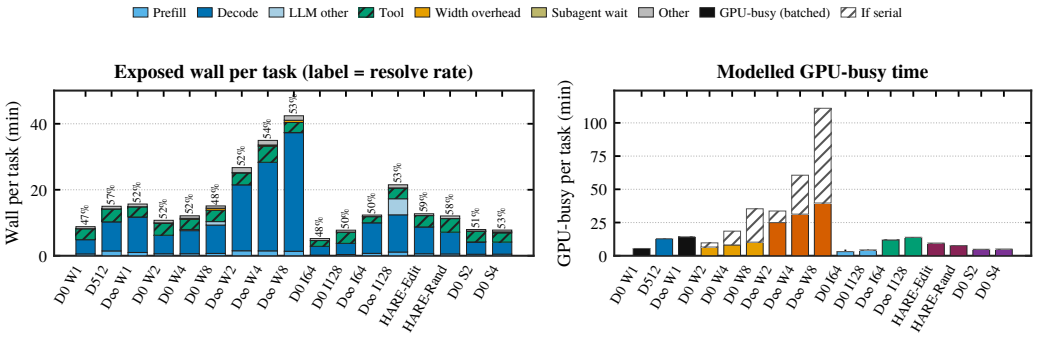


Figure 12: Where wall-clock and modelled GPU-busy time go. Depth extends serial decode. Width can batch decode across branches but still increases hardware work and pays the longest branch. Horizon repeats model and tool work sequentially.

log of every run. The regenerated answer differs from the draft on almost every write turn (99% for *edit*), so the reasoning chain changes the action itself.

K SECOND CAMPAIGN: COMPOSITION AND BASELINE SETTINGS

Task subset. The composition and baseline settings of Section 5 run on 90 tasks nested in the 146, drawn with the same procedure and seed 0, and the paper reports every setting on those 90, except the two repeated-scout compositions of Section 5.1, which report the 84 and 85 tasks they have graded. The per-repository quota is proportional to the 146 (nine each for openlibrary, navidrome, ansible, teleport, element-web and qutebrowser, eight each for flipt, vuls, webclients and NodeBB, four for tutanota), stratified within each repository on the pair of pilot outcomes. The subset resolves 56.7% (reasoning) and 48.9% (non-reasoning) in the pilot, against 56.2% and 48.6% on the 146. Every setting of the first campaign has a result on all 90 tasks, so we pair every comparison against a second-campaign setting on the same tasks; the standard error of a rate at $n = 90$ is ± 5.3 points.

Cluster. The second campaign runs on a second H200 cluster with the same container images, model weights, vLLM version and flags, per-task cap, and grader. Jobs are four hours long under a per-user cap on running GPU-hours; a task that a job's wall-time interrupts leaves no result and re-runs in the setting's next job, so every reported run is complete. A re-run of the non-reasoning baseline on the second cluster (*the second-cluster baseline*) measures the cluster effect on wall-clock (10.4 against 8.8 minutes), which lies inside the seed spread, so wall-clock comparisons across the clusters are unadjusted and Table 6 pairs every second-campaign setting against both baselines.

Settings. Table 10 lists the second-campaign settings, their mechanism, and the first-campaign settings that pair with each. Every mechanism wraps the agent's chat-completion call or tool dispatch and installs only when the run sets its setting variable, so the baseline binary is unchanged; a

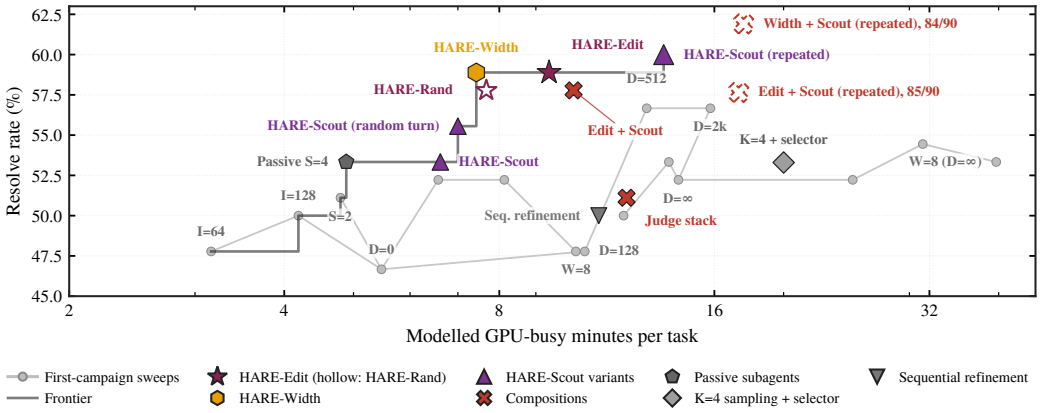


Figure 13: The frontier of Figure 4 against modelled GPU-busy minutes per task, with the same layering: grey first-campaign sweeps, coloured HARE instantiations and compositions, the frontier over every setting graded on SWE-Bench-Pro tasks, the two hollow crosses of Figure 4 at their graded tasks, and the diamond at the sum of the four trajectories' GPU-minutes.

Algorithm 1 HARE serial allocation

Require: context c_t, model $\mathcal{M}$ with a reasoning switch, budget D, write trigger $\mathrm{writes}(\cdot)$

$a \leftarrow \mathcal{M}(c_t;$ reasoning off$)$ ▷ draft
if $\mathrm{writes}(a)$ **then**
    $a \leftarrow \mathcal{M}(c_t;$ reasoning on, budget $D)$ ▷ same request, regenerated
end if
return a ▷ executed; the draft of a write turn is never executed

mislabelled combination (for example selective width together with HARE-Edit) refuses to start. Before its 90-task run, each mechanism passed an offline test against a fake model and container, then one live task on the second cluster. There we checked its verdict, tool-call latency, and per-mechanism event counts against the first-campaign non-reasoning run of the same task.

Sequential-refinement summary. The digest of the first trajectory (the task's first-campaign non-reasoning run) holds its final message without the reasoning chain, its patch (at most 12k characters), its last test command with the tail of the output, and its turn and command counts. The grading verdict is never part of it. The summarizer is the same model with reasoning off, temperature 0, and a 768-token limit, prompted as: *You are a terse engineering note-taker. A coding agent made one independent attempt at a repository task and stopped; a fresh attempt will now start from the ORIGINAL repository state (none of the previous changes exist). Summarize the previous attempt for the fresh one in at most 250 words, as four short sections: (1) what it changed (files, functions, the gist of the edit); (2) what its own test runs showed, if any; (3) what is likely wrong, missing or unverified in it; (4) concrete suggestions for the fresh attempt. Quote exact file paths, symbols and error text; never invent; never claim the attempt was correct. No preamble.* The second trajectory's task statement carries the summary under a note that the previous attempt may be wrong and that the agent must verify it. The summary request is one more request of the second trajectory, so its cost enters that trajectory's wall-clock and GPU time; the cost model (Appendix H) reports the policy's cost as the second trajectory's cost plus the first trajectory's GPU-minutes and its wall-clock scaled by the ratio of the second-cluster baseline to the first-campaign wall-clock on the 90 tasks (the only cluster-adjusted number in the paper, since that trajectory ran on the first cluster).

Cost accounting with children. A scout's requests are real requests of the same server and count in GPU-busy time. On the wall-clock side they overlap with the parent's wait, so the exposed decomposition of the cost model (Appendix H) charges the parent's blocked time once and removes the children's request and command time from the parent's LLM and tool components.

Table 5: Paired comparison of every setting with the $W = 1$ baseline of its mode and its designated comparators (90 tasks): first-campaign settings.

Setting	Baseline	Rate	Base	Won	Lost	Net	p	Flip (%)	Long cap
D=128	D=∞	47.8	52.2	6	10	-4	0.454	18	–
D=2048	D=∞	56.7	52.2	10	6	+4	0.454	18	–
D=512	D=∞	56.7	52.2	10	6	+4	0.454	18	–
D=0, W=2 (comp)	D=0	45.6	46.7	5	6	-1	1.000	12	–
D=0, W=2	D=0	52.2	46.7	10	5	+5	0.302	17	–
D=0, W=2 (raw)	D=0	51.1	46.7	7	3	+4	0.344	11	–
D=0, W=4 (comp)	D=0	44.4	46.7	4	6	-2	0.754	11	–
D=0, W=4	D=0	52.2	46.7	7	2	+5	0.180	10	–
D=0, W=4 (raw)	D=0	50.0	46.7	9	6	+3	0.607	17	–
D=0, W=8 (comp)	D=0	47.8	46.7	8	7	+1	1.000	17	–
D=0, W=8	D=0	47.8	46.7	7	6	+1	1.000	14	–
D=0, W=8 (raw)	D=0	44.4	46.7	6	8	-2	0.791	16	44.4
D=∞, W=2 (comp)	D=∞	54.4	52.2	7	5	+2	0.774	13	54.4
D=∞, W=2	D=∞	52.2	52.2	7	7	+0	1.000	16	53.3
D=∞, W=2 (raw)	D=∞	58.9	52.2	11	5	+6	0.210	18	–
D=∞, W=4 (comp)	D=∞	56.7	52.2	10	6	+4	0.454	18	56.7
D=∞, W=4	D=∞	54.4	52.2	8	6	+2	0.791	16	55.6
D=∞, W=4 (raw)	D=∞	53.3	52.2	8	7	+1	1.000	17	53.3
D=∞, W=8 (comp)	D=∞	51.1	52.2	7	8	-1	1.000	17	55.6
D=∞, W=8	D=∞	53.3	52.2	10	9	+1	1.000	21	54.4
D=∞, W=8 (raw)	D=∞	51.1	52.2	9	10	-1	1.000	21	–
D=0, I=128	D=0	50.0	46.7	8	5	+3	0.581	14	–
D=0, I=64	D=0	47.8	46.7	10	9	+1	1.000	21	–
D=∞, I=128	D=∞	53.3	52.2	8	7	+1	1.000	17	–
D=∞, I=64	D=∞	50.0	52.2	7	9	-2	0.804	18	–
HARE-Edit	D=0	58.9	46.7	13	2	+11	0.007	17	–
HARE-Edit	D=512	58.9	56.7	6	4	+2	0.754	11	–
HARE-Edit, budget 128	D=0	52.2	46.7	9	4	+5	0.267	14	–
HARE-Edit, budget 128	HARE-Edit	52.2	58.9	4	10	-6	0.180	16	–
HARE-Edit, unbounded budget	D=0	57.8	46.7	13	3	+10	0.021	18	–
HARE-Edit, unbounded budget	HARE-Edit	57.8	58.9	4	5	-1	1.000	10	–
HARE-Edit, I=128	D=0	47.8	46.7	6	5	+1	1.000	12	–
HARE-Edit, I=128	HARE-Edit	47.8	58.9	2	12	-10	0.013	16	–
HARE-Edit, I=128	D=0, I=128	47.8	50.0	5	7	-2	0.774	13	–
HARE-Edit, I=64	D=0	43.3	46.7	5	8	-3	0.581	14	–
HARE-Edit, I=64	HARE-Edit	43.3	58.9	2	16	-14	0.001	20	–
HARE-Edit, seed 2	D=0	54.4	46.7	11	4	+7	0.118	17	–
HARE-Edit, seed 2	HARE-Edit	54.4	58.9	2	6	-4	0.289	9	–
HARE-Edit, seed 2	D=0, seed 5	54.4	51.1	8	5	+3	0.581	14	–
HARE-Edit, seed 2	D=0 (second cluster)	54.4	52.2	9	7	+2	0.804	18	–
HARE-Ends	D=0	53.3	46.7	9	3	+6	0.146	13	–
HARE-Rand	D=0	57.8	46.7	13	3	+10	0.021	18	–
HARE-Rand	HARE-Edit	57.8	58.9	3	4	-1	1.000	8	–
Passive S=2	D=0	51.1	46.7	10	6	+4	0.454	18	–
Passive S=4	D=0	53.3	46.7	9	3	+6	0.146	13	–
Passive S=4	Passive S=2	53.3	51.1	7	5	+2	0.774	13	–
S=2, D=∞ (second cluster)	D=∞	54.4	52.2	9	7	+2	0.804	18	–
S=2, D=∞ (second cluster)	Passive S=2	54.4	51.1	10	7	+3	0.629	19	–

Table 6: Paired comparisons, continued: second-campaign settings.

Setting	Baseline	Rate	Base	Won	Lost	Net	p	Flip (%)	Long cap
4 × D=512 + vote	D=0	51.1	46.7	9	5	+4	0.424	16	–
4 × D=512 + vote	Sel. W=4, 512-token judge	51.1	47.8	9	6	+3	0.607	17	–
4 × D=512 + vote	D=0 (second cluster)	51.1	52.2	9	10	-1	1.000	21	–
Uniform W=4, 512-token judge	D=0	53.3	46.7	9	3	+6	0.146	13	53.3
Uniform W=4, 512-token judge	Sel. W=4, 512-token judge	53.3	47.8	9	4	+5	0.267	14	53.3
Uniform W=4, 512-token judge	D=0, W=4	53.3	52.2	4	3	+1	1.000	8	53.3
Uniform W=4, 512-token judge	D=0 (second cluster)	53.3	52.2	7	6	+1	1.000	14	53.3
Sel. W=4, branch-0 control	D=0	54.4	46.7	9	2	+7	0.065	12	–
Sel. W=4, branch-0 control	HARE-Edit	54.4	58.9	2	6	-4	0.289	9	–
Sel. W=4, branch-0 control	D=0 (second cluster)	54.4	52.2	8	6	+2	0.791	16	–
HARE-Width	D=0	58.9	46.7	15	4	+11	0.019	21	–
HARE-Width	HARE-Edit	58.9	58.9	5	5	+0	1.000	11	–
HARE-Width	D=0 (second cluster)	58.9	52.2	11	5	+6	0.210	18	–
HARE-Width, seed 2	D=0	52.2	46.7	12	7	+5	0.359	21	–
HARE-Width, seed 2	HARE-Width	52.2	58.9	3	9	-6	0.146	13	–
HARE-Width, seed 2	HARE-Edit, seed 2	52.2	54.4	6	8	-2	0.791	16	–
HARE-Width, seed 2	D=0, seed 5	52.2	51.1	9	8	+1	1.000	19	–
HARE-Width, seed 2	D=0 (second cluster)	52.2	52.2	8	8	+0	1.000	18	–
Sel. W=4, 512-token judge	D=0	47.8	46.7	7	6	+1	1.000	14	–
Sel. W=4, 512-token judge	HARE-Edit	47.8	58.9	1	11	-10	0.006	13	–
Sel. W=4, 512-token judge	D=512	47.8	56.7	4	12	-8	0.077	18	–
Sel. W=4, 512-token judge	D=0 (second cluster)	47.8	52.2	5	9	-4	0.424	16	–
Random-turn W=4, 512-token judge	D=0	50.0	46.7	8	5	+3	0.581	14	–
Random-turn W=4, 512-token judge	Sel. W=4, 512-token judge	50.0	47.8	7	5	+2	0.774	13	–
Random-turn W=4, 512-token judge	D=0 (second cluster)	50.0	52.2	8	10	-2	0.815	20	–
Scouts, non-reasoning synthesis	D=0	48.9	46.7	6	4	+2	0.754	11	–
Scouts, non-reasoning synthesis	HARE-Scout	48.9	53.3	2	6	-4	0.289	9	–
Scouts, non-reasoning synthesis	D=0 (second cluster)	48.9	52.2	7	10	-3	0.629	19	–
HARE-Scout	D=0	53.3	46.7	7	1	+6	0.070	9	–
HARE-Scout	D=512	53.3	56.7	2	5	-3	0.453	8	–
HARE-Scout	HARE-Edit	53.3	58.9	1	6	-5	0.125	8	–
HARE-Scout	D=0 (second cluster)	53.3	52.2	8	7	+1	1.000	17	–
HARE-Scout	Sel. W=4, 512-token judge	53.3	47.8	7	2	+5	0.180	10	–
HARE-Scout (repeated)	D=0	60.0	46.7	14	2	+12	0.004	18	–
HARE-Scout (repeated)	HARE-Scout	60.0	53.3	7	1	+6	0.070	9	–
HARE-Scout (repeated)	D=0 (second cluster)	60.0	52.2	11	4	+7	0.118	17	–
HARE-Scout (random turn)	D=0	55.6	46.7	10	2	+8	0.039	13	–
HARE-Scout (random turn)	HARE-Scout	55.6	53.3	4	2	+2	0.688	7	–
HARE-Scout (random turn)	D=0 (second cluster)	55.6	52.2	10	7	+3	0.629	19	–
Judge stack, I=128	D=0	52.2	46.7	11	6	+5	0.332	19	–
Judge stack, I=128	Judge stack, I=300	52.2	51.1	7	6	+1	1.000	14	–
Judge stack, I=128	HARE-Edit, I=128	52.2	47.8	9	5	+4	0.424	16	–
Judge stack, I=128	D=0 (second cluster)	52.2	52.2	9	9	+0	1.000	20	–
Judge stack, I=300	D=0	51.1	46.7	8	4	+4	0.388	13	–
Judge stack, I=300	Sel. W=4, 512-token judge	51.1	47.8	9	6	+3	0.607	17	–
Judge stack, I=300	HARE-Scout	51.1	53.3	4	6	-2	0.754	11	–
Judge stack, I=300	HARE-Edit	51.1	58.9	3	10	-7	0.092	14	–
Judge stack, I=300	D=512	51.1	56.7	3	8	-5	0.227	12	–
Judge stack, I=300	D=0 (second cluster)	51.1	52.2	8	9	-1	1.000	19	–
HARE-Edit + HARE-Scout	D=0	57.8	46.7	11	1	+10	0.006	13	–
HARE-Edit + HARE-Scout	HARE-Edit	57.8	58.9	3	4	-1	1.000	8	–
HARE-Edit + HARE-Scout	HARE-Scout	57.8	53.3	5	1	+4	0.219	7	–
HARE-Edit + HARE-Scout	D=0 (second cluster)	57.8	52.2	10	5	+5	0.302	17	–
D=0, seed 2	D=0	51.1	46.7	8	4	+4	0.388	13	–
D=0, seed 2	D=0 (second cluster)	51.1	52.2	7	8	-1	1.000	17	–
D=0, seed 3	D=0	48.9	46.7	8	6	+2	0.791	16	48.9
D=0, seed 3	D=0 (second cluster)	48.9	52.2	8	11	-3	0.648	21	48.9
D=0, seed 4	D=0	50.0	46.7	7	4	+3	0.549	12	–
D=0, seed 4	D=0 (second cluster)	50.0	52.2	6	8	-2	0.791	16	–
D=0, seed 5	D=0	51.1	46.7	6	2	+4	0.289	9	–
D=0, seed 5	D=0 (second cluster)	51.1	52.2	8	9	-1	1.000	19	–
Sequential refinement	D=0	50.0	46.7	5	2	+3	0.453	8	50.0
Sequential refinement	D=0 (second cluster)	50.0	52.2	6	8	-2	0.791	16	50.0

Table 7: Per-request fits of the blocking wall-clock per width and mode: intercept, prefill cost per prompt token, decode cost per aggregate output token, and decode cost per step of the longest branch.

W	Mode	Requests	a (ms)	c_p (ms/tok)	b_{out} (ms/agg. tok)	R^2	c_d (ms/step)	R^2
1	No reasoning	27,612	-7	0.0083	11.917	0.847	11.917	0.847
1	Reasoning	41,038	179	0.0201	12.784	0.826	12.784	0.826
2	No reasoning	22,227	-230	0.0082	7.768	0.925	10.767	0.893
2	Reasoning	19,302	-2922	0.0288	9.086	0.936	11.703	0.991
4	No reasoning	21,199	-662	0.0112	5.466	0.802	9.493	0.593
4	Reasoning	17,842	-3253	0.0420	6.584	0.727	12.289	0.816
8	No reasoning	21,137	-734	0.0239	3.330	0.753	7.540	0.510
8	Reasoning	15,935	-3744	0.0768	4.052	0.792	12.116	0.935

Table 8: The serial instantiation and its two controls end to end on the 90 tasks. Wall, GPU-busy, and tool time are minutes per task; output tokens are thousands per task.

System	Rate (%)	Wall (min)	GPU-busy (min)	Tool (min)	Out tok (k)	LLM calls	Reasoning turns (%)	Net vs D=0	p	Res/ h
D=0, one attempt	46.7	8.8	5.5	3.3	24	91	0	–	–	3.18
D=∞, unbounded reasoning	52.2	15.7	14.2	3.1	61	64	100	+5 (10 / 5)	0.30	2.00
D=512	56.7	15.0	12.9	3.9	52	90	100	+9 (10 / 1)	0.01	2.27
HARE-Edit	58.9	12.7	9.4	3.4	44	104	13	+11 (13 / 2)	0.007	2.78
HARE-Ends	53.3	14.7	11.2	3.7	52	122	18	+6 (9 / 3)	0.15	2.18
HARE-Rand (control)	57.8	12.1	7.7	4.1	36	99	16	+10 (13 / 3)	0.02	2.87

Table 9: Regeneration statistics per task (means over the 90 tasks).

Variant	turns	regenerated	share (%)	same action as draft (%)	regenerated output tokens (k)
Edit	91.7	12.3	13	1	12.0
Ends	103.9	18.5	18	12	14.8
Rand	85.5	13.5	16	5	7.5

Table 10: Second-campaign settings (90 tasks each). All run the non-reasoning agent at $I = 300$ unless stated; an unqualified "judge" or "synthesis" is one 512-token reasoning call.

Setting	mechanism	paired against
Cluster calibration		
Second-cluster baseline	the non-reasoning baseline re-run on the second cluster	D=0
Serial allocation		
HARE-Edit, I=128 / I=64	write-triggered 512-token reasoning, shorter horizon	HARE-Edit; D=0 at I=128 for the 128 setting
HARE-Edit, budget 128 / unbounded	write-triggered reasoning with a 128-token or unbounded budget	HARE-Edit
Selective width at write turns (W=4)		
HARE-Width / 512-token judge	non-reasoning judge (D_r=0) or 512-token judge over four non-reasoning proposals	HARE-Edit; also D=512 for the judge
Branch-0 cost control	four 512-token proposals, first candidate committed: HARE-Edit's answer under an n=4 request	HARE-Edit
Reasoning proposals + vote	four 512-token proposals, majority vote	judge
Uniform / random width	the judge setting on every turn / on a seeded 15% of turns	judge; also D=0, W=4 (pick-1) for uniform
Scouts (S=4, read-only)		
Non-reasoning synthesis / HARE-Scout	one round after the first tool turn, D_r=0 / D_r=512	D=0 / (D=512, HARE-Edit)
HARE-Scout, repeated / random turn	a round per three-turn read-only stall (at most four) / one round at a seeded turn	HARE-Scout
Passive S=4 at D=0 / passive S=2 at D=∞	the passive subagent axis, wider and with reasoning	passive S=2 / (passive S=2, D=∞)
Compositions		
HARE-Edit + HARE-Scout	write-triggered reasoning and one scout round	HARE-Edit, HARE-Scout
Judge stack, I=300	selective width with a 512-token judge and one scout round	512-token judge, HARE-Scout, HARE-Edit, D=512
Judge stack, I=128	the same at I=128	judge stack at I=300, HARE-Edit at I=128
Prior-work policies		
Whole-trajectory sampling	three further seeds of the non-reasoning baseline; with the second-cluster baseline the K=4 pool	the second-cluster baseline
Sequential refinement	second trajectory conditioned on a summary of the first-campaign non-reasoning run	the second-cluster baseline