FUNNEL: PUBLIC-CHECK-GUIDED BUDGET ALLOCATION FOR NATIVE AGENT HARNESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Batch-oriented agent systems often improve task success by repeating complete trajectories, but uniform candidate budgets spend additional computation on both tasks that already pass a check and tasks that still require rescue. We introduce FUNNEL, a training-free, task-level budget scheduler that operates outside a native agent harness. FUNNEL runs one complete native trajectory for every task, then uses only protocol-permitted public-check outcomes to launch seven additional state-isolated trajectories for first-check failures. Candidates share neither mutable execution state nor feedback, and a fixed selector freezes the final candidate before independent verification. Across four configurations spanning two models, two harnesses, and two task families, Funnel-8 differs from fixed eight-candidate search by only -0.52 to $+1.04$ percentage points in final pass rate while reducing the number of generated candidates by 28.3–58.6% and reported GPU-hours by 18.2–48.2%. In matched Qwen3-32B/HERMES comparisons on Reasoning Gym and LiveCodeBench-v6, Funnel remains within -0.26 to $+1.30$ points of ASC and DSC while using fewer candidates and reported GPU-hours. A fixed-average-four UAB adaptation uses a slightly lower budget but trails Funnel by 3.12–7.03 points. These results show that concentrating attempts on first-check failures provides a competitive accuracy–cost operating point without changing native trajectories, while candidate savings do not imply proportional savings in GPU-hours.

1 INTRODUCTION

An agent's final performance depends not only on its underlying language model, but also on the harness that organizes tool use, context, execution state, and interaction with the environment. ReAct demonstrates the value of interleaving reasoning with action, while SWE-agent shows that the agent–computer interface itself shapes software-engineering behavior (Yao et al., 2022; Yang et al., 2024). The harness is therefore part of the experimental object rather than a neutral wrapper (Yao et al., 2026). A complete agent trajectory may involve many model requests, tool calls, and state changes, making it substantially more expensive than an ordinary text sample. Repeating complete trajectories can increase the chance of task success, but assigning the same candidate budget to every task also spends additional computation on tasks whose first attempt already satisfies an available check. The central question is thus not merely whether more attempts help, but which tasks still warrant them when the native harness execution process must remain unchanged.

Existing test-time compute methods already allocate inference non-uniformly. Adaptive-Consistency (ASC) uses agreement among candidate answers to decide when to stop; Difficulty-Adaptive Self-Consistency (DSC) estimates difficulty and assigns different sampling budgets; and Uncertainty-Aware Budget Allocation (UAB) uses information from an initial generation to distribute a fixed global budget (Aggarwal et al., 2023; Wang et al., 2025; Nguyen et al., 2026). More broadly, test-time scaling studies show that prompts need not benefit equally from additional inference (Snell et al., 2025). These methods establish the value of heterogeneous budgets, but their allocation signals are primarily based on answer agreement, estimated difficulty, or generation uncertainty. A complete agent trajectory additionally contains tool interactions, a mutable workspace, and environment state. This leaves a narrower interface question: when the task protocol already exposes a permissible post-execution check, can that signal directly allocate complete trajectories without predicting difficulty, estimating a reward distribution, or altering the native harness execution?

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.

We introduce FUNNEL, a training-free task-level budget scheduler deployed outside the native agent harness. FUNNEL first runs one complete native trajectory for every task in a batch and applies a protocol-permitted public check. A task that passes stops with its first candidate; a task that fails receives seven additional complete trajectories. Every rescue trajectory starts from an independent initial state: candidates share neither mutable workspace state nor feedback, and checker output is not injected into later attempts. FUNNEL therefore changes the candidate budget across tasks, not the reasoning or execution process within a trajectory. The public check is used only for scheduling and candidate selection and is distinct from the hidden final evaluation. The candidate set and selected candidate ID are frozen before independent final verification of each task.

This design yields a simple one-or-eight budget contract. Under the same fixed eight-candidate pool, public-check procedure, and deterministic selector, generating candidates two through eight cannot change the selected answer when the first candidate already passes; when it fails, FUNNEL exposes the full candidate set to the same selector. This selection-preservation property explains which trajectories are unnecessary under the specified selection rule. It is a property of the budget mechanism, not a guarantee that independent executions will produce identical realized outcomes, nor a claim that first-check failure is an oracle measure of intrinsic difficulty. Across fixed, random, and adaptive baselines, our experiments show that this restricted intervention provides a competitive accuracy–cost operating point while preserving native trajectory execution.

Contributions.

- **A public-check-guided budget interface for complete agent trajectories.** We combine batch-wide first attempts with selective rescue while leaving model parameters and native harness logic unchanged and preserving cross-candidate state isolation.
- **An explicit one-or-eight contract.** We formalize the task-level budget rule and its fixed-pool selection-preservation property while separating public checking, candidate selection, and independent final verification.
- **A controlled evaluation of allocation strategies.** We compare FUNNEL with fixed-four, fixed-eight, random-slot, ASC, DSC, and UAB allocations under common model–harness interfaces, reporting selected-answer performance separately from candidate and GPU-hour costs.

2 RELATED WORK

2.1 REPEATED SAMPLING AND FEEDBACK

Chain-of-thought prompting and self-consistency distinguish reasoning within a sampled response from aggregation across responses (Wei et al., 2022; Wang et al., 2022). Large Language Monkeys examines how repeated sampling scales solution coverage (Brown et al., 2024). FUNNEL also generates alternative candidates, but its comparison unit is different: it operates on complete harness executions rather than response samples. Each candidate can include tool calls, environment interaction, and workspace state, yet contributes only one selected result to the reported score. Its primary outcome is therefore the independently verified answer selected from the candidate set, not oracle coverage.

Tree of Thoughts and Language Agent Tree Search explore intermediate reasoning or environment feedback (Yao et al., 2023; Zhou et al., 2023); Self-Refine and Reflexion revise outputs or carry feedback across trials (Madaan et al., 2023; Shinn et al., 2023). In contrast, FUNNEL starts every rescue trajectory from an isolated initial state, without earlier candidates or scheduler-generated critiques. Rescue means independent complete attempts, not answer repairs.

2.2 ADAPTIVE BUDGET ALLOCATION

ASC and Early-stopping Self-Consistency stop based on sampled-answer agreement (Aggarwal et al., 2023; Li et al., 2024). FUNNEL checks the first complete trajectory and expands failures to eight complete runs. DSC ranks difficulty and pre-allocates response samples (Wang et al., 2025); UAB uses initial-generation probabilities to distribute a fixed remaining budget (Nguyen et al., 2026). FUNNEL also starts with broad coverage, but a binary public check determines one or eight complete harness trajectories. Its realized candidate total depends on first-check failures.

A broader literature studies prompt-dependent compute, reward-distribution estimates, optimal stopping, and bandit allocation (Snell et al., 2025; Raman et al., 2025; Kalayci et al., 2025; Zuo & Zhu, 2026). These different observation models do not establish optimality of FUNNEL's fixed rescue batch: a binary failure alone does not distinguish recoverable from persistently unsolved tasks. Comparisons require a common execution interface and accounting for auxiliary allocation costs.

2.3 AGENT EXECUTION, VERIFICATION, AND COST

Learned verifiers can select among mathematical solutions, and process supervision provides feedback on intermediate reasoning steps (Cobbe et al., 2021; Lightman et al., 2024). FUNNEL does not train a verifier or introduce process supervision. Its checker is an externally specified interface whose evidence is permitted for scheduling and selection. Independent final verification begins only after the candidate set and selected identifier have been frozen for each task.

For code, CodeT exploits generated tests and execution agreement; AlphaCodium organizes generation and refinement around tests; and S* combines parallel and sequential generation with candidate-discriminating inputs (Chen et al., 2022; Ridnik et al., 2024; Li et al., 2025). These approaches can change generation prompts, repair loops, or selection evidence, so they are not automatically interchangeable with an outer budget scheduler that preserves every native trajectory. When adapting a sampling method to an agent harness, its stopping signal and final selector must be stated separately. Answer agreement is also task-dependent: different programs or workspace artifacts can be behaviorally equivalent without having identical strings or file layouts.

The distinction between a public check and final correctness is essential. EvalPlus shows why limited code tests can miss errors, while ImpossibleBench studies agents' propensity to exploit test cases (Liu et al., 2023; Zhong et al., 2026). These findings motivate an explicit information boundary: the scheduler may use only protocol-permitted evidence available before selection is frozen. Freezing alone does not establish robustness, and FUNNEL neither treats public-pass counts as final accuracy nor uses final-verifier outcomes to select a different candidate after the fact.

AgentBench, WebArena, and SWE-bench evaluate heterogeneous, web, and repository-level agents (Liu et al., 2024; Zhou et al., 2024; Jimenez et al., 2024). They show that outputs can include environment state and artifacts, not just final text; these are background benchmarks, not evaluations here. We evaluate fixed subsets of Reasoning Gym and LiveCodeBench-v6 and a Harness-Bench panel (Stojanovski et al., 2026; Jain et al., 2025; Yao et al., 2026). The first two use selected-answer pass rates; Harness-Bench uses mean outcome score. We report these metrics separately.

FrugalGPT reduces cost through model cascades, while RouteLLM learns routing decisions from preference data (Chen et al., 2023; Ong et al., 2025). FUNNEL holds the model–harness pair fixed and changes the number of independent attempts, not the model chosen for an attempt. Serving systems such as vLLM address memory management and throughput at a different layer (Kwon et al., 2023). Algorithmic candidate savings do not determine wall-clock savings independently of trajectory lengths, tool execution, concurrency, and serving behavior. Following the motivation of Kapoor et al. (2024), we therefore report selected-answer performance, candidate counts, and reported GPU-hours separately, without treating any one as a substitute for the others.

3 METHOD

3.1 EXECUTION UNIT AND AVAILABLE INFORMATION

Let $X = \{x_1, \ldots, x_N\}$ be a task set, H a native harness, and M a model. Candidate c_{ij} is a complete native trajectory produced from a clean initial state with candidate seed s_j:

$$c_{ij} = H(M, x_i, s_j), \qquad [p_{ij}, q_{ij}] = P(x_i, c_{ij}), \tag{1}$$

where P returns a public pass flag p_{ij} and a predefined public score vector $\mathbf{q}_{ij}$. A candidate includes its final answer and the associated execution evidence. The seed s_j indexes a candidate within the task, not an independently replicated experiment. Each experimental configuration is run once, although that run may generate multiple candidates for every task.

Public means available to the host scheduler under the task protocol; it does not imply that all checker information is included in model context. The final verifier V supplies the final evaluation only after

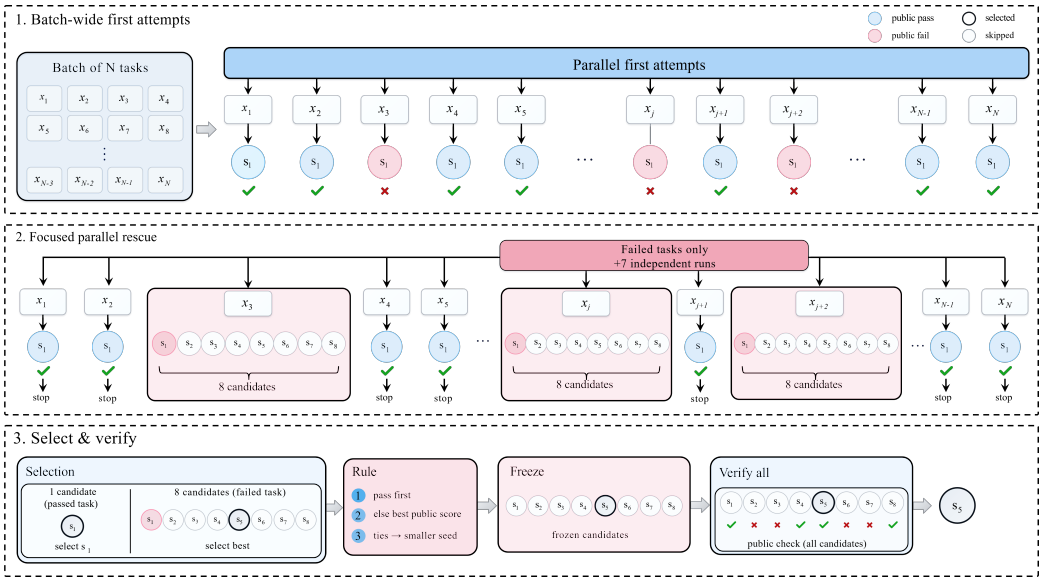


Figure 1: The FUNNEL workflow. First attempts run in parallel, and the public check partitions tasks into stop and rescue groups. A passing task retains one candidate; each failed task receives seven additional state-isolated trajectories. The fixed rule then selects from the resulting one- or eight-candidate set and freezes the selected identifier before independent verification. Verification can examine every frozen candidate, but the reported outcome is that of the selected candidate.

the candidate set and selected ID have been frozen. Checking and final evaluation are different stages: a public pass does not guarantee a final pass. The method therefore addresses tasks with permitted execution feedback, not verifier-free open-ended generation. The scheduler adds no cross-candidate feedback, although the native harness can still use its own tools and observations within a trajectory.

3.2 FIRST ATTEMPT AND ONE-BATCH RESCUE

Every task first executes c_{i1}. If $p_{i1} = 1$, the task stops and retains its first candidate. Otherwise, it enters a rescue queue and receives seven additional complete runs associated with $s_2, \ldots, s_8$. The queue follows first-launch order rather than failure-return order. The seven rescue trajectories form one execution batch but maintain separate state: they do not share a mutable workspace or use other candidates as feedback. The rescue batch does not stop adaptively after individual candidates.

At the task-set level, this is broad first-attempt coverage followed by concentrated candidate generation for the unresolved subset. The concentration is in the number of complete attempts per task, not the depth of reasoning or the number of revisions. This logical two-stage structure alone does not specify an optimal concurrency policy or establish a reduction in batch completion time.

Let F be the number of tasks failing the first public check. The task-level budget and total candidate count are

$$b_i = 1 + 7\mathbb{I}[p_{i1} = 0], \qquad C_{\text{Funnel}} = N + 7F. \tag{2}$$

Fixed-eight search uses $8N$ candidates, so under the exact one-or-eight contract,

$$1 - \frac{C_{\text{Funnel}}}{8N} = \frac{7}{8}\left(1 - \frac{F}{N}\right). \tag{3}$$

This saving is an algebraic property of the allocation rule, independent of later answer correctness. It does not imply a proportional saving in GPU-hours. The total is an outcome of the first-check partition, rather than an externally fixed budget that the scheduler must exhaust.

3.3 FIXED SELECTION AND INDEPENDENT FINAL SCORING

Selection first considers publicly passing candidates and chooses the one with the smallest seed order. If none passes, it chooses the lexicographically largest public score vector, breaking ties by the

smaller seed. A stopped task has one candidate and a rescued task has eight. Selection cannot read final-verifier outcomes; reported performance measures only the chosen candidate.

After candidate sets and selection records are frozen, an independent verifier examines all generated candidates. For binary final outcomes, the reported accuracy is

$$\mathrm{Acc}_{\mathrm{sel}} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}\big[V(x_i, c_{i,j_i^*}) = 1\big]\,. \tag{4}$$

where j_i^* is the already-frozen selected identifier. Benchmarks with non-binary outcomes report their prescribed mean outcome score rather than this binary accuracy. When public checking and final judgment disagree, the final judgment determines the primary outcome.

3.4 SELECTION-REDUNDANT COMPUTATION

Proposition 1 (Selection Preservation under a Fixed Candidate Pool) *For each task, fix the same ordered pool of eight candidates for* FUNNEL *and Depth-8, with s_1 the smallest seed. Assume that both methods use the same public check and selector, and that scheduling changes neither the candidates nor their public-check outcomes. Then* FUNNEL *and Depth-8 select the same candidate identifier.*

Proof. If the first candidate passes publicly, both methods select it because s_1 has the smallest seed order. If it fails, FUNNEL generates the remaining seven candidates, after which both methods apply the same selector to the same pool of eight. They therefore select the same candidate in either case. □

The proposition explains why FUNNEL can treat some subsequent generation as *selection-redundant computation*. Under the specified selector, once the first candidate passes publicly, candidates two through eight cannot change the selected identifier; omitting them preserves the selection that would have been made from the fixed pool. FUNNEL therefore does not merely predict that an apparently easy task needs fewer samples. It uses an observed public pass to identify candidate generation that is redundant for selection, while leaving each generated trajectory unchanged.

4 EXPERIMENTAL SETUP

Experimental configurations. We evaluate fixed-budget search, random allocation, and adaptive allocation. On Reasoning Gym, the core Qwen3-32B/HERMES configuration compares Native, Depth-4, Depth-8, Funnel-8, and Random-slot. Three further configurations change the task to LiveCodeBench-v6, the model to gpt-oss-20b, or the harness to NANOBOT; each compares the same five strategies (Yang et al., 2025; Agarwal et al., 2025; Nous Research, 2026; HKUDS, 2026). These four primary configurations are not a full factorial design; Table 1 also includes three additional complete configurations.

Reasoning Gym uses 384 fixed tasks and LiveCodeBench-v6 uses 96 (Stojanovski et al., 2026; Jain et al., 2025). Harness-Bench has 32 tasks. Each configuration is run once, so we do not estimate between-run variation or infer paired significance from aggregate counts.

Baselines and adaptations. Native generates one complete candidate per task; Depth-4 and Depth-8 generate four and eight. "Depth" means candidate budget, not within-trajectory reasoning depth. Funnel-8 generates one candidate for all tasks and seven additional candidates only for first-check failures. Random-slot assigns one or eight candidates before observing outcomes and uses the same final selector.

Adaptive comparisons use Qwen3-32B/HERMES on all three task panels. ASC stops when the Beta posterior probability that its leading normalized answer is dominant exceeds 0.95, capped at eight candidates. DSC ranks shuffled eight-task batches over five rounds and pre-samples the predicted-hard partition; each task receives at most eight candidates. UAB uses first-trajectory token log probabilities to allocate four candidates per task on average, capped at eight. Normalized answer agreement and generation probabilities are adaptation-specific signals, unlike a public check.

To isolate allocation, the adaptive methods use FUNNEL's public-evidence selector instead of their original aggregators: the earliest public pass, or otherwise the highest predefined public score

Table 1: Fixed- and random-budget results: final selected-answer accuracy and recorded cost. RG contains 384 tasks and LCB-v6 contains 96 tasks per run. Each row is one run. The three additional gpt-oss-20b configurations use separately frozen Random-slot reference budgets that are not candidate-matched to Funnel-8; Hermes/RG is candidate-matched. GPU-hours reproduce recorded per-candidate GPU seconds for the additional configurations and the reported field for the primary configurations; they are not asserted to measure active-device hours, FLOPs, or energy.

Model	Harness / Benchmark	Strategy	Acc. (%)	Candidates	GPU-h
Qwen3-32B	Hermes/RG	Native	50.52	384	8.84
		Depth-4	70.31	1,536	34.18
		Depth-8	77.08	3,072	67.48
		Funnel-8	76.56	1,623	44.53
		Random-slot	62.50	1,567	35.11
Qwen3-32B	Hermes/LCB-v6	Native	34.38	96	2.60
		Depth-4	37.50	384	9.85
		Depth-8	39.58	768	19.72
		Funnel-8	40.63	551	16.13
		Random-slot	38.54	551	14.01
Qwen3-32B	NanoBot/RG	Native	58.59	384	16.64
		Depth-4	78.65	1,536	65.09
		Depth-8	82.03	3,072	127.80
		Funnel-8	82.29	1,441	86.62
		Random-slot	70.31	1,441	59.12
gpt-oss-20b	Hermes/RG	Native	65.36	384	1.75
		Depth-4	87.50	1,536	7.66
		Depth-8	91.15	3,072	15.90
		Funnel-8	91.41	1,273	8.23
		Random-slot	74.48	1,273	5.98
gpt-oss-20b	Hermes/LCB-v6	Native	57.29	96	0.66
		Depth-4	63.54	384	2.53
		Depth-8	70.83	768	5.55
		Funnel-8	63.54	327	4.12
		Random-slot	67.71	551	4.11
gpt-oss-20b	NanoBot/LCB-v6	Native	51.04	96	0.57
		Depth-4	60.42	384	2.35
		Depth-8	64.58	768	4.77
		Funnel-8	62.50	327	3.39
		Random-slot	61.46	551	3.55
gpt-oss-20b	NanoBot/RG	Native	65.89	384	1.64
		Depth-4	82.81	1,536	6.76
		Depth-8	86.20	3,072	13.59
		Funnel-8	86.20	1,280	8.01
		Random-slot	76.56	1,567	6.76

vector. Candidate sets and IDs are frozen before independent verification. These adaptations are not end-to-end reproductions of the published baselines; a sequential-retry control is not included. Harness-Bench Funnel has a different prompt/tool adaptation and is reported descriptively.

Outcomes and cost accounting. We report final selected-answer pass rates for Reasoning Gym and LiveCodeBench-v6, and mean outcome score for Harness-Bench; these metrics are not pooled. Candidate counts and solving tokens describe generated trajectories. The Harness-Bench Funnel solve-token figure sums recorded input and output usage over 242 native sessions; some HTTP attempts lack usage records. Reported GPU-hours include allocated auxiliary time; their aggregation is not fully specified, so we do not equate them with active compute, FLOPs, energy, or end-to-end latency. We compare costs only within the same model–harness–task configuration.

5 RESULTS AND ANALYSIS

5.1 FIXED-BUDGET COMPARISON

All four primary configurations show the same cost direction: FUNNEL uses fewer candidates and lower reported GPU-hours than Depth-8, while final accuracy differs by only -0.52, $+1.04$, $+0.26$, and $+0.26$ percentage points. On Qwen/HERMES/RG, Funnel-8 achieves 76.56% versus 77.08% for Depth-8, reducing candidates from 3,072 to 1,623 and reported GPU-hours from 67.48 to 44.53. The task-, model-, and harness-change configurations have slightly higher observed accuracy than Depth-8. These observed differences do not establish superiority or statistical equivalence.

Candidate savings are consistently larger than reported GPU-hour savings. Qwen/NANOBOT/RG saves 53.1% of candidates but only 32.2% of reported GPU-hours; the corresponding values for Qwen/HERMES/RG are 47.2% and 34.0%. Complete trajectories are therefore not interchangeable cost units. Longer rescue trajectories, harness execution differences, or accounting conventions could contribute, but the aggregate data do not distinguish these explanations. The supported conclusion is to report both cost axes, not to claim a measured mechanism for their discrepancy.

Depth-4 is a more informative low-budget reference than Native alone. Relative to Depth-4, Funnel-8 gains 6.25 points on Qwen/HERMES/RG while using 30.3% more reported GPU-hours, and gains 3.65 points on Qwen/NANOBOT/RG while using 33.1% more. FUNNEL does not dominate fixed search at every budget; it offers an intermediate point near Depth-8 performance.

The LCB evidence is more modest: Funnel-8 reaches 40.63% versus 37.50% for Depth-4 but uses 63.8% more reported GPU-hours. The 96-task subset makes small differences uncertain.

5.2 RANDOM ALLOCATION ABLATION

Table 1 also reports the random-slot control. Its outcomes describe how attempts are allocated, but differences in total cost limit what can be inferred from this comparison.

5.3 ADAPTIVE BASELINES

Table 2 reports adaptive-budget costs; Figure 2 places selected outcomes against candidate and reported GPU-hour budgets across all three task panels.

Table 2: Adaptive-budget results for Qwen3-32B/HERMES, alongside Funnel. RG and LCB-v6 report final selected-answer accuracy; Harness-Bench reports mean outcome score. Solving tokens are in millions (M). GPU-hours are reported allocated hours, not active-device time.

Benchmark	Method	Score	Candidates per task	Solving tokens (M)	GPU-hours
RG	ASC-8	0.7526	7.35	52.24	70.98
	DSC-8	0.7682	7.87	55.61	415.08
	UAB-4	0.6953	4.00	28.05	39.21
	Funnel-8	0.7656	4.23	32.24	44.53
LCB-v6	ASC-8	0.3958	6.94	12.48	24.54
	DSC-8	0.3958	7.12	12.69	103.55
	UAB-4	0.3750	4.00	7.12	15.21
	Funnel-8	0.4062	5.74	10.41	16.13
Harness-Bench	ASC-8	0.7171	7.75	28.81	22.74
	DSC-8	0.7368	7.88	28.73	138.07
	UAB-4	0.7070	4.00	14.87	15.76
	Funnel-8	0.7515	7.56	29.35	51.93

On RG, Funnel is 1.30 points above ASC and 0.26 points below DSC. It uses 42.5% fewer candidates and 37.3% fewer reported GPU-hours than ASC, and 46.3% fewer candidates and 89.3% fewer reported GPU-hours than DSC. On the LCB-v6 task subset, Funnel is 1.04 points above both methods while using 17.3–19.4% fewer candidates and 34.3–84.4% fewer reported GPU-hours. These observed differences do not establish superiority or statistical equivalence.

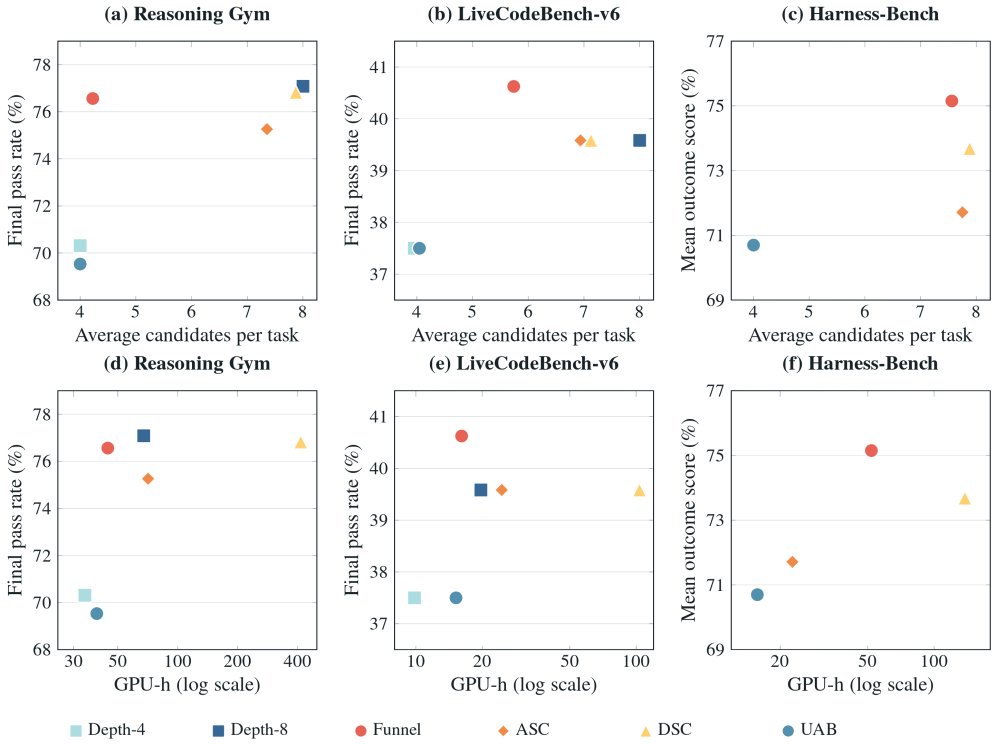


Figure 2: Final selected outcome versus average candidates per task (top) and reported GPU-hours (bottom, logarithmic horizontal axis) for Qwen3-32B/HERMES. RG and LCB-v6 use final pass rate; Harness-Bench uses mean outcome score, displayed as a percentage on its own axis and not pooled with accuracy. Coincident LCB-v6 Depth-4 and UAB points in the top row are slightly offset horizontally for visibility. The Harness-Bench Funnel point uses a different prompt/tool adaptation from the adaptive-baseline points and is descriptive only, not a direct method comparison. GPU-hours are reported allocated hours, not active-device time.

UAB fixes the aggregate candidate budget at four per task on average. Relative to UAB, Funnel uses 5.7% more candidates and 13.6% more reported GPU-hours on RG for a 7.03-point gain; on LCB-v6 it uses 43.5% more candidates and 6.1% more reported GPU-hours for a 3.12-point gain. Thus, no allocator dominates at every budget: ASC and DSC spend more here, while UAB occupies a lower-budget, lower-accuracy point. The Harness-Bench panel is descriptive for Funnel because its prompt and tool adaptation differs from that used in the adaptive-baseline runs.

5.4 CROSS-MODEL AND CROSS-HARNESS BEHAVIOR

On gpt-oss/HERMES/RG, Funnel-8 gains 3.91 points over Depth-4 with fewer candidates, while reported GPU-hours rise from 7.66 to 8.23. This is a favorable observed operating point, not evidence of a general model effect. On Qwen/RG, Funnel-8 remains near Depth-8 on both HERMES and NANOBOT, but its absolute accuracy and reported GPU-hours differ across harnesses: 76.56% and 44.53 versus 82.29% and 86.62. We therefore compare budgets within harnesses.

6 DISCUSSION AND CONCLUSION

FUNNEL concentrates complete-trajectory budgets on tasks that still need another attempt according to their first public check, without changing reasoning or tool use within any native trajectory. Proposition 1 explains the mechanism under a fixed candidate pool and selector: once the earliest candidate passes publicly, later candidates cannot change the selected identifier. This is a selection property, not a claim that separately executed runs yield identical candidates or that the scheduler improves what an individual trajectory can do. The scheduler operates outside the native harness.

The method depends on permitted, informative public checks. A permissive check can stop on an incorrect answer, while a restrictive one can waste rescue attempts; initial failure also does not tell us whether a task is recoverable. We have not measured checker calibration or resistance to exploiting public tests (Liu et al., 2023; Zhong et al., 2026). The primary fixed-budget analysis covers four configurations, not a full factorial design, and each cell has one run; the adaptive comparisons fix one model and harness. Reported costs cannot establish compute or latency; sequential retry is untested.

Across the four primary configurations, Funnel reaches near Depth-8 selected-answer performance with fewer candidates and lower reported GPU-hours. The fixed-four and random-slot controls provide context for budget size and allocation. Under the common selector, adaptive-baseline results place Funnel near ASC and DSC at lower measured cost, while UAB occupies a smaller-budget, lower-accuracy point. Together, these observations support public-check-guided selective additions as a competitive performance–cost option for checkable tasks while preserving native harness execution.

AI Use Statement

Generative AI tools assisted with literature discovery and reference formatting; discussion of research methodology, mathematical exposition, and experimental design; code editing and inspection; organization and interpretation of results; figure and table preparation; and manuscript drafting and language editing. The authors directed the research, reviewed AI-assisted material included in this paper, and take responsibility for the final text, claims, citations, code, and research artifacts.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with llms. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12375–12396, 2023.

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.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397*, 2022.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

HKUDS. nanobot. GitHub repository, 2026. URL `https://github.com/HKUDS/nanobot`. Accessed: 2026-09-19.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

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*, volume 2024, pp. 54107–54157, 2024.

Yusuf Kalayci, Vinod Raman, and Shaddin Dughmi. Optimal stopping vs best-of-n for inference time optimization. *arXiv preprint arXiv:2510.01394*, 2025.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. Ai agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Dacheng Li, Shiyi Cao, Chengkun Cao, Xiuyu Li, Shangyin Tan, Kurt Keutzer, Jiarong Xing, Joseph E Gonzalez, and Ion Stoica. S*: Test time scaling for code generation. In *EMNLP (Findings)*, pp. 15964–15978, 2025.

Yiwei Li, Peiwen Yuan, Shaoxiong Feng, Boyuan Pan, Xinglin Wang, Bin Sun, Heda Wang, and Kan Li. Escape sky-high cost: Early-stopping self-consistency for multi-step reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 14751–14768, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Manh Nguyen, Sunil Gupta, and Hung Le. Uncertainty-aware budget allocation for adaptive test-time reasoning, 2026. URL `https://arxiv.org/abs/2605.26849`.

Nous Research. Hermes Agent. GitHub repository, 2026. URL `https://github.com/NousResearch/hermes-agent`. Accessed: 2026-09-21.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, Mohammed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *International Conference on Learning Representations*, volume 2025, pp. 34433–34448, 2025.

Vinod Raman, Hilal Asi, and Satyen Kale. Adabon: Adaptive best-of-n alignment. *arXiv preprint arXiv:2505.12050*, 2025.

Tal Ridnik, Dedy Kredo, and Itamar Friedman. Code generation with alphacodium: From prompt engineering to flow engineering. *arXiv preprint arXiv:2401.08500*, 2024.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

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*, volume 2025, pp. 10131–10165, 2025.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. *Advances in Neural Information Processing Systems*, 38, 2026.

Xinglin Wang, Shaoxiong Feng, Yiwei Li, Peiwen Yuan, Yueqi Zhang, Chuyi Tan, Boyuan Pan, Yao Hu, and Kan Li. Make every penny count: Difficulty-adaptive self-consistency for cost-efficient reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 6919–6932, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

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 Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yilun Yao, Xinyu Tan, Chao-Hsuan Liu, Yaoming Li, Zhengyang Wang, Wenhan Yu, Zhewen Tan, Yuxuan Tian, Guangxiang Zhao, Lin Sun, et al. Harness-bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

Ziqian Zhong, Aditi Raghunathan, and Nicholas Carlini. Impossiblebench: Measuring llms' propensity of exploiting test cases. In *International Conference on Learning Representations*, volume 2026, pp. 124959–124986, 2026.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Bowen Zuo and Yinglun Zhu. Strategic scaling of test-time compute: A bandit learning approach. In *International Conference on Learning Representations*, volume 2026, pp. 134105–134128, 2026.