BENCHMARKING LLM AGENTS ON PUSHING THE TEST-TIME REWARD–COST FRONTIER

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents can now autonomously form hypotheses, modify code, run experiments, and interpret results. This workflow is commonly known as autoresearch. A prominent goal of this paradigm is to use LLM agents to improve LLMs themselves. Existing autoresearch testbeds typically pursue this goal through model training. This makes each research iteration slow and expensive, leaving agents with limited opportunities to develop methods that are practical and transferable to larger-scale settings within a fixed time budget. Moreover, because agents are often given broad control over data, training pipelines, and engineering choices, final performance may depend as much on auxiliary capabilities, including finding benchmark-relevant training data, as on the ability to conduct research. We instead introduce a controlled benchmark for autoresearch at test time. It asks an agent to push the reward–cost frontier of a frozen LLM on agentic tasks solely by reorganising its inference-time computation: achieving a higher reward at the same cost or the same reward at a lower cost. This setting enables faster research iterations, and any method discovered can be applied directly to already-trained models without additional training. We build a controlled evaluation on τ^2-bench, Terminal-Bench 2.1, and SWE-bench Verified, using a small development set for autoresearch and a separate held-out set for evaluation. During autoresearch, coding agents may change only how the given LLM is used at inference time. We find that (1) frontier agents cannot yet push the reward–cost frontier of their own backbone models, even when they appear to do so on the development set; (2) different agents converge to similar methods; and (3) agents retain the broad research direction they choose at the outset, even when given access to extensive relevant literature.

1 INTRODUCTION

Language-model agents can now autonomously form hypotheses, modify code, run experiments and interpret results. This workflow is commonly known as autoresearch. A prominent goal of this paradigm is to use LLM agents to improve LLMs themselves. Existing testbeds typically pursue this goal through model training: pretraining a small language model in five-minute runs (Karpathy, 2026), post-training a base model in ten GPU-hours (Rank et al., 2026; Li et al., 2026), or closing a reinforcement-learning loop (Chen et al., 2026). This setup presents two challenges. First, because each experiment requires training a model, research iterations are slow and expensive, leaving an agent with few opportunities to develop methods that are practical and transferable to larger-scale settings within a fixed budget. Second, agents are often given broad control over data, training pipelines and engineering choices, so final performance may depend as much on auxiliary capabilities, including finding benchmark-relevant training data, as on the ability to conduct research itself.

Training, however, is not the only way to improve an LLM. On a multi-turn agentic task, the same frozen model can achieve substantially different reward at substantially different cost depending on how its test-time computation is organised: how many candidates are sampled at each turn, how one is selected, whether an action is checked before execution and how much context each turn rereads. This setting is particularly well suited to autoresearch. Each experiment consists of running episodes rather than training a model, enabling faster iteration, and any method discovered can be applied directly to an already-trained model without additional training. We therefore ask: *can an*

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.

autoresearch agent push the reward–cost frontier of a given LLM on agentic tasks at test time? We focus on the frontier rather than scaling alone. Test-time scaling asks how reward grows as more computation is spent (Wang et al., 2022; Brown et al., 2024; Snell et al., 2024); on agentic tasks, where an episode may already require tens of model calls, reaching the same reward with fewer tokens can be as valuable as reaching higher reward with more. Prior automated design of test-time methods (Zheng et al., 2026b) operates offline on single-turn mathematics, while agent-design search (Hu et al., 2024; Zhang et al., 2025) optimises reward alone. Neither directly evaluates whether an agent can improve the test-time reward–cost frontier on multi-turn agentic tasks.

We introduce a controlled evaluation on three established benchmarks: τ^2-bench (Barres et al., 2025), Terminal-Bench 2.1 (Merrill et al., 2026) and SWE-bench Verified (Jimenez et al., 2024). Each autoresearch agent receives 12 hours and a 50-task development set, and may change only the inference policy that produces each assistant turn of a frozen target model. The model weights, harness, benchmark-provided prompts, environments and evaluators remain fixed, allowing improvements to be attributed specifically to how the target model's test-time computation is organised. Reward is measured by each benchmark's official metric, cost by the number of output tokens, and the reference frontier by single-sample decoding and best-of-n. We use three frontier models as autoresearch agents: Opus 5 in Claude Code, and GPT-5.5 and GPT-5.6-sol in Codex. Each agent optimises its own backbone. We evaluate every delivered method on a separate set of 817 held-out tasks.

Our evaluation yields three findings. First, **frontier agents cannot yet push the reward–cost frontier of their own backbone models**. Apparent gains of up to 10 points on the development set shrink to at most 3 points on held-out tasks and require up to 81% more output tokens, indicating that the development gains largely reflect noisy selection and task-specific fitting rather than general improvements (Section 3.1). Second, **different agents converge to similar methods**: all three centre their search on risk-gated sampling, which draws one sample by default and additional samples for risky actions, while none retains a model-based judge (Section 3.2). Third, **agents retain the broad research direction they choose at the outset**, spending most of the remaining budget tuning that direction rather than exploring fundamentally different method families (Section 3.3).

We further examine whether these outcomes are explained by the target model, the research budget or access to prior work (Section 4). On the open-weight Qwen3-Coder-Next, which leaves more room for improvement, both Opus 5 and GPT-5.5 improve the held-out frontier. Their methods reduce output tokens by 19% and 10%, respectively, while matching or slightly improving reward. Doubling the GPT-5.5 session from 12 to 24 hours increases its reported development gain from 7.1 to 17.6 points, but its held-out reward remains 0.7 points below single-sample decoding while using 1.56 times as many tokens, as the additional time is spent fitting individual development tasks. Providing 200 recent papers on test-time scaling likewise does not change the agent's tendency to commit early to one broad direction. Together, these results suggest that current autoresearch agents can rapidly implement and tune plausible test-time policies, and can sometimes improve efficiency when the target leaves clear headroom, but still struggle to identify and validate improvements that generalise beyond the tasks used during research. Advancing autoresearch therefore requires not only faster experimentation or more prior knowledge, but better exploration of research directions and more reliable reasoning under noisy empirical feedback.

2 Benchmark Design

Figure 1 summarises the evaluation. Each autoresearch agent receives a frozen target model, a 50-task development set, implementations of single-sample decoding and best-of-n, and a 12-hour research budget. The agent develops an inference policy, which we evaluate on 817 held-out tasks.

2.1 Task Formulation

An inference policy produces each assistant turn in an agentic episode. It may call the target model multiple times and combine its outputs, but it may not change the model's weights, call another model or modify the benchmark harness. Single-sample decoding, which makes one model call per turn, is the simplest policy.

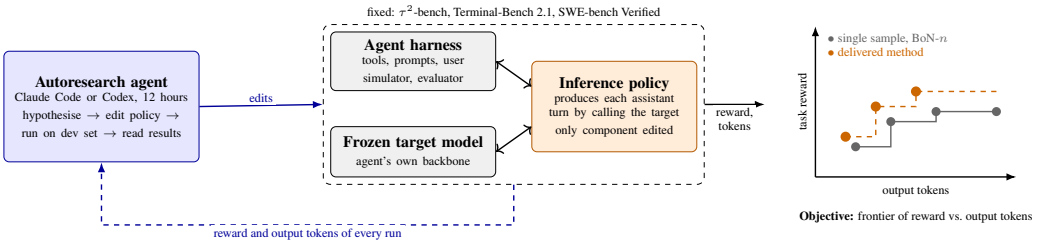


Figure 1: Overview of the evaluation. The grey components remain fixed, while the autoresearch agent edits the orange inference policy using feedback from the development set. The objective is to improve the reward–cost frontier defined by single-sample decoding and best-of-n. The delivered method is evaluated on the held-out set.

Table 1: Benchmarks, harnesses and data splits.

Benchmark	Harness	Dev	Held out
τ^2-bench (Barres et al., 2025)	–	30	248
Terminal-Bench 2.1 (Merrill et al., 2026)	Terminus-2	10	79
SWE-bench Verified (OpenAI, 2024)	mini-swe-agent	10	490

We characterise a policy by its reward and cost. Reward is the benchmark's official metric, and cost is the number of output tokens generated by all calls to the target model. Both are averaged over tasks and then macro-averaged across the three benchmarks; we also report cost relative to single-sample decoding. The baselines define a reference frontier in reward–cost space. A policy improves this frontier if it achieves higher reward at no greater cost or the same reward at lower cost.

2.2 Environment

Agents and targets. We use three frontier models as autoresearch agents, each in its provider's native scaffold: Opus 5 in Claude Code, and GPT-5.5 and GPT-5.6-sol in Codex. Each agent optimises its own backbone model. Agents use the highest reasoning effort available in their scaffold, while target models use high reasoning effort (Table 2). Further details are in Appendix B.

Autoresearch environment. Each session starts in a fresh workspace containing the task statement (Appendix A), baseline implementations and a command that evaluates a policy on the development set and returns per-task reward, output tokens and trajectories. The agent can edit files, run shell commands and search the workspace, but has no internet access or human assistance. It iteratively proposes a method, implements it and evaluates it, then delivers a final method and the compute levels at which it should run. Only the inference policy is editable; model weights, tools, benchmark-provided prompts, environments, user simulator and evaluators remain fixed.

Benchmarks. We use three agentic benchmarks with their official harnesses and evaluators (Table 1). In τ^2-bench (Yao et al., 2024; Barres et al., 2025), the model uses domain tools to resolve simulated user requests while following written policies for airline, retail and telecom tasks. Terminal-Bench 2.1 (Merrill et al., 2026) evaluates tasks completed in a sandboxed Linux shell with task-specific tests. SWE-bench Verified (Jimenez et al., 2024; OpenAI, 2024) evaluates patches for real GitHub issues using repository tests.

Splits and baselines. We sample 30 τ^2-bench tasks, 10 Terminal-Bench tasks and 10 SWE-bench tasks for development. The remaining 248, 79 and 490 tasks, respectively, form the held-out set and are never shown to the agent. The development baselines are single-sample decoding and best-of-n for $n \in \{2, 4, 8\}$. Best-of-n samples n candidate messages at each turn and asks the target model to select one. We evaluate these baselines for every target and provide their implementations to the agent as starting points.

Table 2: Agent–target pairings in the main experiment.

Agent	Scaffold	Agent effort	Target	Target effort
Opus 5	Claude Code	Max	Opus 5	High
GPT-5.5	Codex	Extra high	GPT-5.5	High
GPT-5.6-sol	Codex	Extra high	GPT-5.6-sol	High

Table 3: Development reward and output tokens per task, macro-averaged over the three benchmarks. Grey subscripts report output tokens relative to single-sample decoding of the same target. Held-out results are in Table 4.

Method	Opus 5		GPT-5.5		GPT-5.6-sol	
	Reward (%)	Output tokens	Reward (%)	Output tokens	Reward (%)	Output tokens
Single sample	81.6	8,860 $_{\times 1.00}$	80.0	12,016 $_{\times 1.00}$	80.0	4,747 $_{\times 1.00}$
Best-of-n, n=2	86.2	14,500 $_{\times 1.64}$	80.2	37,242 $_{\times 3.10}$	85.3	7,456 $_{\times 1.57}$
Best-of-n, n=4	92.7	31,276 $_{\times 3.53}$	79.6	46,085 $_{\times 3.84}$	88.0	17,684 $_{\times 3.73}$
Best-of-n, n=8	86.7	55,324 $_{\times 6.24}$	83.1	85,843 $_{\times 7.14}$	88.2	32,945 $_{\times 6.94}$
Autoresearch frontier	85.7	8,544 $_{\times 0.96}$	87.1	8,820 $_{\times 0.73}$	90.4	4,856 $_{\times 1.02}$

Evaluation protocol. We conduct two independent autoresearch sessions for each agent–target pairing and report their mean. Each delivered method is then evaluated on the held-out set. Table 2 lists the pairings in the main experiment; Appendix B gives the trial counts and token-accounting details.

3 Can Autoresearch Agents Push the Frontier?

Table 3 gives the reward and output tokens per task of every baseline and of every delivered method on the development set, and the baselines fix how much room each target leaves for an agent.

3.1 Finding 1: Frontier agents cannot yet push their own frontier

All three agents appear to improve the frontier on the development set (Figure 2 and Table 3). Relative to single-sample decoding, the methods delivered by Opus 5 and GPT-5.5 gain 4.1 and 7.1 reward points while using 4% and 27% fewer output tokens. The method from GPT-5.6-sol gains 10.4 points at 2% greater cost and also exceeds best-of-4 by 2.4 points while using only 27% as many tokens. These results, however, are measured on the same 50 tasks used to develop and select each method.

Two features of the research process make these estimates optimistic. First, repeated evaluations of the same policy are noisy. For GPT-5.5, eight single-sample runs range from 88.7% to 94.0% on τ^2-bench and from 60% to 80% on Terminal-Bench; six runs of the Terminal-Bench policy delivered by GPT-5.6-sol range from 70% to 90%. Both GPT agents select their best run, whereas Opus 5 pools repeated runs and estimates its own gain at only 0.6 points relative to a same-day control. Second, the agents repeatedly target the few development tasks that fail. The GPT agents probe these tasks directly and add task-specific instructions, including setup hints for individual Terminal-Bench tasks. Because each delivered method is selected after evaluating 20 to 54 versions on the same small development set, its reported development reward reflects both measurement noise and adaptation to individual tasks.

Held-out evaluation confirms this gap (Table 4). The Opus 5 method reaches 89.5% reward at 14,532 tokens, compared with 90.4% at 12,447 tokens for single-sample decoding, and is therefore dominated. The GPT-5.5 method reduces cost by 5% but loses 2.8 reward points. Only the GPT-5.6-sol method improves reward, by 2.7 points, but it uses 81% more tokens; its development gain was 10.4 points at nearly unchanged cost. Across the three agents, reward gains fall by 5.0–9.9 points from development to held-out evaluation, and seven of the nine per-benchmark gains are no greater

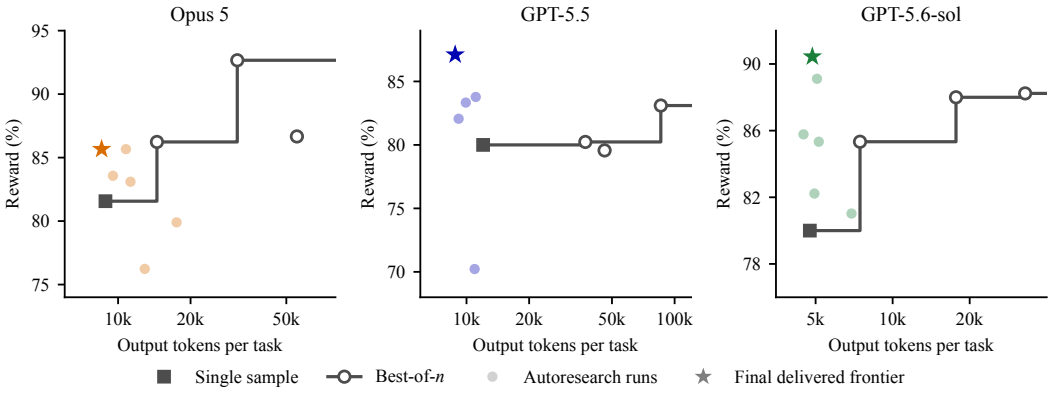


Figure 2: Development frontiers for agents optimising their own backbones. Grey squares show single-sample decoding, and grey circles show best-of-n for $n = 2, 4, 8$; the staircase traces their frontier. Coloured circles show evaluated settings, averaging repeated runs, and stars show the delivered methods. Reward and output tokens per task are unweighted means across the three benchmarks.

Table 4: Held-out reward and output tokens per task, macro-averaged over the three benchmarks. Grey subscripts report output tokens relative to single-sample decoding of the same target.

Method	Opus 5		GPT-5.5		GPT-5.6-sol	
	Reward (%)	Output tokens	Reward (%)	Output tokens	Reward (%)	Output tokens
Single sample	90.4	12,447 $_{\times 1.00}$	83.0	11,045 $_{\times 1.00}$	78.6	5,759 $_{\times 1.00}$
Autoresearch frontier	89.5	14,532 $_{\times 1.17}$	80.2	10,501 $_{\times 0.95}$	81.3	10,409 $_{\times 1.81}$

than one point (Figure 3a; Table 10). Relative cost also rises from 0.96, 0.73 and 1.02 times single-sample decoding on the development set to 1.17, 0.95 and 1.81 times on the held-out set (Figure 3b). Thus, none of the delivered methods demonstrates a robust improvement in the reward–cost frontier of its own backbone.

3.2 Finding 2: Agents converge to similar methods

Working independently, all three agents centre their search on risk-gated sampling. This policy draws one sample by default and requests additional samples only when the proposed action is judged risky, such as an action that changes the environment or ends the episode. The Opus 5 and GPT-5.5 methods retain this mechanism in their delivered configurations. Opus 5 selects the action with the strongest agreement across samples, while GPT-5.5 accepts an action only when samples match exactly. GPT-5.6-sol explores the same mechanism but disables additional sampling in its delivered configuration, leaving single-sample decoding with benchmark-specific checklists. The convergence is therefore strongest in the agents' research direction, although their final configurations differ (Table 5). This design applies adaptive sampling and consistency-based aggregation to individual turns of a multi-turn episode (Wang et al., 2022; Aggarwal et al., 2023).

The agents also converge in rejecting a model-based judge. Opus 5 estimates that each judge call would add about 500 output tokens and does not implement one. GPT-5.5 evaluates a model-based selector but finds it consistently unhelpful, while GPT-5.6-sol tests both a selector and a critic but retains neither. The resulting methods use simple agreement rules or instruction changes rather than additional calls for model-based evaluation.

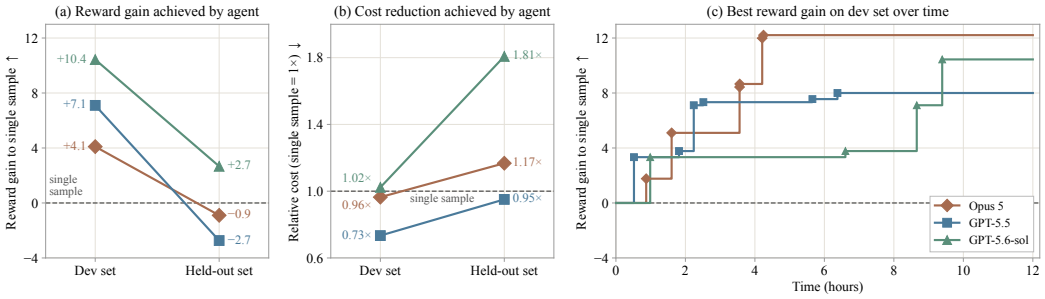


Figure 3: Development gains do not carry over to held-out tasks. Panel a shows reward gains over single-sample decoding on the development and held-out sets. Panel b shows relative output-token cost. Panel c tracks the best development gain found over active research time; markers indicate runs that improve it.

Table 5: Methods developed for each agent's own backbone.

Agent	Category	Method
Opus 5	Risk-gated sampling	Consequence-gated consensus. Samples once by default, draws additional samples for actions that are hard to undo and selects the action with the strongest agreement.
GPT-5.5	Risk-gated sampling	Risk-gated exact voting. Samples once by default, draws additional samples for risky actions and accepts an action only when samples match exactly.
GPT-5.6-sol	Instruction changes	Risk-gated lossless deliberation. The delivered configuration disables additional sampling and uses single-sample decoding with benchmark-specific checklists.

3.3 Finding 3: Agents keep the direction they choose at the start

Each agent commits to risk-gated sampling early in its session. Opus 5 adopts consequence-gated consensus after 20 minutes (Appendix C), and GPT-5.5 proposes risk-gated exact voting after 3 minutes. GPT-5.6-sol begins developing risk-gated deliberation after about 80 minutes, once its target calls return complete tool actions. The remaining experiments tune the same components: which actions trigger the gate, how many additional samples to draw and how to aggregate disagreements.

When risk-gated sampling does not improve a benchmark, the GPT agents fall back to single-sample decoding with local changes, including output limits, task-specific hints and benchmark-specific checklists. They retain the gating mechanism in their code rather than explore a different method family. No agent investigates trajectory-level search or context management. More research time may therefore deepen the search around an initial direction without broadening it, a hypothesis we test in Section 4.2.

4 What Shapes Autoresearch Performance?

The findings above admit three natural explanations for the size of the gain an agent finds: the target model it optimises, the time it has, and what it knows about test-time methods. We examine each by changing one factor at a time: we let Opus 5 and GPT-5.5 optimise an open-weight model with more headroom instead of their own backbone, we rerun the GPT-5.5 own-backbone pairing with a 24-hour budget, and we rerun the Opus 5 own-backbone pairing with 200 papers on test-time scaling in its workspace.

Table 6: Held-out performance on Qwen3-Coder-Next, macro-averaged across the three benchmarks. Rewards are percentages, and grey subscripts report output-token cost relative to single-sample decoding.

Method	Agent	Reward	Output tokens
Single sample	–	58.4	21,273 $_{\times 1.00}$
Autoresearch frontier	Opus 5	58.4	17,255 $_{\times 0.81}$
Autoresearch frontier	GPT-5.5	59.2	19,175 $_{\times 0.90}$

Table 7: Held-out performance of GPT-5.5 methods developed with 12- and 24-hour research budgets. Rewards are percentages, and grey subscripts report token cost relative to single-sample decoding.

Method	Budget	τ^2-bench		Terminal-Bench 2.1		SWE-bench Verified	
		Reward	Tokens	Reward	Tokens	Reward	Tokens
Single sample	–	90.0	2,485 $_{\times 1.00}$	71.5	17,071 $_{\times 1.00}$	87.4	13,578 $_{\times 1.00}$
Autoresearch frontier	12 h	85.6	2,575 $_{\times 1.04}$	72.2	14,030 $_{\times 0.82}$	82.9	14,898 $_{\times 1.10}$
Autoresearch frontier	24 h	85.3	3,718 $_{\times 1.50}$	79.1	22,428 $_{\times 1.31}$	82.6	25,573 $_{\times 1.88}$

4.1 Can agents push the frontier of other models?

We test whether the target model limits autoresearch by asking Opus 5 and GPT-5.5 to optimise the open-weight Qwen3-Coder-Next (Cao et al., 2026) under the same protocol. Both agents improve its held-out frontier (Table 6). The Opus 5 method matches the 58.4% reward of single-sample decoding while reducing output tokens by 19%. The GPT-5.5 method reaches 59.2%, a gain of 0.8 points, while reducing output tokens by 10%.

The Opus 5 method improves efficiency primarily by limiting runaway generation. It caps each call at 32,768 output tokens and applies an additional transcript budget on SWE-bench, prompting the model to reason compactly after 8,000 tokens and to finish after 60,000 tokens or when recent actions repeat. It draws extra samples only before irreversible actions. On SWE-bench, this method gains 2.3 points while using 28% fewer tokens; on the other two benchmarks, it reduces tokens by 11% with small reward losses (Table 10).

The GPT-5.5 method improves reward by 1.7 points on τ^2-bench and 0.8 on SWE-bench, while matching single-sample decoding on Terminal-Bench. It reduces Terminal-Bench tokens by 25%, leaves SWE-bench cost nearly unchanged and spends 1.8 times as many tokens on τ^2-bench. These results suggest that target-model headroom matters. Unlike the methods developed for the agents' own backbones, both methods developed for Qwen3-Coder-Next improve its held-out frontier.

4.2 Is research time the bottleneck?

To test whether the 12-hour budget limits research, we rerun the GPT-5.5 self-pairing for 24 hours with all other conditions fixed. The longer session substantially improves the selected development result: its gain over single-sample decoding rises from 7.1 to 17.6 points. This improvement does not transfer to the held-out set. The 24-hour method reaches 82.3% reward at 17,240 output tokens per task, compared with 83.0% at 11,045 tokens for single-sample decoding. It recovers 2.1 points over the 12-hour method, but remains 0.7 points below the baseline while using 56% more tokens.

The effect also varies across benchmarks (Table 7). Relative to single-sample decoding, the 24-hour method gains 7.6 points on Terminal-Bench but loses 4.7 points on τ^2-bench and 4.8 points on SWE-bench. Its token cost increases by 31%, 50% and 88%, respectively. More research time therefore improves one benchmark without producing a consistent reward–cost gain.

The session reaches its best development reward within five hours and never surpasses it. It spends much of the remaining time reducing the cost of that development point and adding prompt-triggered hints for individual tasks. Its method family also remains unchanged: the session begins with risk-

Table 8: Held-out performance of Opus 5 autoresearch frontiers developed with and without access to 200 test-time scaling papers. Rewards are percentages, and grey subscripts report token cost relative to single-sample decoding.

Method	Papers	τ^2-bench		Terminal-Bench 2.1		SWE-bench Verified	
		Reward	Tokens	Reward	Tokens	Reward	Tokens
Single sample	–	94.1	2,289 $_{\times 1.00}$	82.3	22,111 $_{\times 1.00}$	94.7	12,942 $_{\times 1.00}$
Autoresearch frontier	0	92.7	2,607 $_{\times 1.14}$	81.0	27,568 $_{\times 1.25}$	94.7	13,421 $_{\times 1.04}$
Autoresearch frontier	200	93.2	4,214 $_{\times 1.84}$	77.2	33,894 $_{\times 1.53}$	94.2	18,742 $_{\times 1.45}$

gated sampling and ends with the same mechanism, enabling additional samples only where they help on the development set. The extra time thus strengthens local optimisation and task-specific fitting rather than broadening the search or improving held-out generalisation.

4.3 Is knowledge of the literature missing?

To test whether limited knowledge of prior work constrains autoresearch, we rerun the Opus 5 self-pairing with 200 recent test-time scaling papers from NeurIPS, ICLR and ICML. All other conditions remain fixed. Access to the papers does not improve the held-out frontier (Table 8). The resulting method reaches 88.2% reward at 18,950 output tokens per task, compared with 89.5% at 14,532 tokens without the papers and 90.4% at 12,447 tokens for single-sample decoding. It therefore loses 1.3 points and uses 30% more tokens than the method developed without literature.

The result is consistent across benchmarks. Relative to single-sample decoding, the literature-assisted method loses 0.9 points on τ^2-bench, 5.1 on Terminal-Bench and 0.5 on SWE-bench, while increasing token cost by 84%, 53% and 45%. The agent also remains within the risk-gated sampling family chosen early in the session. Access to relevant papers therefore neither improves generalisation nor broadens the agent's research direction, suggesting that the bottleneck is not simply a lack of exposure to existing methods.

5 Related Work

Autoresearch and automated discovery. Existing autoresearch testbeds commonly ask agents to improve models through pretraining (Karpathy, 2026), post-training or fine-tuning (Rank et al., 2026; Li et al., 2026), and reinforcement learning (Chen et al., 2026). MLE-bench (Chan et al., 2024), MLAgentBench (Huang et al., 2024), RE-Bench (Wijk et al., 2024) and AutoResearchBench (Xiong et al., 2026) evaluate broader machine-learning engineering and research capabilities, while PaperBench focuses on paper replication (Starace et al., 2025). These settings expose several parts of the research pipeline, so outcomes may reflect data selection and engineering as well as method design. Training also makes each iteration expensive. Our benchmark instead fixes the model, data and evaluation pipeline and exposes only the inference policy. This enables faster experimentation and more directly measures whether an agent can develop a generalisable method.

Language-model-guided search has discovered mathematical constructions (Romera-Paredes et al., 2024), heuristics (Liu et al., 2024), algorithms (Novikov et al., 2025), agentic systems (Hu et al., 2024) and model harnesses (Lee et al., 2026), and has also been applied to the research process itself (Lu et al., 2024). Related work on self-improvement searches over prompts, scaffolds or agent code within author-designed outer loops (Zelikman et al., 2023; Yin et al., 2024; Zhang et al., 2025). Our agents instead design the inference policy of a fixed target model. The own-backbone experiments provide a controlled test of whether current agents can improve the models on which they are based.

Test-time scaling. Existing methods scale inference through repeated sampling and aggregation (Wang et al., 2022; Brown et al., 2024), verifier-based selection (Cobbe et al., 2021), adaptive stopping (Aggarwal et al., 2023; Li et al., 2024), pruning (Fu et al., 2025; Zheng et al., 2026a; Tu et al., 2026), budget forcing (Muennighoff et al., 2025), tree search (Yao et al., 2023) and compute-optimal allocation (Snell et al., 2024). Most are designed for single-turn reasoning and measure how addi-

tional computation improves reward. AutoTTS automates the design of such methods, but searches offline over a fixed controller space for single-turn mathematics (Zheng et al., 2026b). We instead evaluate open-ended policy design on live, multi-turn agentic tasks. Our objective treats lower cost at matched reward as equally valuable as higher reward at matched cost, with single-sample decoding and best-of-n defining the reference frontier.

6 Conclusion

We introduce a controlled benchmark for testing whether autoresearch agents can improve the reward–cost frontier of a frozen LLM on multi-turn agentic tasks. Across three own-backbone pairings, apparent development gains do not translate into robust held-out frontier improvements. The agents independently centre their search on risk-gated sampling and remain close to the direction chosen early in each session.

The controlled variations clarify when this limitation changes. Target-model headroom matters: when optimising Qwen3-Coder-Next, both Opus 5 and GPT-5.5 reduce output-token cost while matching or slightly improving reward. More research time produces a larger development gain but not a better held-out frontier. Access to 200 papers on test-time scaling likewise does not improve the held-out frontier or broaden the agent's initial research direction. Current autoresearch agents can rapidly implement and tune plausible inference policies, but they still struggle to distinguish generalisable improvements from noisy measurements and task-specific gains. Progress therefore requires stronger exploration and more reliable empirical reasoning, not only larger research budgets or more background material.

AI use statement

In this work, we used generative AI tools as the autoresearch agents evaluated in our experiments and for language editing of the manuscript. The authors designed the benchmark and experimental protocol, verified the implementation and reported results, and reviewed all AI-assisted text and artifacts.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, and Mausam. Let's sample step by step: Adaptive-consistency for efficient reasoning and coding with LLMs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 12375–12396, 2023.

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

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.

Ruisheng Cao, Mouxiang Chen, Jiawei Chen, Zeyu Cui, Yunlong Feng, Binyuan Hui, Yuheng Jing, Kaixin Li, Mingze Li, Junyang Lin, Zeyao Ma, Kashun Shum, Xuwu Wang, Jinxi Wei, Jiaxi Yang, Jiajun Zhang, Lei Zhang, Zongmeng Zhang, Wenting Zhao, and Fan Zhou. Qwen3-Coder-Next technical report. *arXiv preprint arXiv:2603.00729*, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Wanyi Chen, Xiao Yang, Xu Yang, Tianming Sha, Qizheng Li, Zhuo Wang, Bowen Xian, Fang Kong, Weiqing Liu, and Jiang Bian. Agent^{2}RL-bench: Can LLM agents engineer agentic RL post-training? *arXiv preprint arXiv:2604.10547*, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John

Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. *arXiv preprint arXiv:2508.15260*, 2025.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLagentBench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning (ICML)*, 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 (ICLR)*, 2024.

Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically. `https://github.com/karpathy/autoresearch`, 2026. GitHub repository.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Qizheng Li, Yifei Zhang, Xiao Yang, Xu Yang, Zhuo Wang, Weiqing Liu, and Jiang Bian. FT-dojo: Towards autonomous LLM fine-tuning with language agents. *arXiv preprint arXiv:2603.01712*, 2026.

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. *arXiv preprint arXiv:2401.10480*, 2024.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language models. *arXiv preprint arXiv:2401.02051*, 2024.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *arXiv preprint arXiv:2601.11868*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B. Hashimoto. s1: Simple test-time scaling. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 20275–20321, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. Introducing SWE-bench verified. `https://openai.com/index/introducing-swe-bench-verified/`, 2024.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can LLM agents automate LLM post-training? *arXiv preprint arXiv:2603.08640*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. PaperBench: Evaluating AI's ability to replicate AI research. *arXiv preprint arXiv:2504.01848*, 2025.

Shangqing Tu, Yaxuan Li, Yushi Bai, Lei Hou, and Juanzi Li. Deepprune: Parallel scaling without inter-trace redundancy. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 13389–13403, 2026.

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.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-bench: Evaluating frontier AI r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

Lei Xiong, Kun Luo, Ziyi Xia, Wenbo Zhang, Jin-Ge Yao, Zheng Liu, Jingying Shao, Jianlyu Chen, Hongjin Qian, Xi Yang, Qian Yu, Hao Li, Chen Yue, Xiaan Du, Yuyang Wang, Yesheng Liu, Haiyu Xu, and Zhicheng Dou. Autoresearchbench: Benchmarking AI agents on complex scientific literature discovery. *arXiv preprint arXiv:2604.25256*, 2026.

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.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*, 2024.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*, 2023.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025.

Tong Zheng, Chengsong Huang, Runpeng Dai, Yun He, Rui Liu, Xin Ni, Huiwen Bao, Kaishen Wang, Hongtu Zhu, Jiaxin Huang, Furong Huang, and Heng Huang. Parallel-probe: Towards efficient parallel thinking via 2D probing. *arXiv preprint arXiv:2602.03845*, 2026a.

Tong Zheng, Haolin Liu, Chengsong Huang, Huiwen Bao, Sheng Zhang, Rui Liu, Runpeng Dai, Ruibo Chen, Chenxi Liu, Tianyi Xiong, Xidong Wu, Hongming Zhang, and Heng Huang. LLMs improving LLMs: Agentic discovery for test-time scaling. *arXiv preprint arXiv:2605.08083*, 2026b.

A Instruction to autoresearch agents

All agents receive the instructions below, with only the target-model name changed. We omit setup, launch and troubleshooting instructions. In the research-time condition, only the stated budget differs. In the literature condition, an additional paragraph directs the agent to the provided papers.

RESEARCH TASK

You are given a frozen LLM and 50 multi-turn agentic tasks. Your goal is to develop a training-free test-time scaling method that achieves the best performance–cost Pareto frontier and generalises to unseen tasks. Specifically, you will start by improving the frontier of that LLM on the tasks below.

Benchmark	Tasks	Trials	What it measures
τ^2-bench	30	5	Multi-turn dialogue with tools, scored on database state and communication checks
Terminal-Bench	10	5	Terminal tasks in a container, scored by task-specific tests
SWE-bench	10	5	GitHub issues, scored by repository test suites

Performance is the official task score, and cost is the number of output tokens per task. Both are macro-averaged across the three benchmarks. A run is on the frontier if no other run achieves at least the same performance at no greater cost. Develop one method with a parameter that trades cost for performance, producing multiple operating points along this frontier.

The research budget is 12 hours. Continue iterating until the budget is exhausted.

RESEARCH LOOP

1. Design the policy. The file `research/policy.py` must define `build_policy(**kwargs)`, which returns an inference policy. The policy is called once per agent turn with the frozen target model, the full conversation and the available tool schemas, and must return exactly one assistant message. It may make one or more calls to the target model and may use critique, voting, reranking, branching or other training-free methods. Implementations of single-sample decoding and best-of-n are provided as examples. Commit each policy version before evaluating it so that every result maps to a specific revision.

2. Run the policy. Evaluate the policy on the development set with the provided runner. Small task subsets may be used for smoke tests, but truncated runs are excluded from frontier comparisons because their rewards are not comparable. Monitor each run and stop it early if the method or implementation is clearly broken.

3. Score the run. For each run, inspect reward, output tokens per task and termination status. Every episode is scored. An episode that crashes, times out or produces no response receives zero reward and remains in the denominator. A run with missing scores is marked unreliable and excluded from the frontier.

4. Analyse the results. Each run records its configuration, aggregate reward and cost, per-call token usage and official harness output. Inspect trajectories to identify which turns lose reward, which are expensive and where additional computation could help.

5. Decide, record and repeat. Compare reward and cost per task with the current frontier. Keep a change if it moves toward higher reward or lower cost; otherwise revert it. Record the result, the reasoning behind the decision and the next hypothesis. Then refine the method or propose a new one and repeat the loop.

When the research budget is exhausted, report the best method at each selected cost level and write a final summary naming the resulting frontier points. Do not stop before the budget is exhausted, and do not ask the human for guidance; make decisions from the available evidence.

RULES

You may read the policy, framework, baseline implementations, development split, your own results and the benchmark harness source. You may modify only `research/policy.py`, and all modifications must be training-free. Examples include adaptive sampling, self-critique, voting,

dynamic token budgets, uncertainty-triggered inference, early stopping, log-probability-based decisions, branching and reranking. Design a new method rather than reproducing a known one.

You may not:

- train, fine-tune or modify the target model's weights;
- use any LLM other than the assigned target model;
- modify an answer generated by the target model;
- modify the framework, benchmark harness, system prompts, tool definitions, development split, evaluator or scoring code;
- execute benchmark tools directly from inside the policy or change environment state except through the assistant message returned by the policy;
- carry private memory across turns of the external agent loop;
- read or use any data outside the development split, including held-out tasks, evaluator code or unprovided baseline results; or
- tune decoding hyperparameters such as temperature, top-p, top-k or penalties to increase reward.

B Implementation details

Agents. Opus 5 runs in Claude Code with reasoning effort set to max, and GPT-5.5 and GPT-5.6-sol run in Codex with reasoning effort set to extra high. Each session runs without human interaction in its own workspace and without internet access.

Target models. Opus 5, GPT-5.5 and GPT-5.6-sol are called through their providers' APIs with reasoning effort set to high and the providers' default sampling parameters. Qwen3-Coder-Next is sampled with temperature 1.0, top-p 0.95 and top-k 40, and is served on four nodes of eight A100 GPUs each with a context length of 256K tokens. Every policy of a target, baseline or delivered, uses the same settings; an inference policy may only choose how many calls to make and how to combine them (Section 2.1).

Harnesses. Each harness calls the inference policy wherever it would call the model to produce an assistant turn, and is otherwise unchanged. τ^2-bench runs under its official harness, with GPT-5.6-sol as the user simulator for every target. Terminal-Bench 2.1 runs the Terminus-2 agent under the benchmark's own task time limits. SWE-bench Verified runs mini-swe-agent with at most 250 steps per episode, and the resulting patch is graded by the official evaluation.

Trials and cost accounting. On the development set, every policy runs 5 trials per task for all three benchmarks. On the held-out set, each policy runs 5 trials per τ^2-bench task and 1 trial per Terminal-Bench and SWE-bench task. Every call a policy makes to the target is recorded with the output tokens reported by the provider, including reasoning tokens, and cost is summed per task.

C Example session

We use the session in which Opus 5 optimises its own backbone to illustrate the research process. The agent begins by inspecting the task, baselines and harness interfaces. Six minutes into the session, it probes the target and finds that a model-based judge requires 538 output tokens even when asked to return only a digit. It therefore abandons model-based selection before implementing it and instead tests whether candidate actions can vote directly.

The resulting method, consequence-gated consensus, treats turns differently according to their consequences and cost. It first draws one sample and assigns the proposed action a stake based on whether it is malformed, ends the episode, changes the environment, communicates with the user, follows an error or only reads state. It estimates the price of another sample from the draft's output length and draws additional samples only when the stake exceeds the price-adjusted threshold τ. The samples vote first on the exact action and then on its operation type; terminal actions require a strict majority. The maximum sample count n and threshold τ control the reward–cost trade-off.

Table 9: Key decisions and results from the session in which Opus 5 optimises its own backbone. Times are measured from the start of the session.

Time	Step
0:00	Reads the task statement, the baseline implementations and the harness code.
0:06	Probes the target backend. A judge call returns 538 output tokens, so the agent drops a model-based selector before implementing one.
0:09	Runs an instrumented policy that draws four samples per turn and returns the first, to record action formats and how often samples disagree.
0:20	Writes consequence-gated consensus: one sample per turn, and for actions that are hard to undo, additional samples and an election of the action with the strongest agreement. The first development runs start at 0:23.
0:39	Telemetry shows that the gate escalates the longest turns: on Terminal-Bench, 37% of turns account for 84% of output tokens. Adds a price term so that longer drafts need a higher stake to be escalated.
0:54	First τ^2-bench run of the gated policy: 90.0% at 3,269 tokens.
3:22	A deep setting (τ=0.05, n=8) scores 60% on Terminal-Bench: four of ten trials reach the harness time limit because sequential sampling rounds add wall-clock time.
3:43	The setting τ=0.40, n=2 scores 100% on Terminal-Bench at 13,319 tokens.
4:13	Runs the policy with the gate off as a same-day single-sample control.
4:24–9:29	Replicates settings, up to 300 τ^2-bench, 90 Terminal-Bench and 40 SWE-bench episodes per setting. The 100% Terminal-Bench result pools to 86.7% over 90 episodes.
9:30–11:30	Tests a one-round variant of the deep setting to separate sampling depth from sampling width; it recovers one of the four timed-out Terminal-Bench trials.
11:30	Delivers τ=0.40, n=2: 85.7% at 8,544 tokens pooled over all runs, 0.6 points above the same-day control at 1.5% more tokens.

The first experiments reveal that a stake-only gate targets the most expensive turns. On Terminal-Bench, 37% of turns account for 84% of output tokens, so the agent adds the price term and shifts extra computation toward shorter turns. It also tests a deeper configuration with τ=0.05 and n=8. This configuration scores only 60% on Terminal-Bench because four of ten trials reach the harness time limit. Drawing all additional samples in one round recovers only one trial, showing that both sequential depth and sampling width increase wall-clock cost.

Replication changes the apparent ranking of several methods. The selected τ=0.40, n=2 configuration initially scores 100% on Terminal-Bench, but falls to 86.7% after pooling 90 episodes. The final development result is 85.7% at 8,544 output tokens per task, compared with 81.6% at 8,860 tokens for the published single-sample baseline. A same-day single-sample control reaches 85.1% at 8,419 tokens, reducing the estimated contribution of the method to 0.6 points at 1.5% greater cost. On the held-out set, the method reaches 89.5% at 14,532 tokens, compared with 90.4% at 12,447 tokens for single-sample decoding. The session therefore illustrates the paper's main pattern: the agent quickly develops a coherent mechanism and tunes it extensively, but the apparent development gain shrinks under replication and does not generalise to held-out tasks.

D PER-BENCHMARK RESULTS

Table 10: Per-benchmark held-out reward and output tokens per task.

Target	Method	τ^2-bench		Terminal-Bench 2.1		SWE-bench Verified	
		Reward (%)	Output tokens	Reward (%)	Output tokens	Reward (%)	Output tokens
Opus 5	Single sample	94.1	2,289	82.3	22,111	94.7	12,942
	Autoresearch frontier	92.7	2,607	81.0	27,568	94.7	13,421
GPT-5.5	Single sample	90.0	2,485	71.5	17,071	87.4	13,578
	Autoresearch frontier	85.6	2,575	72.2	14,030	82.9	14,898
GPT-5.6-sol	Single sample	74.9	1,769	74.1	8,979	86.9	6,530
	Autoresearch frontier	75.6	1,424	78.5	22,061	89.8	7,743
Qwen3-Coder-Next	Single sample	79.8	2,018	39.2	32,844	56.1	28,957
by Opus 5	Autoresearch frontier	79.4	1,791	37.3	29,257	58.4	20,716
by GPT-5.5	Autoresearch frontier	81.5	3,653	39.2	24,550	56.9	29,322