How Much Can Language Models Gain from Test-Time Computation?

Anonymous authors
Paper under double-blind review

Abstract

How much can test-time computation improve a language model, and at what cost? Test-time scaling is widely proposed as a substitute for larger models, but existing comparisons mostly evaluate one domain at a time and rarely charge selection to the budget. We introduce SELF-POT, a benchmark and evaluation framework that measures the test-time potential of a model across competition mathematics, competitive programming, and agentic workflows. SELF-POT separates candidate coverage from final accuracy on static tasks, tracks correctness transitions under revision, and measures protocol completion alongside task success in agentic environments. Under a unified budget rule, it compares Direct inference with parallel sampling and self-revision under fixed multiples of the Direct budget, and charges every model call, including selection and critique, in dollars. This supports comparisons of both within-model gains and lower-cost models with additional inference against a stronger model. Across five low-cost reasoning models on 350 sealed tasks, with Claude Opus 5.5 Direct as the reference, the returns depend on the domain, selection rule, and failure handling. Replaying retained programming pools, public-example selection raises correct submissions from 376 to 453 of 500 scheduled cells while saving 12-49% of logical API cost across models; simply retaining an available candidate when judging fails recovers 61 submissions at unchanged cost. On identical mathematics pools, judging with fallback yields 186 correct submissions versus 182 for voting, which saves 12-21% of logical API cost. These controlled replays show how selection and failure handling change the gains realized from the same generated candidates and quantify the marginal value of a model judge.

1 Introduction

Test-time scaling improves a fixed language model by spending more computation at inference rather than on training. Parallel methods draw several candidate answers and keep one, chosen by majority vote (Wang et al., 2023), by running candidate programs on tests (Chen et al., 2023a), or by a model that judges the candidates (Chen et al., 2025b). Sequential methods let the model critique an earlier answer and then revise it (Madaan et al., 2023). When the computation is allocated well, a smaller model can match a much larger one on problems it already solves some of the time (Snell et al., 2025), so test-time scaling looks like a way for a low-cost model to replace a stronger and more expensive one. For a fixed model and budget, a practitioner must choose among a single attempt, parallel sampling with selection, and self-revision. The right choice is unlikely to be the same across tasks. Voting and public-example selection use no additional model tokens, whereas a model judgement can cost as much as another answer. In an app workflow, actions can change the environment, so a flawed attempt cannot simply be discarded and drawn again.

Prior comparisons of test-time scaling methods do not tell a practitioner which option to choose in a given task domain. Several studies compare parallel and sequential methods under a matched budget (Snell et al., 2025; Wu et al., 2025; Wang et al., 2024), but they leave three gaps. First, most evaluate a single domain, usually mathematics (Snell et al., 2025; Ghosal et al., 2025), so a method that helps there may or may not help elsewhere. Second, few charge selection or verification calls to the same budget as generation (Wang et al., 2024; Singhi et al., 2025), and most count the budget in FLOPs, samples, or tokens rather than in money. Third, the comparison that motivates test-time scaling, a low-cost model with more computation against a stronger model answering once, 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.

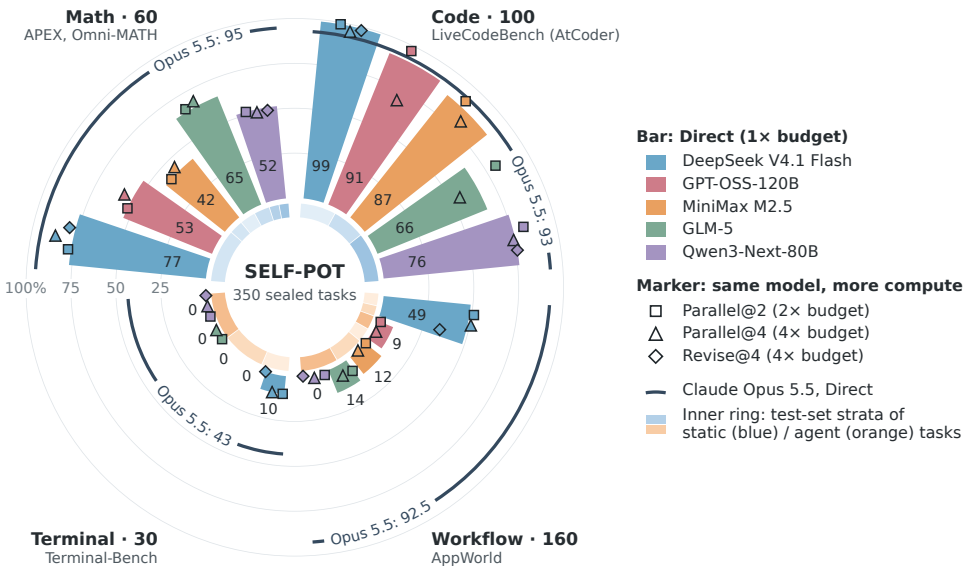


Figure 1: **SELF-POT: frozen protocol matrix.** Each quarter (equal angles) is one domain of the 350-task sealed test set, with its sampling strata in the inner ring. Bars show each low-cost model's Direct accuracy (%), markers the same model under the TTS protocols, and arcs Claude Opus 5.5 answering once. A marker beyond the end of a bar is a gain, and a marker inside a bar is a loss.

been made only in narrow settings. Snell et al. (2025) and Wu et al. (2025) compare a smaller and a larger model from the same family (e.g., Llemma-7B and Llemma-34B in the latter) at matched FLOPs and mainly on mathematics. Stroebl et al. (2026) compare a weaker and a stronger model on programming tasks. Neither setting measures the comparison in dollars or across domains.

We therefore ask three questions. **RQ1:** Under stated budget allowances, which tested parallel and sequential configurations turn extra computation into accuracy, and in which domains? **RQ2:** On static tasks, is opportunity lost in generating a correct candidate, selecting it, or preserving it under revision, and do agent protocols complete their intended execution? **RQ3:** Can a low-cost model with test-time scaling reach the accuracy of a stronger model answering once at no greater cost, and in which cases can it not?

Answering them requires charging every call, including selection and critique, while keeping the model fixed across roles. Candidate count, choice rule, and failure handling must also be distinguished. Domains must differ in feedback and in whether actions can be undone, and the test set must be fixed before any test-time call. SELF-POT meets these requirements (Figure 2). It gives a frozen model a budget of one, two, or four times a single answer and compares answering once (*Direct*) with two parallel protocols and one sequential protocol, each in the simplest form that a reasoning model can run by itself. *Parallel@2* keeps one of two answers by majority vote in mathematics or by the *public examples* given in a programming task, *Parallel@4* lets the same model read three answers and pick one in a selection stage (*joint selection*), and *Revise@4* critiques the answer and then revises it. On agent tasks, the parallel protocols compare plans before a single execution (*plan-level parallelism*), and Revise@4 reviews the trajectory every two steps (*periodic self-review*). We evaluate five low-cost reasoning models on 350 tasks in competition mathematics (APEX and Omni-MATH (Balunovic et al., 2025; MathArena, 2025; Gao et al., 2025)), AtCoder programming from LiveCodeBench (Jain et al., 2025), AppWorld workflows (Trivedi et al., 2024), and Terminal-Bench tasks (Merrill et al., 2026), fixed and hashed before any evaluation run (a *sealed* test set), and compare them with Claude Opus 5.5 answering once, the stronger reference model (the *anchor*). We complement the protocol comparison with controlled selection and fallback analyses on the saved candidate pools, separating generation from conversion. Figure 1 summarizes the frozen protocol matrix. The combined findings are:

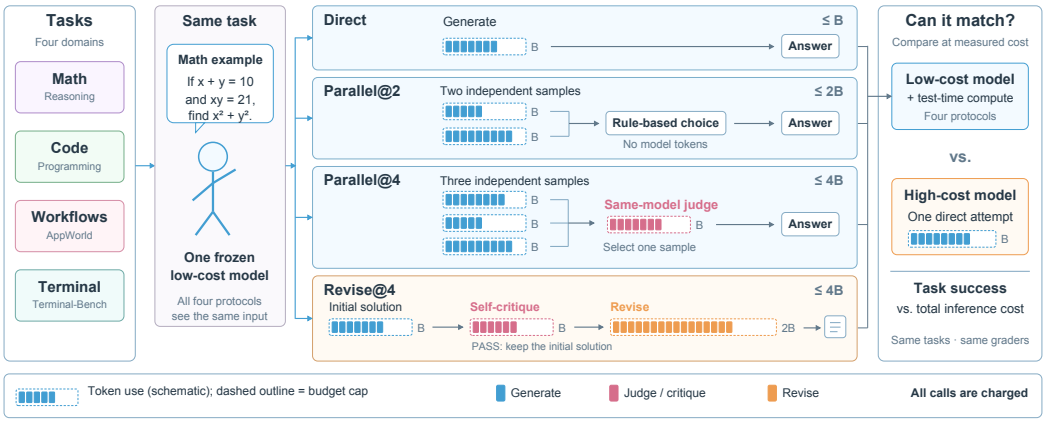


Figure 2: **SELF-POT protocols.** One frozen low-cost model answers the same task under four protocols whose total output is capped at a multiple of the budget B of one answer, and every call is charged. On agent tasks, the parallel protocols compare plans that are executed once, and Revise@4 reviews the trajectory every two steps (§2.2). The mathematics problem is illustrative.

- **Failure handling can reverse the measured gain.** Programming joint selection loses 8.6 pp against Direct under the frozen submission rule; retaining the first available candidate when no nonempty choice is returned changes this to a 3.6 pp gain at unchanged cost (§3.3).
- **Public evidence can convert more of the same candidates.** Programming public-example selection yields 453 correct submissions versus 376 for judging, with 12–49% lower API cost. On identical mathematics pools, judging with fallback reaches 186/292 versus 182/292 for voting (§3.3).
- **Agent gains require completed protocols.** Workflow stage failures reach 117/160 for GPT-OSS and 159/160 for MiniMax. These configurations primarily measure interface reliability; they do not establish the value of completed plan comparison (§3.3).
- **Deployment value differs from within-model gain.** Flash Direct remains the coding frontier after adding replays; discounted mathematics vote@3 adds a cheaper operating point than judging. Opus remains most accurate outside programming among the tested configurations (§3.4).

Our contributions are (1) a measurement protocol that holds the model, information, and grading fixed, caps total output at stated multiples of one attempt, and charges every call (§2); (2) a sealed four-domain test set whose tasks pass a model-independent eligibility check (§2.4); (3) an evaluation of five low-cost models and a stronger anchor, per model and pooled, with the cost of every configuration (§3); and (4) controlled candidate-pool replays separating selection quality, submission failures, and the cost of deployable alternatives (§3.3). We do not propose a new test-time scaling method. The protocols share an accounting rule; their feedback and realized costs differ across domains. Same-pool replays isolate selection and failure handling on static tasks.

2 Methodology

SELF-POT measures how much additional inference a model turns into final task success, and whether the resulting accuracy justifies its measured API cost. The same frozen model supplies every role within a configuration—solver, selector, critic, and agent—and every call it makes is charged.

2.1 Measured quantity and budget

We measure $A(\mathcal{M}, \pi;\ D, I, E, H, B)$, the expected final success rate of model $\mathcal{M}$ under protocol π, given the task distribution D, the information I visible to the solver, the environment E, the harness H, and the budget B. We call the gains attainable under these stated conditions the model's *test-time potential*. The measured rate estimates A for the stated configuration, rather than an intrinsic

property of the model. One execution of a protocol on a task by a model is a *cell*. For domain d, let B_d be the Direct output-token allowance and m_π the protocol multiplier. All model calls $c_1, \ldots, c_T$ in a cell obey

$$\sum_{t=1}^{T} \tau(c_t) \leq m_\pi B_d, \tag{1}$$

where τ counts reasoning and answer tokens. B_d is 131,072 tokens for mathematics and programming and 393,216 for an agent episode, and the caps on model calls, tool calls, and wall-clock time scale with m_π in the same way (Appendix B). These are maximum allowances, not matched realized costs: a protocol may stop early, and provider limits can impose tighter stage caps. We therefore report accuracy together with realized token use and dollar cost; these configurations do not constitute a full accuracy–budget response curve.

2.2 Protocols

Direct answers once with the allowance B_d. On programming, a Direct attempt may run its program on the public examples before submitting it, and on agent tasks it is a complete multi-call episode, so Direct is neither a short answer nor a single request. We do not shorten Direct to create headroom for another method.

Static tasks. Parallel@2 generates two candidates and keeps one by a rule that uses no model tokens: the majority final answer in mathematics and the most public examples passed in programming. Ties go to the earlier candidate: with two parsed answers, voting cannot improve on the first. We therefore also compare three-way voting and judging on identical pools. Parallel@4 generates three candidates and allocates another B to a joint-selection stage: the same model reads all three and names one, with public-tool calls available on code. Revise@4 runs one Self-Refine iteration (Madaan et al., 2023). The model first critiques its answer without writing a new one, a `PASS` verdict returns the initial answer, and otherwise the model revises it with up to $2B$ tokens. The suffix is a budget cap, not a sample count, and Parallel@4 and Revise@4 share the cap but not their realized cost. We omit revision at $2B$ because our allocation gives each of its three stages a full allowance (Appendix E).

Agent tasks. Actions change the environment, so no agent protocol resets an episode or chooses among executed worlds. Parallel@k with $k \in \{2, 4\}$ drafts k plans from the initial observation, selects one with a same-model call, and executes only that plan in a single episode. Revise@4 inserts a same-model review after every two executor turns, appends its advice to the history, and continues from the current state. Planning and plan selection run at low reasoning effort and execution and review at high effort, with all stages sharing the cell's allowance (Appendix D).

2.3 From candidate coverage to final accuracy

For a protocol that selects from a recorded pool of candidates, let O indicate that at least one candidate passes the hidden grader and S that the final submission is correct. Since the submission must be one of the candidates,

$$\mathbb{E}[S] = P(O)\, P(S \mid O), \tag{2}$$

the product of *candidate coverage* and *conversion*. We partition covered failures into no valid selection index and an indexed candidate that grades incorrect, including an empty candidate. Coverage uses hidden grades, so it is an offline ceiling for the recorded candidates rather than a deployable oracle. To isolate conversion, we replay task-appropriate choice rules on the saved Parallel@4 pools: majority voting over three mathematics answers and public-example selection over three programs, each with an earliest-candidate tie rule. We also assess a deterministic fallback that retains the first nonempty candidate when judging yields no usable answer. These post-hoc analyses hold generation fixed; they do not constitute new equal-cost trials. Public examples were available to the model judge as well, and hidden grades are used only to score the resulting choices. Fallback retains the already-incurred judge cost; a rule that omits judging excludes that stage's cost (Appendix K.2). For revision, let p be the accuracy of the protocol's own initial answers, r the fraction of initially incorrect submissions that become correct, and d the fraction of correct answers lost in the final

submission. Then

$$\Delta = (1 - p)\, r - p\, d, \tag{3}$$

so for $r + d > 0$ revision gains exactly when $p < r/(r + d)$. These transitions include empty submissions; we distinguish output failures from changes to completed answers. Unchosen agent plans are never executed, so coverage is undefined for agent protocols, which we compare on the same task across separate episodes rather than through repairs within one trajectory. Both identities are elementary accounting that locates where an opportunity is lost.

2.4 Test set

The mathematics problems come from APEX (Balunovic et al., 2025; MathArena, 2025) and Omni-MATH (Gao et al., 2025), the programming problems from AtCoder through LiveCodeBench v6 (Jain et al., 2025), and the agent tasks from AppWorld (Trivedi et al., 2024) and Terminal-Bench (Merrill et al., 2026) (Appendix Table 5). The domains differ in the feedback available while solving and in whether actions can be undone. Mathematics gives no feedback, programming gives execution on the public examples, and the two agent domains return API responses or shell output from an environment whose changes persist. Tasks pass a model-independent eligibility check: where a reference solution exists, it must pass and a blank or no-op submission must fail under the pinned runtime. Every task seen by a model during development was excluded, and the held-out tasks were drawn within difficulty strata with a fixed seed and at most one task per leakage group, then sealed before any test-time call (Appendix C). No test item was chosen for an observed test-time gain. Task type, difficulty, feedback, and protocol instantiation change together across domains, so a contrast between domains does not isolate any one of these factors.

2.5 Grading and cost accounting

Mathematics uses a rule-based expression grader, programming runs hidden tests, and workflow and terminal cells use their benchmarks' native final-state evaluators (Appendix F). An absent or ungradable submission counts as incorrect, including the 15 scheduled cells that have no final record. For agents, the native grade of the final state is authoritative even when an episode stops at a resource limit, and we report termination status separately. Selection, critique, planning, review, and execution calls all count toward the cell's allowance and cost. Identical requests may reuse stored responses across static protocols, but every configuration is charged the logical cost of its own calls. Reported costs are mean logical API costs over retained cells at the configured rates, not invoices: DeepSeek is charged at its peak rate, with a half-price scenario where it matters, and Bedrock models at list rates (Bedrock prompt-cache discounts apply only to Opus agent requests). Model outcomes are never retried, and bounded infrastructure retries and interface amendments made after sealing are documented in Appendix I.

3 Experiments

3.1 Setup

Models. Five lower-priced reasoning models—DeepSeek V4.1 Flash, Qwen3-Next-80B-A3B, GLM-5, GPT-OSS-120B, and MiniMax M2.5—are compared with Claude Opus 5.5 Direct as the stronger anchor (Appendix J). "Lower-priced" refers to token rates, not to cost per solved task. Only Flash and Qwen3-Next run revision, because the other models' context windows cannot hold a full-budget critique on top of an answer. GPT-OSS and MiniMax run only Direct on terminal tasks, an adaptive omission after their zero Direct scores, and unrun cells are not counted as failures. The matrix has 68 configurations and 6,180 scheduled cells, of which 6,165 have a final record.

Statistics. Each cell is sampled once. Within a model and domain, we count paired wins (protocol correct, Direct incorrect) and losses and apply an exact two-sided sign test, Holm-corrected across all 44 protocol-versus-Direct comparisons. For a pooled domain estimate, we average the paired change over the fixed panel of evaluated models and resample tasks within the frozen strata, keeping all models' outcomes on a task together (20,000 replicates, pointwise 95% intervals; Appendix K).

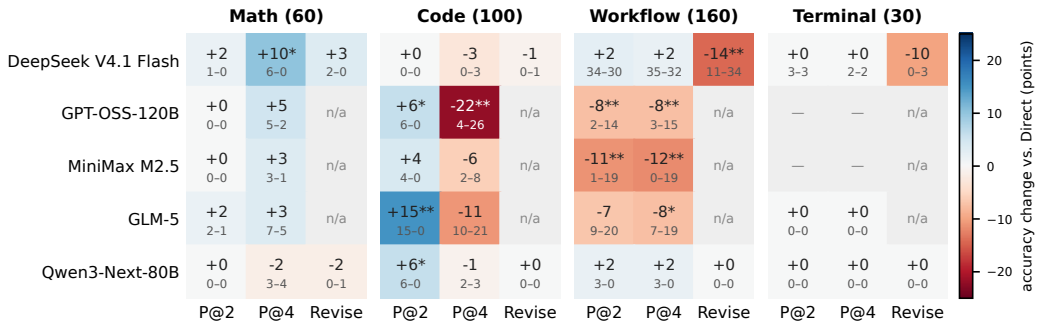


Figure 3: Frozen protocol matrix: accuracy change against the same model's Direct run, in percentage points (large) and paired wins–losses (small). Stars: uncorrected exact sign tests ($^{*}p < 0.05$, $^{**}p < 0.01$); Holm-adjusted tests are in Table 8. n/a: excluded from revision by context; —: not run.

Table 1: Frozen protocol matrix: mean paired accuracy change Δ (pp) against Direct over the fixed panel of low-cost models (five for Parallel, three on terminal tasks, two for Revise@4), with pointwise 95% task-bootstrap intervals and pooled wins–losses (W–L). The intervals describe this panel, not unseen model families.

	Parallel@2			Parallel@4			Revise@4		
Domain	Δ	95% interval	W–L	Δ	95% interval	W–L	Δ	95% interval	W–L
Math	$+0.7$	$[-0.7, +2.0]$	3–1	$+4.0$	$[+0.3, +7.7]$	24–12	$+0.8$	$[-1.7, +3.3]$	2–1
Code	$+6.2$	$[+4.2, +8.2]$	31–0	-8.6	$[-11.8, -5.4]$	18–61	-0.5	$[-1.5, +0.0]$	0–1
Workflow	-4.2	$[-7.2, -1.4]$	49–83	-4.6	$[-7.4, -1.9]$	48–85	-7.2	$[-11.2, -3.4]$	11–34
Terminal	$+0.0$	$[-5.6, +5.6]$	3–3	$+0.0$	$[-4.4, +4.4]$	2–2	-5.0	$[-10.0, +0.0]$	0–3

3.2 RQ1: How gains vary across protocols and domains

Figure 3 gives each model's paired change and Table 1 the panel means. A larger allowance does not order the protocols by accuracy. On programming, Parallel@2 gains 31 task–model outcomes and loses none, improving four models while Flash stays at 99/100. Parallel@4, which adds a third candidate and a same-model judge, records 18 wins and 61 losses, a panel mean of -8.6 percentage points (pp).

Mathematics responds differently. Parallel@4 raises the accuracy of four of five models, with a panel mean of $+4.0$ pp and a 95% interval of $[0.3, 7.7]$. Flash improves from 46 to 52 of 60 tasks (nominal $p = 0.031$, not significant after Holm correction); Qwen3-Next moves from 31 to 30. Two-way voting preserves the first parsed answer, so its small mathematics change does not measure the value of self-consistency at larger sample counts. On workflows, plan-level parallelism lowers the panel mean by 4.2 and 4.6 pp: Flash and Qwen3-Next gain a few tasks, while GLM-5, GPT-OSS, and MiniMax lose. Periodic self-review lowers Flash from 78 to 55 of 160 workflows, with 11 wins and 34 losses (Holm-adjusted $p = 0.033$). On terminal tasks, Parallel preserves aggregate accuracy, although Flash exchanges solved tasks with Direct; Revise lowers Flash from 3/30 to 0/30.

Finding 1. Gains depend on both the domain and the choice rule. Public-example selection improves four coding models and preserves the fifth. Joint selection's lower programming score motivates separating candidate generation from selection and submission. Cross-domain differences do not isolate feedback or difficulty as causes.

3.3 RQ2: Converting candidate coverage into task success

Coverage does not guarantee a correct submission. Figure 4 separates generation from conversion in Parallel@4. In mathematics, Qwen3-Next generates a correct candidate for 38 tasks but submits only 30 correctly; each lost opportunity is a wrong choice. In programming, GLM-5 covers

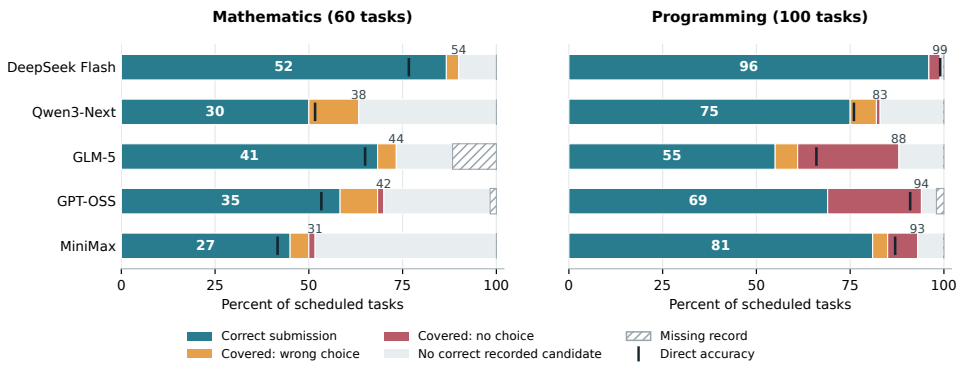


Figure 4: Candidate coverage and conversion under frozen Parallel@4. Each bar partitions scheduled tasks. White numbers count correct submissions, labels above bars count covered tasks, and black ticks mark Direct accuracy. "No choice" denotes no valid index; "wrong choice" includes selecting an empty candidate. Same-pool alternatives are compared in Appendix K.2.

88 tasks but submits 55 correctly, with missing choices accounting for most of the gap. The additional computation has already produced useful answers in these cases; the remaining question is how to deliver them.

Selection and fallback recover much of this potential. On the same programming pools, retaining the first nonempty candidate when judging yields no answer raises correct submissions from 376 to 437 of 500 scheduled cells. This alone reverses the mean change against Direct from -8.6 to $+3.6$ pp, with the judge's cost still charged. Public-example selection yields 453/500 ($+6.8$ pp against Direct), converting 453 of 457 covered pools while saving 12–49% of logical API cost across models. Among the 98 pools with both correct and incorrect candidates, it solves 94 versus 78 for fallback (16 paired wins, no losses): selection adds value beyond preserving an output. In mathematics, judging with the same fallback yields 186 correct submissions on 292 retained pools, compared with 182 for vote@3. This smaller observed difference has a paired interval spanning zero, while voting saves 12–21% of API cost. Grading sensitivity, per-model counts, uncertainty, and accounting details appear in Appendix K.2.

Finding 2. Candidate coverage becomes useful performance through task-appropriate selection and reliable submission. Programming's apparent TTS loss reverses with answer-preserving fallback, and public examples recover nearly all recorded coverage. The smaller mathematics difference shows why a judge's marginal value must be measured against a cheaper choice rule.

Revision and protocol execution. Revision likewise requires separating content changes from lost submissions. Flash's two mathematics recoveries comprise one repaired answer and one recovered empty output; Qwen3-Next changes one correct answer to a wrong one. Flash's sole coding loss follows a truncated critique and an empty final submission; retaining its completed initial program restores the score at unchanged cost. Thus lower revision accuracy alone does not establish that a correct answer was rewritten incorrectly (Appendix K). For agents, missing plans or malformed choices frequently prevent GPT-OSS and MiniMax workflow protocols from reaching execution (Appendix K.1). These outcomes measure the model–protocol–interface combination and leave the quality of unexecuted plans unresolved.

Finding 3. Agent completion and plan quality remain separate questions. Frequent pre-execution failures prevent judging the value of a functioning multi-plan strategy. Static fallback does not transfer to a changed environment; agent interface validation and repeated episodes remain necessary.

3.4 RQ3: When extra inference is competitive with a stronger model

Within-model gains and deployment value are different tests. Opus 5.5 Direct leads in mathematics (57/60 against Flash's best 52/60), workflows (148/160 against 82/160), and terminal tasks

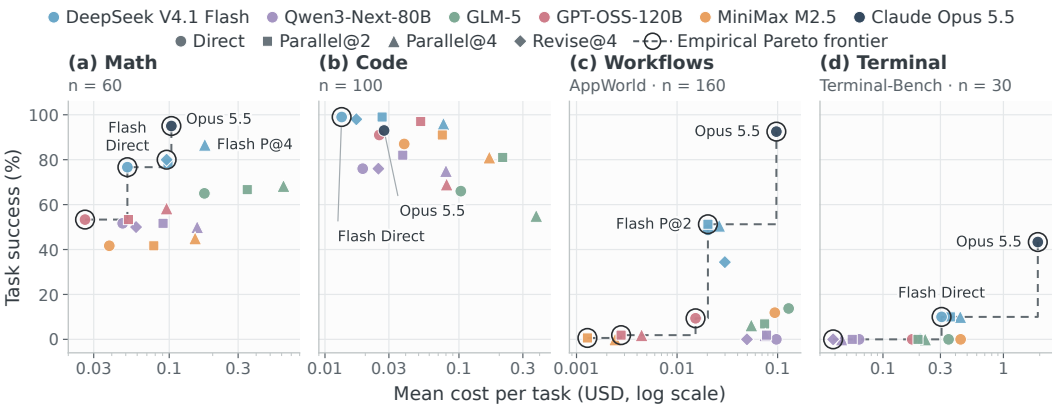


Figure 5: Cost and accuracy of the frozen configurations; rings mark their Pareto frontier. Selector replays are compared separately in Appendix K.2. Costs are mean logical API costs per task over retained cells (DeepSeek peak rates; Bedrock list rates, with Bedrock caching only for Opus agent requests). Code uses the official grades (Opus 93%, 99% with `numpy`), and very low workflow costs can reflect early protocol failures. The frontiers are point estimates, not certified dominance relations.

(13/30 against 3/30). In programming, GPT-OSS Parallel@2 exceeds the anchor's frozen score (97 against 93) but costs more ($0.052 against $0.028); Flash Direct scores 99/100 at $0.013. Supplying the anchor's missing `numpy` dependency raises it to 99/100 without new inference (Appendix I); GPT-OSS's advantage is therefore runtime-dependent. Flash Direct still dominates the tested coding replays in API cost and observed accuracy.

The mathematics trade-off changes with the price paid. At peak rates, Flash judging (52/60, $0.176/task) and vote@3 (50/60, $0.144) are both dominated by Opus Direct (57/60, $0.103). At half price, they cost $0.088 and $0.072 and both lie on the expanded frontier. The replay thus adds a cheaper operating point; judging buys two further correct answers (Appendix K.2).

Finding 4. Whether test-time scaling pays off depends on the alternatives and on the price basis. Coding replays substantially improve several models without displacing Flash Direct. Discounted mathematics inference adds accuracy–cost trade-offs while remaining below the stronger reference. The choice depends on the model, selector, runtime, and price.

4 Related Work

Test-time scaling. Parallel methods generate several candidates and select one, with a trained verifier that scores answers or reasoning steps (Cobbe et al., 2021; Lightman et al., 2024; Zhang et al., 2025; Saad-Falcon et al., 2025) or without one, by majority vote (Wang et al., 2023; Chen et al., 2023b), by executing programs on tests that the model writes (Chen et al., 2023a), or by letting the model judge the candidates (Chen et al., 2025b; Zhao et al., 2025). Selection often limits the gains: coverage keeps rising with more samples while voting and reward-model selection level off (Brown et al., 2024), an imperfect verifier caps the accuracy that resampling can reach (Stroebl et al., 2026), and verification draws on the generation budget (Singhi et al., 2025). Sequential methods revise an earlier attempt (Madaan et al., 2023). Without external feedback, self-correction rarely improves reasoning and can turn correct answers into wrong ones (Huang et al., 2024; Stechly et al., 2025; Kamoi et al., 2024), whereas feedback from tools, execution, or the environment (Gou et al., 2024; Chen et al., 2024c; Shinn et al., 2023) and a model trained to revise (Snell et al., 2025) help more, at a cost that can offset the gain (Olausson et al., 2024). Reasoning models also check their work inside one long chain of thought (Muennighoff et al., 2025), which beyond a point does worse than parallel sampling at the same budget (Ghosal et al., 2025; Zeng et al., 2025), and tree search (Yao et al., 2023) and debate (Du et al., 2024) combine the two families. SELF-POT runs the simplest verifier-free form of each family that a reasoning model can execute by itself.

Table 2: Controlled studies of test-time scaling. M/C/A/O: mathematics, code, agent tasks, other; P/S: parallel, sequential. *Judge charged*: selection calls count against the generation budget (—: no judge call). *vs. stronger*: a cheaper model with more compute is compared with a stronger model.

Study	Domains	Methods	Judge charged	Budget unit	Reasoning models	vs. stronger
Snell et al. (2025)	M	P, S	✗	samples, FLOPs	✗	✓
Wu et al. (2025)	M, C	P	✗	FLOPs	✗	✓
Wang et al. (2024)	M, O	P, S	✓	tokens, $	✗	✗
Chen et al. (2024a)	O	P	✗	calls	✗	✗
Singhi et al. (2025)	M, O	P	✓	FLOPs	✓	✗
Ghosal et al. (2025)	M	P, S	—	tokens	✓	✗
Zeng et al. (2025)	M, O	P, S	—	tokens	✓	✗
Stroebl et al. (2026)	C	P	—	samples	✗	✓
Kapoor et al. (2025a)	A	S	—	$	✓	partly
Li et al. (2026)	A, M	P, S	✗	samples, tokens	✓	✗
Balachandran et al. (2025)	M, O	P, S	✗	tokens, calls	✓	✓
SELF-POT (ours)	M, C, A	P, S	✓	tokens, $	✓	✓

Controlled comparisons. Several studies compare test-time scaling methods under a matched budget (Table 2). In mathematics, a smaller model given more inference compute can match or surpass a larger one on questions it already solves at a non-trivial rate, and voting saturates as samples grow (Snell et al., 2025; Wu et al., 2025). Across single-turn reasoning tasks, self-consistency frequently outperforms more elaborate strategies at equal budget (Wang et al., 2024), and accuracy can first rise and then fall as the number of calls grows (Chen et al., 2024a). For agents, higher reasoning effort gives equal or lower accuracy in 21 of 36 model–agent–benchmark combinations (Kapoor et al., 2025a), and neither sequential nor parallel scaling helps general agents much (Li et al., 2026). A concurrent report covers mathematics, science questions, and planning across nine models (Balachandran et al., 2025). None of these studies combines mathematics, programming, and agent tasks with charged selection calls and a comparison against a stronger model, which is the combination SELF-POT provides.

Cost-aware evaluation. Holistic evaluation suites report efficiency in runtime and hardware units (Liang et al., 2023), agent evaluations argue for and adopt joint reporting of dollar cost and accuracy (Kapoor et al., 2025b;a), and cost-of-pass measures the expected dollar cost of one correct answer (Erol et al., 2025). Test-time scaling studies mostly count FLOPs (Snell et al., 2025; Wu et al., 2025), yet the unit of cost matters: inference demand changes the compute-optimal model size (Sardana et al., 2024), memory-access cost removes much of the apparent advantage of small models under best-of-N sampling and long reasoning (Sadhukhan et al., 2025), and models often reason far longer than a problem requires (Chen et al., 2025a; Han et al., 2025). Cascades and routers instead choose a model for each query (Chen et al., 2024b; Yue et al., 2023; Gupta et al., 2024; Aggarwal et al., 2024; Ding et al., 2024; Ong et al., 2025; Dekoninck et al., 2025), sometimes together with the number of samples (Ding et al., 2025), and adaptive stopping rules reduce the samples a single model needs (Aggarwal et al., 2023; Li et al., 2024; Damani et al., 2025). These approaches compare models at their native settings or assume that sampling a cheaper model is the more economical path, an assumption that SELF-POT tests directly.

5 Conclusion

SELF-POT measures how additional inference becomes task success across 350 sealed tasks. On fixed programming pools, public selection raises correct submissions from 376 to 453; output-preserving fallback alone recovers 61. Selection and failure handling can reverse an observed TTS loss. Mathematics judging with fallback adds four correct submissions over cheaper voting; discounted vote@3 adds a deployment trade-off, while Flash Direct retains the coding frontier. Agent interfaces require validation before attributing their outcomes to planning. These measurements connect candidate coverage, executable selection rules, and cost to inference allocation. Limitations appear in Appendix A.2.

AI USE STATEMENT

We used generative AI assistants to help implement and test the evaluation harness and the analysis scripts, to monitor experiment runs, to search and summarize related work, and to draft and edit the text. Our experimental results and diagnostic plots are computed from retained result files. The authors reviewed the AI-assisted code and text and take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

SELF-POT evaluates existing models on existing public benchmarks and involves no human subjects. Following AppWorld's usage terms, we report only aggregate results on its test split and do not inspect individual test trajectories (Appendix F).

REPRODUCIBILITY STATEMENT

Section 2 and Appendices B–J specify the budgets, test-set construction, protocols and parse rules, graders, agent harness, accounting rules, and model settings, and Appendix I lists every change made after the test set was sealed. Diagnostics in Section 3.3 and Appendix K use retained records without new model calls. The selector replay additionally executes stored programs on public examples; scripts and accounting rules are specified in Appendix K.2. Candidate-composition and grading-sensitivity analyses reuse saved public-test results and verifier judgments (Appendix K.2).

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 Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 12375–12396. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.761. URL https://doi.org/10.18653/v1/2023.emnlp-main.761.

Pranjal Aggarwal, Aman Madaan, Ankit Anand, Srividya Pranavi Potharaju, Swaroop Mishra, Pei Zhou, Aditya Gupta, Dheeraj Rajagopal, Karthik Kappaganthu, Yiming Yang, Shyam Upadhyay, Manaal Faruqui, and Mausam. Automix: Automatically mixing language models. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/ecda225cb187b40ea8edc1f46b03ffda-Abstract-Conference.html.

Vidhisha Balachandran, Jingya Chen, Lingjiao Chen, Shivam Garg, Neel Joshi, Yash Lara, John Langford, Besmira Nushi, Vibhav Vineet, Yue Wu, and Safoora Yousefi. Inference-time scaling for complex tasks: Where we stand and what lies ahead. *CoRR*, abs/2504.00294, 2025. doi: 10.48550/ARXIV.2504.00294. URL https://doi.org/10.48550/arXiv.2504.00294.

Mislav Balunovic, Jasper Dekoninck, Ivo Petrov, Nikola Jovanovic, and Martin T. Vechev. Matharena: Evaluating llms on uncontaminated math competitions. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/1d27c01ebd3e3aebe226b44fc970d803-Abstract-Datasets_and_Benchmarks_Track.html.

Bradley C. A. Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V. Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with re-

peated sampling. *CoRR*, abs/2407.21787, 2024. doi: 10.48550/ARXIV.2407.21787. URL https://doi.org/10.48550/arXiv.2407.21787.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023a. URL https://openreview.net/forum?id=ktrw68Cmu9c.

Lingjiao Chen, Jared Quincy Davis, Boris Hanin, Peter Bailis, Ion Stoica, Matei A. Zaharia, and James Y. Zou. Are more LLM calls all you need? towards the scaling properties of compound AI systems. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024a. URL http://papers.nips.cc/paper_files/paper/2024/hash/51173cf34c5faac9796a47dc2fdd3a71-Abstract-Conference.html.

Lingjiao Chen, Matei Zaharia, and James Zou. Frugalgpt: How to use large language models while reducing cost and improving performance. *Trans. Mach. Learn. Res.*, 2024, 2024b. URL https://openreview.net/forum?id=cSimKw5p6R.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/chen25bx.html.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language model generation. *CoRR*, abs/2311.17311, 2023b. doi: 10.48550/ARXIV.2311.17311. URL https://doi.org/10.48550/arXiv.2311.17311.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024c. URL https://openreview.net/forum?id=KuPixIqPiq.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. Provable scaling laws for the test-time compute of large language models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/f3f607e4c13bd1cb8885de44b4ec45b7-Abstract-Conference.html.

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. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Mehul Damani, Idan Shenfeld, Andi Peng, Andreea Bobu, and Jacob Andreas. Learning how hard to think: Input-adaptive allocation of LM computation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=6qUUgw9bAZ.

Jasper Dekoninck, Maximilian Baader, and Martin T. Vechev. A unified approach to routing and cascading for llms. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International*

Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/dekoninck25a.html.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=02f3mUtqnM.

Dujian Ding, Ankur Mallick, Shaokun Zhang, Chi Wang, Daniel Madrigal, Mirian del Carmen Hipolito Garcia, Menglin Xia, Laks V. S. Lakshmanan, Qingyun Wu, and Victor Rühle. Best-route: Adaptive LLM routing with test-time optimal compute. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/ding25d.html.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/du24e.html.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yüksekgönül, and James Zou. Cost-of-pass: An economic framework for evaluating language models. *CoRR*, abs/2504.13359, 2025. doi: 10.48550/ARXIV.2504.13359. URL https://doi.org/10.48550/arXiv.2504.13359.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=yaqPf0KAlN.

Soumya Suvra Ghosal, Souradip Chakraborty, Avinash Reddy, Yifu Lu, Mengdi Wang, Dinesh Manocha, Furong Huang, Mohammad Ghavamzadeh, and Amrit Singh Bedi. Does thinking more always help? mirage of test-time scaling in reasoning models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/fc067ac218430c409d6f65403328f740-Abstract-Conference.html.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Neha Gupta, Harikrishna Narasimhan, Wittawat Jitkrittum, Ankit Singh Rawat, Aditya Krishna Menon, and Sanjiv Kumar. Language model cascades: Token-level uncertainty and beyond. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=KgaBScZ4VI.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, volume ACL 2025 of *Findings of ACL*, pp. 24842–24855. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.FINDINGS-ACL.1274. URL https://doi.org/10.18653/v1/2025.findings-acl.1274.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, 2026. URL https://doi.org/10.5281/zenodo.20953922.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=IkmD3fKBPQ.

Naman Jain, King Han, 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 *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms *Actually* correct their own mistakes? A critical survey of self-correction of llms. *Trans. Assoc. Comput. Linguistics*, 12:1417–1440, 2024. doi: 10.1162/TACL_A_00713. URL https://doi.org/10.1162/tacl_a_00713.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S. Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, Juyong Lee, Tengjun Jin, Yifan Mai, Yifei Zhou, Yuxuan Zhu, Rishi Bommasani, Daniel Kang, Dawn Song, Peter Henderson, Yu Su, Percy Liang, and Arvind Narayanan. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. *CoRR*, abs/2510.11977, 2025a. doi: 10.48550/ARXIV.2510.11977. URL https://doi.org/10.48550/arXiv.2510.11977.

Sayash Kapoor, Benedikt Stroebl, Zachary S. Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *Trans. Mach. Learn. Res.*, 2025, 2025b. URL https://openreview.net/forum?id=Zy4uFzMviZ.

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. *CoRR*, abs/2602.18998, 2026. doi: 10.48550/ARXIV.2602.18998. URL https://doi.org/10.48550/arXiv.2602.18998.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=ndR8Ytrzhh.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel J. Orr, Lucia Zheng, Mert Yüksekgönül, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=iO4LZibEqW.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Shudong Liu, Hongwei Liu, Junnan Liu, Linchen Xiao, Songyang Gao, Chengqi Lyu, Yuzhe Gu, Wenwei Zhang, Derek F. Wong, Songyang Zhang, and Kai Chen. Compassverifier: A unified and robust verifier for llms evaluation and outcome reward. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 33466–33494. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1698. URL https://doi.org/10.18653/v1/2025.emnlp-main.1698.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

MathArena. MathArena Apex. https://matharena.ai/apex/, 2025.

Mike A. Merrill, Alexander Glenn Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Yeon Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Michael Rytting, Ryan Marten, Yixin Wang, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *CoRR*, abs/2601.11868, 2026. doi: 10.48550/ARXIV.2601.11868. URL https://doi.org/10.48550/arXiv.2601.11868.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel J. Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 20275–20321. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.EMNLP-MAIN.1025. URL https://doi.org/10.18653/v1/2025.emnlp-main.1025.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=y0GJXRungR.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. Routellm: Learning to route llms from preference data. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore,*

April 24-28, 2025. OpenReview.net, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

Jon Saad-Falcon, Estefany Kelly Buchanan, Mayee F. Chen, Tzu-Heng Huang, Brendan McLaughlin, Tanvir Bhathal, Shang Zhu, Ben Athiwaratkun, Frederic Sala, Scott W. Linderman, Azalia Mirhoseini, and Christopher Ré. Weaver: Shrinking the generation-verification gap by scaling compute for verification. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/715f48023c26ad210321a4c0da67f27c-Abstract-Conference.html.

Ranajoy Sadhukhan, Zhuoming Chen, Haizhong Zheng, Yang Zhou, Emma Strubell, and Beidi Chen. Kinetics: Rethinking test-time scaling laws. *CoRR*, abs/2506.05333, 2025. doi: 10.48550/ARXIV.2506.05333. URL https://doi.org/10.48550/arXiv.2506.05333.

Nikhil Sardana, Jacob P. Portes, Sasha Doubov, and Jonathan Frankle. Beyond chinchilla-optimal: Accounting for inference in language model scaling laws. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pp. 43445–43460. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/sardana24a.html.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Nishad Singhi, Hritik Bansal, Arian Hosseini, Aditya Grover, Kai-Wei Chang, Marcus Rohrbach, and Anna Rohrbach. When to solve, when to verify: Compute-optimal problem solving and generative verification for LLM reasoning. *CoRR*, abs/2504.01005, 2025. doi: 10.48550/ARXIV.2504.01005. URL https://doi.org/10.48550/arXiv.2504.01005.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. On the self-verification limitations of large language models on reasoning and planning tasks. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=4O0v4s3IzY.

Benedikt Stroebl, Sayash Kapoor, and Arvind Narayanan. The limits of inference scaling through resampling. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=j8H84v6AZ1.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 16022–16076. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.850. URL https://doi.org/10.18653/v1/2024.acl-long.850.

Junlin Wang, Siddhartha Jain, Dejiao Zhang, Baishakhi Ray, Varun Kumar, and Ben Athiwaratkun. Reasoning in token economies: Budget-aware evaluation of LLM reasoning strategies. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pp. 19916–19939. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-MAIN.1112. URL https://doi.org/10.18653/v1/2024.emnlp-main.1112.

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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for LLM problem-solving. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=VNckp7JEHn.

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. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Murong Yue, Jie Zhao, Min Zhang, Liang Du, and Ziyu Yao. Large language model cascades with mixture of thoughts representations for cost-efficient reasoning. *CoRR*, abs/2310.03094, 2023. doi: 10.48550/ARXIV.2310.03094. URL https://doi.org/10.48550/arXiv.2310.03094.

Zhiyuan Zeng, Qinyuan Cheng, Zhangyue Yin, Yunhua Zhou, and Xipeng Qiu. Revisiting the test-time scaling of o1-like models: Do they truly possess test-time scaling capabilities? In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pp. 4651–4665. Association for Computational Linguistics, 2025. doi: 10.18653/V1/2025.ACL-LONG.232. URL https://doi.org/10.18653/v1/2025.acl-long.232.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=Ccwp4tFEtE.

Eric Zhao, Pranjal Awasthi, and Sreenivas Gollapudi. Sample, scrutinize and scale: Effective inference-time search by scaling verification. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/zhao25a.html.

A Extended discussion and limitations

A.1 Discussion

Test-time potential is a conversion profile. The most informative distinction in our measurements is between producing a correct candidate and successfully submitting it. Qwen3-Next's mathematics coverage exceeds its Direct accuracy while judging does not. GLM-5's 88 covered programming pools produce 55 correct submissions under the original judge, 78 with nonempty fallback, and 87 with public-example selection. An aggregate gain alone would hide these opportunities, and an aggregate loss alone would conflate wrong decisions with stages that return no decision. Coverage, conversion, and revision transitions separated by output availability make these cases distinguishable without proposing a new composite score. The contribution is a concrete measurement framework and its cross-domain evidence, rather than a claim that one protocol realizes all of a model's potential.

Improve the limiting stage before increasing its allowance. The coding results motivate making stage completion an explicit design objective: most losses against Direct occur without a usable final selection. Returning a completed candidate when judging fails recovers 61 coding submissions without new inference, demonstrating an avoidable failure-handling loss rather than a limit of candidate generation. For models such as Qwen3-Next in mathematics, however, the selection stage completes and still discards correct candidates, so the relevant target is decision quality. This distinction sharpens the connection to work separating sampling from selection (Brown et al., 2024) and examining feedback in self-correction (Huang et al., 2024; Kamoi et al., 2024). Public-example selection converts 453 of the 457 covered programming pools, improving on the frozen judge while avoiding its API cost. In the 98 pools with mixed correctness, it converts 94 versus 78 for nonempty fallback, separating the benefit of choosing a better candidate from that of retaining an answer (Appendix K.2). A public-first gate still calls the judge on 465 of 498 retained pools because scores often tie, and yields 452 correct submissions. Thus public evidence is useful even when incomplete; in these pools the extra judge does not improve on the earliest-best tie rule. This is evidence about the tested selector, not about all possible judges or learned verifiers. On mathematics, judging yields three more correct submissions than voting, or four with nonempty fallback, on identical pools; both paired intervals cross zero. Removing the judge saves 12–21% of the judge-based configuration's logical API cost. The saved Compass judgments preserve the direction of the difference, with 194 correct submissions for judging with fallback versus 189 for voting (Appendix K.2). Neither grader establishes equivalence between the selectors.

Choosing a method is also choosing a model and a runtime. A deployment decision should compare an entire configuration—model, protocol, harness, and price—against the available alternatives. GLM-5's sizeable coding gain is valuable evidence about its response to inference, yet a cheaper Flash Direct configuration solves more tasks. After incorporating the replays, Flash Direct remains the coding frontier. In mathematics, discounted Flash vote@3 adds 50/60 at \$0.072/task alongside judging's 52/60 at \$0.088; both are dominated by Opus Direct at peak rates. The anchor's coding sensitivity to one missing library further shows why a performance difference should not automatically be interpreted as a capability difference. These observations connect the scientific question of attainable gains to a practical decision: whether to spend the next unit of budget on another candidate, a more reliable selection mechanism, or a different base model.

What should be reported in a TTS evaluation. Our results motivate reporting the Direct operating point and actual resource use, candidate coverage and availability where observable, the outcome of each selection stage, revision transitions, and final native grades alongside termination status. The budget cap and the measured cost answer different questions and should both be visible. A stronger Direct reference makes deployment claims assessable, while an explicit runtime and price sensitivity prevent an incidental environment or tariff from becoming a general conclusion. For agent protocols, unexecuted plans do not provide an oracle coverage estimate; additional controlled executions would be required to separate plan quality from execution reliability.

A.2 LIMITATIONS

Sampling and statistical scope. Each cell is sampled once, so our intervals quantify variation across tasks rather than repeated-run variability. Mathematics and terminal evaluation contain only 60 and 30 tasks; one task changes accuracy by 1.7 and 3.3 pp, respectively. Workflow outcomes also vary substantially across protocols: 64–67 Flash tasks change correctness between Direct and a Parallel run. Stratified task bootstrap intervals retain cross-model dependence, but remain pointwise intervals for a fixed model panel; they do not justify a universal scaling law. The five Holm-surviving per-model comparisons are reported separately from nominal annotations (Table 8).

Coverage of methods and models. We evaluate a small set of same-model protocol families, not the strongest conceivable TTS system. Trained verifiers, specialized debate and repair methods, alternative long-reasoning instructions, and larger candidate pools may change the results. Three models are excluded from revision under our chosen context-headroom requirement; agent review could also be evaluated under a separately calibrated design. Terminal TTS for GPT-OSS and MiniMax was omitted adaptively after earlier results, so the matrix is incomplete by design. The benchmark's source and difficulty composition was informed by development, and a different mixture may produce a different average gain.

Implementation and grading. Static and agent protocols share an allocation principle but differ in their unit of parallelism and available feedback, which prevents attributing cross-domain differences to a single mechanism. Provider-interface amendments, per-call limits, bounded infrastructure retries, and fifteen missing final records are part of the evaluated implementation (Appendix I). Six resource-limited agent episodes still receive correct native grades; status alone is therefore not a substitute for the task evaluator. The math rule grader and Compass disagree on 25 of 1,069 scored cells, and the anchor's coding result is sensitive to a missing dependency. We retain the frozen primary graders and make these sensitivities explicit rather than selecting a favorable regrading. The frozen static revision harness also discards an initial answer if critique or revision fails to produce the required output. This avoidable submission loss is distinct from rewriting a correct answer incorrectly: a deterministic fallback restores one Flash programming answer at unchanged cost (Appendix K).

Cost and diagnostic scope. Logical API costs use configured rates and available records, not invoices; they omit local execution expense, screening, and superseded infrastructure attempts. Cached response reuse saves campaign spending but is charged logically to each evaluated method. Price changes, provider caching, and latency requirements can change a deployment frontier. Candidate coverage is an oracle diagnostic of recorded static pools, not an executable selection policy or a bound on all future sampling. The static replays control candidate strings and expose the effects of selection and fallback. These post-hoc comparisons identify effects on the saved pools; prospective policy performance requires fresh held-out evaluation. They do not replace end-to-end trials, repeated agent episodes, or evaluation of a new judge prompt. Agent interface validation and repeated runs remain necessary before generalizing the observed agent losses to additional planning.

B BUDGETS AND CAPS

Each domain has one unit budget B and a set of hard caps; a protocol at multiplier m receives m times every cap.

Table 3: Nominal stage allowances for static tasks. The suffix denotes a total budget multiplier, not a sample count. Provider limits can reduce individual allowances; in particular, Qwen3-Next's revision is capped at B, giving at most $3B$ across its three stages.

Protocol	Generation	Selection or feedback	Total cap
Direct	B	–	B
Parallel@2	$2 \times B$	rule, no model tokens	$2B$
Parallel@4	$3 \times B$	joint selection, B	$4B$
Revise@4	initial B, revision up to $2B$	critique, B	$4B$

Table 3 and Algorithm 1 specify the static allocations. Parallel@4 and Revise@4 share an upper budget envelope, but their accuracy difference does not isolate a mechanism at identical realized cost.

Table 4: Per-cell caps for Direct ($m = 1$). All caps are multiplied by m for Parallel@2 ($m = 2$), Parallel@4 and Revise@4 ($m = 4$).

Domain	Output tokens B	Model calls	Tool calls	Wall clock
Math	131,072	16	–	2 h
Code	131,072	64	128	2 h
Workflow	393,216[a]	48	64	1 h
Terminal	393,216	128	256	8 h[b]

[a]At most 131,072 per executor turn. [b]The native task limit; a single shell command is additionally limited to 1,800 s (Appendix G).

Within agent protocols, each plan may use up to 32,768 output tokens, plan selection up to 65,536, and each review up to 131,072, and a review call is never capped below an executor turn; all stages share the cell's total allowance. The mathematics allowance exceeds the longest Flash Direct solution observed during development (94,834 tokens); one of Flash's 60 mathematics Direct cells on the test set was truncated. This headroom does not guarantee that every provider-specific stage is untruncated.

C TEST SET CONSTRUCTION

Tasks were admitted by a model-independent eligibility check, drawn by stratum with a fixed seed, and sealed before any test-time call.

Eligibility. Where an executable reference is distributed, eligibility requires that it succeeds and a blank or no-op submission fails under the pinned runtime. AppWorld uses the separate contract below because its test references are unavailable. For code, an official editorial program must pass every hidden test in our container; duplicate references are removed by code hash. AppWorld's test split ships no reference solutions, so eligibility there requires the official development examples to pass, the native evaluator chain to be fixed, and a blank state to fail; all 585 test tasks met this contract. For Terminal-Bench, the Harbor `oracle` agent must obtain reward 1 and the `nop` agent reward 0; 62 of 66 tasks qualified, three were excluded for requiring GPUs, and one because its reference could not run in the shipped image.

Sampling and exclusions. Within each source, tasks were stratified by published difficulty and sampled without replacement using seed 20260917, taking at most one task per scenario cluster or leakage group. Every task seen by any model during development (math 52, code 25, workflow 15, terminal 1) was excluded, together with three AtCoder contests held out as a whole (abc391, abc398, arc191). The set was sealed on 2026-09-21; the seal records hashes of the specification and of the public and private task files, and the tasks did not change afterwards.

Table 5: The four domains of the sealed test set. Hidden answers, tests, and goal checks are used only for grading.

Domain	Source	Tasks	Feedback while solving	Actions reversible
Math	APEX (Balunovic et al., 2025; MathArena, 2025), Omni-MATH (Gao et al., 2025)	60	none	yes
Code	LiveCodeBench v6 (Jain et al., 2025) (AtCoder)	100	public examples and execution	yes
Workflow	AppWorld (Trivedi et al., 2024)	160	app API responses	no
Terminal	Terminal-Bench 4.0 (Merrill et al., 2026)	30	shell output	no

Table 6: Composition of the sealed test set.

Domain	Source	Strata (tasks)
Math (60)	APEX shortlist; APEX 2025 (MathArena)	27; 7
	Omni-MATH-Rule, difficulty $[3.5, 5)$ / $[5, 6)$ / $[6, 7)$ / $[7, 10)$	8 / 8 / 5 / 5
Code (100)	LiveCodeBench v6, AtCoder, 2023-08 to 2025-04	easy / medium / hard: 30 / 28 / 42
Workflow (160)	AppWorld test (data 0.1.0), normal, levels 1 / 2 / 3	17 / 14 / 19
	AppWorld test, challenge, levels 1 / 2 / 3	19 / 40 / 51
Terminal (30)	Terminal-Bench 4.0.0 via Harbor	easy / medium / hard: 7 / 11 / 12

The programming test set contains 42 hard, 28 medium, and 30 easy tasks. Its composition was fixed before test inference; development performance informed the source and quota choices, not the selection of individual held-out items by observed TTS gain.

D PROTOCOL PSEUDOCODE

Algorithm 1 Static protocols. $\mathcal{M}(\cdot\,; b)$ denotes a generation stage with output allowance b; on code it can include multiple calls with public-tool feedback. Provider limits can reduce a stage allowance. Identical requests may reuse stored samples across protocols, charged logically to each.

function DIRECT(x)
  return $\mathcal{M}(x;\, B)$
end function
function PARALLEL@2(x)
  $y_1, y_2 \leftarrow \mathcal{M}(x;\, B)$ ▷ independent draws
  return RULE(y_1, y_2) ▷ no model tokens
end function
function PARALLEL@4(x)
  $y_1, y_2, y_3 \leftarrow \mathcal{M}(x;\, B)$
  $j \leftarrow \mathcal{M}(\text{JUDGE}(x, y_1, y_2, y_3);\, B)$ ▷ same model reads all three
  return y_j
end function
function REVISE@4(x)
  $y_0 \leftarrow \mathcal{M}(x;\, B)$
  $f \leftarrow \mathcal{M}(\text{CRITIQUE}(x, y_0);\, B)$ ▷ feedback only, no new answer
  if $f = \texttt{PASS}$ **then**
    return y_0
  end if
  return $\mathcal{M}(\text{REVISE}(x, y_0, f);\, 2B)$
end function

Algorithm 2 Protocols on agent tasks. o_0 is the initial observation; the whole episode obeys Eq. (1).

function PARALLEL@k(x)
  $p_i \leftarrow \mathcal{M}(\text{PLAN}(x, o_0))$ for $i = 1, \ldots, k$ ▷ no tool calls
  $j \leftarrow \mathcal{M}(\text{CHOOSE}(p_1, \ldots, p_k))$
  return EXECUTE(x, p_j) ▷ a single episode
end function
function REVISE@4(x)
  $h \leftarrow (x, o_0)$
  while the executor issues a tool call and the budget allows **do**
    if two turns have passed since the last review **then**
      $h \leftarrow h \,\|\, \mathcal{M}(\text{REVIEW}(h))$ ▷ continue, repair, replan, or stop
    end if
    $h \leftarrow h \,\|\, \text{STEP}(\mathcal{M}, h)$ ▷ one tool call and its observation
  end while
  return final environment state
end function

E PROTOCOL DETAILS

The original protocols require a formatted choice and use no fallback after an unparseable selection. Appendix K.2 evaluates output-preserving alternatives on the same candidates.

Joint selection (Parallel@4, static tasks). The model receives the problem and the three candidates, labelled A to C, and is told to verify each candidate's decisive steps rather than compare their popularity. It must end with `SELECTED: <letter>`. If that line is missing but the last boxed expression is a single candidate letter, the choice is accepted and the parse route is recorded, so the strict score remains recomputable; any other reply is a format failure.

Self-Refine (Revise@4, static tasks). The critique prompt forbids a replacement solution and a final answer, asks the model to quote each faulty step and state what should change, and requires `VERDICT: PASS` or `VERDICT: REVISE`. A `PASS` returns the initial solution without a revision call; an unparseable verdict proceeds to revision. If critique fails to finish or revision yields no completed final answer, the frozen harness returns an empty submission without falling back to the initial answer. Appendix K evaluates this implementation choice offline. Whether the feedback asserts an answer different from the initial one is recorded per cell. Relative to Madaan et al. (2023), we run one feedback–refine iteration and no few-shot demonstrations.

Public-test selection (Parallel@2, code). Each candidate program is compiled and run on the public examples; the one matching the most examples is kept, with ties going to the earlier candidate. Compile failures or empty programs receive a penalty below a runnable program that matches no public examples. Hidden tests are never consulted, and the step consumes no model tokens, although it requires executing the candidates.

Agent protocols. For Parallel@k, the model proposes k plans from the initial observation without calling tools, then replies `SELECT: N`; an invalid selection ends the cell as a stage failure. The chosen plan is appended to the context and executed once. For Revise@4, after every second executor turn the model is asked to inspect the goal, the visible state, and recent errors, to choose among continuing, inspecting and repairing, replanning, or stopping, to preserve actions that already succeeded, and not to claim success without checking the observable state. The review text is appended to the context; the program does not parse the chosen option, and the episode ends when the executor issues no further tool call.

Protocols not included. A separately instructed longer-reasoning protocol was not run. Direct already uses high effort and a generous allowance, but this does not establish equivalence to all alternative long-reasoning prompts or provider effort settings. Revise at $2\times$ is excluded by our choice to allocate a full solution allowance to each of its three stages; other revision allocations remain possible. Domain-specific methods, such as moderated debate for mathematics or test-driven repair for code, were planned as a separate audit and are not part of the reported results.

F GRADING AND RUNTIME

Static outputs use the declared answer or execution grader, and agents use native final-state evaluators; hidden answers and tests are used only for offline grading.

- **Math.** A rule-based expression grader checks the final scalar answer. An offline CompassVerifier-3B model (Liu et al., 2025) grades the same outputs as a second opinion; the two disagree on 25 of 1,069 scored mathematics cells.
- **Code.** Programs run against all hidden tests in a pinned container (image `sha256:4a33fba9...`; CPython 3.11; g++ 12.2 with `-std=c++20`; AtCoder Library). Re-running eligibility for all 282 AtCoder candidates under C++20 in the same image added six tasks and removed none.
- **Workflow.** AppWorld's native evaluator reports task goal completion, which requires every check to pass; we do not call it the official scenario goal completion.
- **Terminal.** Terminal-Bench's native verifier, run by Harbor in a separate container, returns a binary reward.

Following AppWorld's usage terms, we report only aggregate results on its test split and do not inspect individual test trajectories.

G AGENT HARNESS

Agents act through one uniform tool interface per domain, and the harness enforces limits inside the task environment.

Each cell begins from the prescribed initial state, with no memory carried across tasks. Agents can change the environment and may sometimes undo individual actions, but the harness forbids branching into cloned worlds, resetting an episode, or selecting among multiple executed worlds.

Terminal tasks run in their native Harbor (Harbor Framework Team, 2026) containers through a single shell-command tool rather than the tmux interface of the reference Terminus-2 agent, so our scores are not comparable to the public leaderboard. A single command is limited to 1,800 s, the longest native verifier timeout in our terminal set. The limit is enforced inside the container and terminates the whole process group; a stopped command returns exit status 124 to the model, and the tool description states the limit.

The AppWorld environment server serves one episode at a time. Half of the 160 workflow runs used four replica servers that match the original in image, read-only data, and resource limits and differ only in port; each run records the server it used.

H DEVELOPMENT EVIDENCE BEHIND THE DESIGN

Three design rules—full budgets for judges, a judge at the solver's effort, and verification instead of voting in Parallel@4—were adopted after development measurements, before the test set was sealed.

- **Observed judging costs.** A critique at low effort used 58,173 output tokens, 58,100 of them reasoning, to emit 255 characters; critiques capped at $B/8$ or $B/16$ stopped mid-reasoning and returned no verdict. Selection calls used 14,637–61,719 output tokens.
- **A low-effort critique error.** On a problem the solver answered correctly in three of three attempts, a low-effort critique asserted a different answer as established.
- **Development selector comparison.** On 16 development problems, Direct scored 0.625, majority voting over three samples 0.562, and at least one of three samples was correct 81% of the time. On the seven contested problems, joint verification recovered four of five that had a correct candidate; voting recovered one. Self-verification accepted a wrong solution in 7 of 15 usable judgments.
- **Eight-sample pilot.** On the hardest seven development problems with eight samples each, voting peaked at four samples while coverage kept rising.

These small development observations motivated the original design; they do not establish that judging is preferable on held-out tasks. The same-pool test analysis in Appendix K.2 measures that marginal value explicitly.

I ACCOUNTING, FAILURES, AND CHANGES AFTER SEALING

We distinguish logical method cost from campaign spending, task correctness from termination status, and a fixed task seal from versioned implementation amendments.

Cost ledger. Before each call, the ledger reserves the stage cap at the configured price and settles actual usage afterwards; a failed call keeps its reservation. This is conservative for retained calls, but configuration means are not an upper bound on whole-campaign expense: they average retained final cell records and omit superseded infrastructure attempts and records that never completed. Identical requests reused across protocols are charged logically in every configuration, although they are physically paid for only once. DeepSeek V4.1 Flash uses one fixed peak-rate snapshot for every call: \$0.30 per million uncached input tokens, \$1.20 per million output tokens, and \$0.006 per million cache-read tokens. Token counts and cache hits come from the recorded usage; prices do not depend on the call timestamp. The half-price sensitivity multiplies every DeepSeek configuration cost by 0.5, holding usage fixed. Neither scenario reconstructs the actual invoice, although most calls ran off-peak. Bedrock models are charged at list price; Opus 5.5 agent requests use prompt caching, which changes cost but not model inputs or outputs. The GPT-OSS price on the pricing page is the Sydney region's, so its ledger errs high. An accounting audit recovered 12 Flash terminal cell records whose parent runs had not written an aggregate evaluation file. Their grades were already retained, but the merger had defaulted their costs to zero. We restore the saved costs (\$6.20 in total) and token usage, leaving all outcomes unchanged: Direct, Parallel@2, and Parallel@4 cost \$0.307, \$0.363, and \$0.438 per retained task, respectively. The tables and Figure 5 use these corrected values; `analysis/audit_terminal_costs.py` records the source hashes and correction. Terminal frontier membership is unchanged. Ledger totals are about \$101 for DeepSeek (including screening), \$282 for Qwen3-Next and GLM-5, \$83 for Opus 5.5, and \$82 for GPT-OSS and MiniMax.

Failures. Cells with no usable submission or no final record score zero, and model outcomes are not retried to obtain a better answer. For agents, the native evaluator scores the final world state independently of the termination status. The retained matrix includes six correct resource-limited cells: two Flash workflow Direct cells, one Flash terminal Direct cell, one Flash terminal Parallel@4 cell, and two Opus terminal Direct cells. Reporting every non-completed status as incorrect would therefore misdescribe the primary scores. Only infrastructure failures—dropped connections, provider server errors, permission errors, and task containers that failed to start—were resumed or re-run, at most twice per task. Fifteen cells never finished and are scored as incorrect: two Flash terminal cells that reached their wall-clock limit, ten GLM-5 mathematics cells whose Bedrock requests hung until the client timeout (about 5% of GLM-5 requests hung; every successful call finished within 1,798 s), and three GPT-OSS cells whose identical request hung or returned a server error on every retry. Six terminal Direct cells whose Harbor container failed to start or whose command exceeded the operating system's argument limit were re-run and now have valid grades; two verifier timeouts count as failures, as Terminal-Bench scores them.

Opus 5.5 on code without numpy. Six of Opus 5.5's seven code failures import numpy, which the frozen grading image does not contain, so these programs fail at import. The prompt states no library restriction, but it offers a tool that runs code in the same sandbox, and Opus 5.5 submitted these six programs without running them; no other model imported numpy. We therefore keep the official score of 0.930. Regrading the six programs in the same image with numpy 2.4.6 installed and `OPENBLAS_NUM_THREADS=1` set makes all six pass, giving 0.990.

Changes after sealing. Four infrastructure changes were made after sealing, and each affected configuration declares it. (i) Terminal tasks run through a native Harbor bridge, including the per-command limit. (ii) Amazon Bedrock's Converse API rejected three agent-loop situations as validation errors, ending runs for interface reasons: a tool-less review call over a transcript with tool turns, a call to a tool name the model invented, and tool calls written as plain JSON text. The first two are now rendered as text, and calls to declared tools written as text are recovered and marked (24% of Qwen3-Next's workflow calls and 11% of its terminal calls). (iii) Prompt caching on Bedrock agent requests. (iv) Output caps of selection stages fitted to the remaining context window for models whose window is smaller than the frozen cap. These changes preserve the sealed tasks and primary

graders, but they affect tool-history serialization, executable tool handling, or the effective per-request cap. Nominal cell allowances do not imply identical realized inputs and stage limits across interface versions. The Qwen3-Next runs on the earlier interface are retained as an implementation baseline, not pooled into the final matrix.

J MODELS

Solver and review stages request high reasoning effort; agent planning and plan selection request low effort. Provider-specific limits and interface settings are part of the reported configuration.

Table 7: Models. Prices are USD per million input and output tokens on our ledger. Revision eligibility uses the configured input headroom plus a full-budget critique, not a universal context threshold for all revision methods.

Model	Access	Price (in / out)	Context	Protocols
DeepSeek V4.1 Flash	DeepSeek API	0.30 / 1.20[a]	1M	all
Qwen3-Next-80B-A3B	Amazon Bedrock	0.15 / 1.20	262,144	all
GLM-5	Amazon Bedrock	1.00 / 3.20	202,752	Direct, Parallel
GPT-OSS-120B	Amazon Bedrock	0.1545 / 0.618	131,072	Direct, Parallel[b]
MiniMax M2.5	Amazon Bedrock	0.30 / 1.20	196,608	Direct, Parallel[b]
Claude Opus 5.5 (anchor)	Amazon Bedrock	4.00 / 20.00	–	Direct

[a]Peak rate; the off-peak rate is about half. [b]Terminal: Direct only, since both models scored 0/30 on terminal Direct and no terminal protocol had improved on Direct for the first three models.

Table 7 lists access, prices, and context windows. Only Flash and Qwen3-Next meet the context-headroom requirement of our full-solution critique configuration; this exclusion does not imply that shorter or differently structured revision is impossible for the other models. DeepSeek V4.1 Flash is accessed through the official API with thinking enabled, reasoning effort high, and top-$p = 0.95$; no temperature or seed is sent. The other models are accessed through Amazon Bedrock (us-east-1; Opus 5.5 through the global endpoint with adaptive thinking at effort high). We accept a single-call output limit only when a generation actually stops at that limit, because provider model cards understated capacity for four of eight candidates during screening. Bedrock caps output per call at 128,000 tokens for Opus 5.5 and GPT-OSS, below the nominal 131,072; affected stages are capped explicitly while total cell allowances remain unchanged. Qwen3-Next's static revision is capped at 131,072 rather than the nominal 262,144, so its three-stage allocation totals at most $3B$ under a $4B$ cell allowance. GPT-OSS's 131,072-token window additionally limits its solution stages to 118,000 tokens, which truncated three mathematics candidates, and its selection stages are fitted to the remaining window. On one terminal task, Bedrock's content filter stopped two of Opus 5.5's turns, ending the episode; it counts as a model outcome.

K OFFLINE DIAGNOSTIC AND STATISTICAL AUDIT

The audit in `analysis/audit_results.py` validates the number of retained records and correct outcomes for all 68 configurations against the frozen result matrix before computing diagnostics. It also checks that each task index agrees with the sealed public manifest. Only retained final runs and declared parts are merged; superseded attempts are excluded. AppWorld records are projected to aggregate status and grade counts without inspecting individual trajectories. The output contains aggregate statistics and source hashes, not task text or model answers.

Dependence and uncertainty. For each domain–protocol panel, the bootstrap resamples task identifiers within their frozen strata, taking all evaluated models' paired outcomes for each sampled task together. Models are fixed rather than sampled from a population. We use 20,000 replicates with seed 20260925 and percentile 95% intervals. Per-model exact sign tests are separately corrected across all 44 evaluated non-Direct comparisons using Holm's procedure (Table 8); the twelve panel intervals are pointwise descriptive intervals and are not multiplicity-adjusted significance tests. The absence of repeated seeds prevents estimating within-task stochastic variability.

Reused outputs and revision transitions. Request caching couples some static protocol outputs, but tool continuations or split runs can produce different initial strings. For example, Flash's coding revision initial output is byte-identical to Direct on 88/100 tasks, although its initial correctness agrees on all 100. Revision transitions are consequently computed from each cell's own saved initial and final output, not inferred from cross-protocol labels. The four static transition rows in Table 9 reproduce the reported revision accuracy changes. The same interpretation does not extend to separately executed agent episodes.

Content changes and missing submissions. The 320 static revision cells contain 68 final failures: 66 initially incorrect submissions remain incorrect, one correct answer becomes an incorrect answer, and one correct initial program is followed by an empty final submission. The content-level loss is Qwen3-Next mathematics: both its initial and revised answers are nonempty and complete. The submission loss is Flash programming: its critique exhausts its 131,072-token stage allowance before revision begins, and the harness discards the correct initial program. Conversely, Flash's two mathematics W→R transitions include one repair of a nonempty wrong answer and one recovery from an initial generation truncated at 131,072 tokens. Thus the operational W→R and R→W counts do not alone identify content-level repair or damage.

Post-hoc sensitivity: retaining the initial answer. We replay a deterministic fallback on all 320 static revision records: if the cell ends with a stage failure, truncation, or exhausted budget, has no final submission, and has a nonempty completed initial answer, return that initial answer. The trigger uses only termination status and output availability; existing hidden grades score the returned answer but never determine whether fallback occurs. Exactly one cell qualifies, the Flash programming case above, raising its accuracy from 98/100 to 99/100, equal to Direct. All other static revision scores are unchanged, and all spent tokens and logical API costs remain charged. This analysis uses saved outputs and grades without new inference or grading; the frozen primary scores remain unchanged. It isolates one avoidable loss from the failure-handling rule and does not establish that revision preserves correctness in general. The replay does not apply to agent episodes, where actions may already have changed the environment.

Candidate-pool replay. Every retained mathematics Parallel@4 record contains three graded candidates: 60 pools each for Flash, Qwen3-Next, and MiniMax, 53 for GLM-5, and 59 for GPT-OSS. We apply the original final-answer parser and voting tie rule to these same strings, then look up the selected candidate's existing grade. The choice rule has no access to correctness labels. This isolates the choice rule on observed pools, while the eight missing records remain failures in the primary 60-task denominators. On programming, 496 records contain all three attempt slots, including empty attempts; two GPT-OSS records are partial and two are missing. Coverage of partial pools is reported only when a correct candidate is observed. No unavailable candidate is imputed correct or incorrect for the conditional conversion calculation.

Table 8: Per-model comparisons with Holm-adjusted $p < 0.05$ across the full family of 44 protocol-versus-Direct tests. The family includes every evaluated non-Direct configuration, not only displayed effects.

Model	Domain	Protocol	Wins–losses	Adjusted p
MiniMax M2.5	Workflow	Parallel@4	0–19	0.0001678
MiniMax M2.5	Workflow	Parallel@2	1–19	0.001722
GPT-OSS-120B	Code	Parallel@4	4–26	0.002498
GLM-5	Code	Parallel@2	15–0	0.002502
DeepSeek V4.1 Flash	Workflow	Revise@4	11–34	0.03296

Table 9: Static Revise@4 transitions from its own recorded initial answer to its final submission. R and W denote correct and incorrect, including missing or ungradable outputs. Recovery and loss are operational rates, not solely content changes.

Model	Domain	R→R	W→R	R→W	W→W	Recovery	Loss
DeepSeek V4.1 Flash	Math	46	2	0	12	2/14	0/46
Qwen3-Next-80B	Math	30	0	1	29	0/29	1/31
DeepSeek V4.1 Flash	Code	98	0	1	1	0/1	1/99
Qwen3-Next-80B	Code	76	0	0	24	0/24	0/76

Flash mathematics has one repair and one recovery from an empty, truncated initial output. Flash code loses a correct initial program after critique truncation, without a revised submission. Qwen mathematics has one correct-to-incorrect rewrite. See Appendix K for fallback sensitivity.

K.1 PER-MODEL DIAGNOSTICS BEHIND SECTION 3.3

Mathematics. Table 10 grades every available three-candidate pool and applies an offline vote to exactly the same candidates used by the model judge. Flash's pools contain a correct answer on 54/60 tasks, and self-selection converts 52 of those opportunities. Qwen3-Next's pools cover 38/60 tasks, exceeding its Direct count of 31, yet its judge submits only 30 correct answers: eight covered pools are lost through an incorrect choice, with no missing choice. On the 292 available pools across the five models, self-selection submits 185 correct answers and the counterfactual vote@3 submits 182, with ten wins and seven losses between them. Flash contributes two additional successes under self-selection (52 versus 50), and Qwen3-Next submits one fewer (30 versus 31). This replay holds the candidate pool fixed and changes only the choice rule, but it does not establish a broadly superior selector: the outcomes vary by model, and the model judge incurs additional inference cost. Missing pools remain failures in the primary accuracy table and are excluded only from this explicitly conditional replay.

Programming. Table 11 gives the corresponding decomposition for programming. GLM-5 has a correct program in 88 pools but submits only 55; 27 covered pools return no usable selection and six select an incorrect program. Flash loses three of its 99 covered pools, all without a usable choice. Qwen3-Next exhibits a different failure profile: seven wrong choices and only one absent choice among its eight lost opportunities. Across the five models, 81 observed covered pools fail to convert: 64 have no valid selection index and 17 select an incorrect candidate, including one empty answer. MiniMax accounts for that empty selected answer: its eight absent indices plus this case explain the nine submissions recovered by nonempty fallback (81 to 90). Of the 61 coding outcomes lost against Direct, 56 have a retained record with no valid selection index, two have no final record, and three select an incorrect nonempty candidate. This does not mean that every failure occurs inside the judge: GPT-OSS also has two partial candidate pools, and a missing record cannot be assigned a selection-stage cause. It does show that the main loss in submitted accuracy cannot be described solely as choosing the wrong program after successful execution of the protocol. The contrast with Parallel@2 is also not a clean selector ablation, because that protocol uses two candidates rather than three; the candidate-level accounting establishes the conversion loss without requiring that causal interpretation.

The corresponding failure partition is shown in Figure 4.

Agents. On workflows, Flash's small gains under plan-level parallelism (78/160 to 82/160 and 81/160) coexist with losses for GLM-5, GPT-OSS, and MiniMax and near-floor performance for Qwen3-Next. For workflow Parallel@2 and Parallel@4, GPT-OSS records 117/160 and 111/160 stage failures, respectively, and MiniMax records 159/160 under either protocol. The recorded aggregate diagnostics associate these failures with absent final planning messages or tool-call markup in stages that require a plan or a selection. These rows measure end-to-end performance of the model–protocol–interface combination, not the quality of a successfully executed multi-plan strategy. Flash's workflow review lowers accuracy from 78/160 to 55/160 with 11 paired wins and 34 losses (Holm-adjusted $p = 0.033$); because these are separate episodes, the 34 losses are not observations of an individual trajectory being repaired into failure. On terminal tasks, Flash remains at 3/30 under either Parallel protocol, while Qwen3-Next and GLM-5 remain at zero; Flash nevertheless exchanges solved tasks with Direct, so unchanged accuracy does not mean that the protocols solve identical instances. Thirteen recorded Flash terminal Direct cells and eight Opus terminal Direct cells terminate at a token or call limit, including cases whose final state is nevertheless correct. Across the entire matrix, six resource-limited agent cells pass the native evaluator (Appendix I); retaining these grades distinguishes task completion from the harness's stopping condition.

Table 10: Mathematics Parallel@4: coverage, self-selection, and counterfactual vote@3 on the identical recorded pools. Every available pool has three graded candidates. Counts in the middle columns use the available-pool denominator; official accuracy always uses all 60 scheduled tasks.

Model	Pools	Coverage	Self-select	Vote@3	Conversion
DeepSeek V4.1 Flash	60	54	52	50	52/54
Qwen3-Next-80B	60	38	30	31	30/38
GLM-5	53	44	41	41	41/44
GPT-OSS-120B	59	42	35	34	35/42
MiniMax M2.5	60	31	27	26	27/31

GLM-5 and GPT-OSS have seven and one missing records, respectively. Voting is recomputed from stored final-answer strings with the existing parser and tie rule; it uses no hidden correctness labels to choose a candidate and is not an independently rerun protocol.

Table 11: Programming Parallel@4: where recorded candidate coverage is lost. Each model has 100 scheduled tasks. O: a correct candidate is observed; S: the final submission is correct. The last two columns partition $O - S$. A missing record is not counted as an observed empty pool.

Model	3 attempts	O	S	No choice	Wrong choice
DeepSeek V4.1 Flash	100	99	96	3	0
Qwen3-Next-80B	100	83	75	1	7
GLM-5	100	88	55	27	6
GPT-OSS-120B	96	94	69	25	0
MiniMax M2.5	100	93	81	8	4

GPT-OSS has 96 pools with all three attempts recorded, two partial pools (both covered), and two missing records. "No choice" means no valid selection index; the selection stage need not have been reached. "Wrong choice" means an indexed candidate grades incorrect, including an empty answer. MiniMax has eight absent indices and one indexed empty answer (within its four wrong choices), so nonempty fallback recovers nine submissions. All other models record three attempts on 100 tasks; an attempt may be empty.

K.2 SAME-POOL SELECTION AND FAILURE-HANDLING REPLAY

We compare choice rules on stored candidate strings, using hidden grades only after selection. This post-hoc analysis preserves the frozen primary matrix; it does not add generation samples, replace missing records, or estimate repeated-generation variability. We retain per-pool choices, public execution evidence, stage costs, and input hashes in the replay records; `analysis/run_selector_replay.py` and `analysis/summarize_selector_replay.py` reproduce execution and aggregation.

Programming rules and execution. We process all 498 retained pools: 496 pools with all three attempt slots recorded and two partial GPT-OSS pools. Two missing GPT-OSS records remain wrong in the denominator of 500 scheduled cells. We compare the frozen judge (J), fallback to the first nonempty candidate when J returns no valid nonempty choice (J+F), public-example fallback under the same condition (J+P), and public-example selection on all three candidates (P3) or only the first two of the same pool (P2). Public selection maximizes examples passed and breaks ties by candidate order; an empty or uncompilable candidate receives a negative score. No hidden grade determines selection or fallback. These pools retain empty or truncated attempts as failures; they do not require three usable programs. As further availability-conditioned checks, the 459 pools with three nonempty answers give 354 correct submissions for J versus 423 for P3. The 419 pools where all three candidates compile give 336 versus 392. These descriptive subsets do not replace the scheduled denominator.

Public execution reuses the frozen grading helper and immutable runtime image with network disabled, read-only containers, bounded memory, and the original compilation and execution limits. The replay makes 3,959 public-test executions in 838 seconds with four workers (3,299 summed worker-seconds), with no environment errors. Every candidate recorded as hidden-correct passes its public examples. No hidden tests are re-executed, and no new API request is made. These runtime measurements report the local execution expense separately from API dollars.

A public-first judge gate. An additional post-hoc rule (G) returns a unique public-score winner directly. For tied nonnegative maxima, it reuses the original judge's choice if that choice is among the maxima, otherwise returns the earliest maximum. If no candidate has a nonnegative score it returns no submission. The rule was recorded before inspecting aggregate public-test results. It neither uses hidden labels to decide whether to judge nor changes the judge prompt: the retained judge saw the original full pool, not newly supplied public scores or a filtered candidate list. This is a conditional replay of that recorded call, not an evaluation of a newly prompted judge. G calls the judge on 465 of 498 retained pools; its choices equal those of always judging with the same public-score guard, but its logical cost omits judging on unique winners.

Accuracy and selection cost. The original index-only fallback recovers 60 correct submissions (436/500). Inspection revealed one further case in which a valid judge index refers to an empty candidate. The output-aware amendment triggers fallback for this public condition as well, giving J+F 437/500 and J+P 440/500, compared with 439/500 for index-only public fallback. This additional rule was specified after that inspection and is reported as post-hoc. J+F therefore recovers 61 submissions at unchanged logical cost, changing the gain against Direct from -8.6 to $+3.6$ pp. P3 yields 453/500 ($+6.8$ pp against Direct), converts 453 of 457 covered pools, and saves 12–49% of logical API cost across models. P2 yields 448/500, so five extra correct submissions are attributable to adding the third candidate under this fixed-pool rule. The independently generated Parallel@2 score of 450/500 is a different comparison. G yields 452/500: in these observations the extra judging adds cost without an accuracy gain over P3.

Selection among mixed candidates. Table 14 separates pools by the correctness of their recorded candidates. Of 498 retained pools, 359 contain only correct candidates, 41 only incorrect candidates, and 98 contain both. The first group includes the two partial pools; it does not require three recorded attempts. In the mixed group, the judge, nonempty fallback, and public-example selection obtain 61, 78, and 94 correct submissions, respectively. Public selection gains 16 choices and loses none against fallback, accounting for its entire additional gain. Thus it improves selection within mixed pools as well as preserving available answers. These groups are defined retrospectively for diagnosis; hidden grades never determine the selection rule.

Table 12: Programming selector replay on the recorded Parallel@4 pools. Counts use 100 scheduled tasks per model. J: frozen judge; J+F: first-nonempty fallback; J+P: public-example fallback (both also handle an empty chosen answer); P3/P2: public-example choice on all three/first two candidates; G: public tests first, judge only tied maxima, with public fallback.

Model	Direct	J	J+F	J+P	P3	G	P2
Flash	99	96	99	99	99	99	99
Qwen3-Next	76	75	76	76	82	81	82
GLM-5	66	55	78	81	87	87	83
GPT-OSS	91	69	94	94	94	94	94
MiniMax	87	81	90	90	91	91	90
Total	419	376	437	440	453	452	448
Choice indices	—	421	498	497	497	497	493

All rules use public information only. Two missing GPT-OSS records remain failures. Its two partial pools are retained where a candidate exists; 496 pools with three recorded attempts are analyzed separately; some attempts are empty. Choice indices count a selected index, not guaranteed correct or nonempty output (one J choice is empty). The index-only fallbacks score 436 and 439; output-aware J+F/J+P also recover one empty choice. Both retain all judge cost; G calls the judge only for tied maxima and guards its choice by public scores; P3/P2 omit judge calls. P2 reuses the first two candidates of these pools, not the independently recorded Parallel@2 runs.

Table 13: Logical API costs of the coding replay (USD per retained task). Fallback retains all judge spending; public-only variants omit it, and G charges the judge only for public-score ties.

Model	J / J+F / J+P	P3	P2	G
Flash	0.0766	0.0391	0.0269	0.0766
Qwen3-Next	0.0800	0.0566	0.0381	0.0792
GLM-5	0.3774	0.3058	0.2096	0.3647
GPT-OSS	0.0809	0.0709	0.0533	0.0793
MiniMax	0.1686	0.1172	0.0750	0.1646

Candidate generation includes its original public-tool interactions. Public-test replay itself makes no model calls; its local execution time is reported separately and is not priced as zero-cost computation. These are accounting counterfactuals on saved pools, not fresh end-to-end budget trials.

Among the 465 pools with tied highest public scores, the tied candidates are all correct in 426 pools, all incorrect in 26, and mixed in 13. Considering every candidate in those same pools instead gives 358 all-correct, 26 all-incorrect, and 81 mixed pools. The distinction matters: lower-scoring incorrect candidates need not remain in the tied set. Public selection submits correctly on 435 of these 465 pools, including nine of the 13 with mixed tied maxima. This explains why frequent public-score ties need not imply frequent consequential ambiguity, and why the tested judge gate adds little value.

All candidate-generation calls, including their original tool interactions, remain charged. Fallback after judging retains every spent judge-stage call; policies that omit judging remove all calls in that stage, including continuations. G charges the recorded judge stage only on ties. Per-call sums reconcile with each original cell's logical cost. These are accounting counterfactuals on fixed pools, not refunds of historical spending or fresh end-to-end trials at matched expenditure.

Mathematics. Vote@3 uses the original parser and earliest-candidate tie rule on all 292 retained complete pools. Original judging obtains 185 correct submissions versus 182 for voting, with ten paired wins and seven losses. Applying the same nonempty-answer fallback to all 292 mathematics pools recovers one MiniMax answer after an invalid judge choice, raising MiniMax from 27 to 28 and the total to 186, at unchanged cost. All other models remain unchanged. Removing all judge-stage calls saves 12.3–21.1% of logical API cost across models (Table 15); generation remains charged. Flash gains two correct answers with judging at an additional $0.032 per task, whereas Qwen3-Next voting is cheaper and one answer more accurate.

Table 14: Programming conversion by recorded candidate correctness. J: frozen judge; J+F: nonempty fallback; P3: public-example selection. Counts exclude the two missing records, which remain failures in scheduled accuracy.

Candidate pool	Pools	J	J+F	P3
All correct	359	315	359	359
Mixed	98	61	78	94
All incorrect	41	0	0	0

The two partial pools are included in the all-correct group. These are descriptive groups, not information supplied to selectors. P3 versus J+F on mixed pools has 16 wins and no losses.

Table 15: Mathematics original judging (J), judging with nonempty fallback (J+F), and vote@3 on identical pools. Costs are mean logical API dollars per retained task at the original price basis; voting removes the entire judge stage.

Model	Pools	J	J+F	Vote	J / J+F $	Vote $	Saving
Flash	60	52	52	50	0.1764	0.1444	18.1%
Qwen3-Next	60	30	30	31	0.1561	0.1369	12.3%
GLM-5	53	41	41	41	0.6183	0.4882	21.1%
GPT-OSS	59	35	35	34	0.0959	0.0758	21.0%
MiniMax	60	27	28	26	0.1511	0.1193	21.0%

Missing pools are excluded from this controlled comparison, while remaining failures in the frozen matrix. Fallback recovers one MiniMax answer (27 to 28), giving 186 correct versus 185 for J and 182 for voting; all judge cost remains charged. Choice rules use no hidden correctness labels.

Mathematics grading sensitivity. We keep the candidates and selected indices fixed and score all 292 retained pools using their saved CompassVerifier-3B candidate judgments (Table 16). The verifier's source hashes and candidate order are checked against the original evaluations; no new generation or grading is performed. The primary rule grader gives 186 correct submissions for judging with fallback and 182 for voting. Compass gives 194 and 189, respectively: 11 paired wins and six losses, or $+1.71$ pp with a pointwise 95% interval of $[-1.04, 4.79]$. Two of the 16 pairs discordant under the primary grader have a selected-answer grading disagreement. The observed direction is preserved, while both graders leave uncertainty about the small selector difference. We retain the frozen primary scores; this is an auxiliary sensitivity analysis, not human adjudication or selection of the more favorable grader. The task-stratified bootstrap uses 20,000 replicates and seed 20260926.

The script `analysis/audit_selector_sensitivity.py` reproduces the candidate-composition and grading-sensitivity analyses from saved records, including per-model counts and source hashes.

Deployment comparison after replay. We add vote@3, mathematics nonempty fallback, and all coding replay variants to the original configurations and recompute nondominance using scheduled-task accuracy and mean logical API cost over retained records (`analysis/audit_replay_deployment.py`). The peak-price mathematics and coding frontiers are unchanged. At half the Flash price, vote@3 adds a frontier point at 50/60 and \$0.0722/task, below judging's 52/60 and \$0.0882; Opus remains at 57/60 and \$0.1035. On code, Flash Direct remains the frontier at either price basis. These point-estimate, API-cost comparisons omit local execution expense and do not establish statistically certified dominance.

Paired uncertainty and scope. We bootstrap tasks within the frozen strata, keeping the five models together (20,000 replicates, seed 20260926). These post-hoc pointwise intervals describe the fixed panel and saved pools; they are not simultaneous significance tests. On all 500 scheduled coding cells, P3 versus J gains 15.4 pp with interval $[12.6, 18.2]$ (77 wins, no losses). Restricting both to the 496 pools with three recorded attempts gives 15.3 pp, $[12.7, 18.1]$ (76 wins, no losses). P3 versus same-pool P2 gains 1.0 pp, $[0.2, 2.0]$ (five wins, no losses); G versus P3 changes -0.2 pp, $[-0.6, 0.0]$ (no wins, one loss). Conditional on the 292 retained mathematics pools, original judging

Table 16: Mathematics selector sensitivity under two grading rules on identical candidates and fixed choices. J: frozen judge; J+F: nonempty fallback; V: vote@3. W–L compares J+F with V.

Grader	Pools	J	J+F	V	W–L
Primary rule	292	185	186	182	10–6
Compass	292	192	194	189	11–6

Compass judgments were saved during the evaluation campaign. This auxiliary comparison does not replace the primary grader or constitute human adjudication.

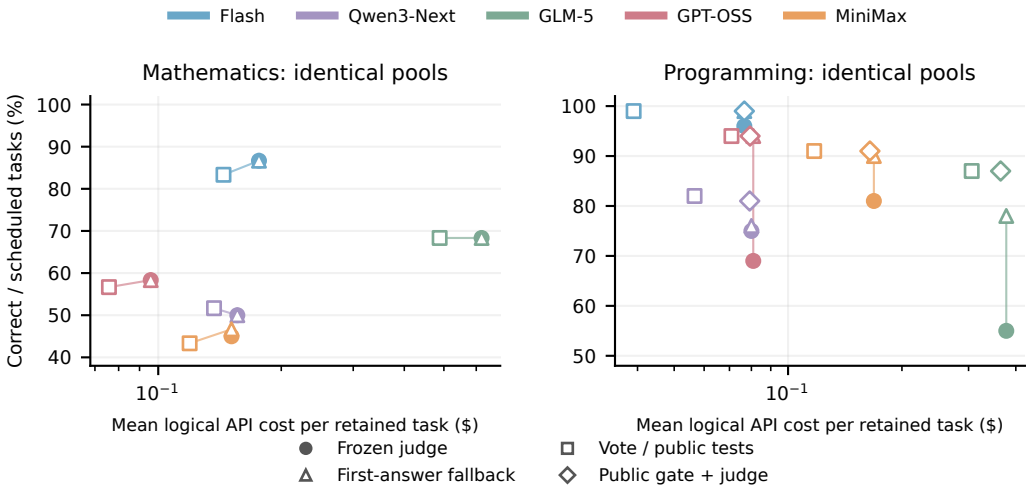


Figure 6: Same-pool choices and logical API cost. Accuracy uses scheduled denominators (60 mathematics or 100 coding tasks per model), retaining missing records as failures; cost averages over retained records. Mathematics voting and programming public selection remove judge-stage calls. Fallback retains already-spent cost. Gate costs assume calling the unchanged judge only on public-score ties. Local execution expense is reported separately.

versus voting gains 1.0 pp, $[-1.7, 3.8]$; judging with nonempty fallback gains 1.4 pp, $[-1.4, 4.1]$ (10 wins, 6 losses). None of these comparisons estimates variability from new generations or repeats the agent environments.

L FULL RESULTS

Table 17: Frozen protocol matrix: accuracy of each cheap model under each protocol, with wins–losses against the same model's Direct on the same tasks (exact two-sided sign test: $^{*}p < 0.05$, $^{**}p < 0.01$, uncorrected) and cost relative to Direct. The rightmost column gives the expensive anchor, Claude Opus 5.5, answering once. n/a: excluded by the configured revision headroom requirement. –: not run. ‡: includes cells that never finished, scored incorrect. §: the plan or plan-selection step failed its required format in at least 69% of tasks, so the row reflects format compliance rather than the value of extra compute.

		Direct		Parallel@2		Parallel@4		Revise@4		Anchor
Domain	Model	Acc	$/task	Acc (W–L)	Cost	Acc (W–L)	Cost	Acc (W–L)	Cost	Opus 5.5
Math (60)	DeepSeek V4.1 Flash	0.767	0.051	0.783 (1–0)	1.9×	0.867 (6–0)*	3.4×	0.800 (2–0)	1.9×	0.950
	GPT-OSS-120B	0.533	0.026	0.533 (0–0)	2.0×	0.583‡ (5–2)	3.7×	n/a	n/a	
	MiniMax M2.5	0.417	0.038	0.417 (0–0)	2.0×	0.450 (3–1)	3.9×	n/a	n/a	
	GLM-5	0.650	0.175	0.667‡ (2–1)	2.0×	0.683‡ (7–5)	3.5×	n/a	n/a	
	Qwen3-Next-80B	0.517	0.048	0.517 (0–0)	1.9×	0.500 (3–4)	3.3×	0.500 (0–1)	1.2×	
Code (100)	DeepSeek V4.1 Flash	0.990	0.013	0.990 (0–0)	2.0×	0.960 (0–3)	5.8×	0.980 (0–1)	1.3×	0.930 (0.990†)
	GPT-OSS-120B	0.910	0.025	0.970 (6–0)*	2.0×	0.690‡ (4–26)**	3.2×	n/a	n/a	
	MiniMax M2.5	0.870	0.039	0.910 (4–0)	1.9×	0.810 (2–8)	4.3×	n/a	n/a	
	GLM-5	0.660	0.103	0.810 (15–0)**	2.1×	0.550 (10–21)	3.7×	n/a	n/a	
	Qwen3-Next-80B	0.760	0.019	0.820 (6–0)*	2.0×	0.750 (2–3)	4.2×	0.760 (0–0)	1.3×	
Workflow (160)	DeepSeek V4.1 Flash	0.487	0.020	0.512 (34–30)	1.0×	0.506 (35–32)	1.3×	0.344 (11–34)**	1.5×	0.925
	GPT-OSS-120B	0.094	0.015	0.019§ (2–14)**	0.2×	0.019§ (3–15)**	0.3×	n/a	n/a	
	MiniMax M2.5	0.119	0.094	0.006§ (1–19)**	<0.1×	0.000§ (0–19)**	<0.1×	n/a	n/a	
	GLM-5	0.138	0.129	0.069 (9–20)	0.6×	0.062 (7–19)*	0.4×	n/a	n/a	
	Qwen3-Next-80B	0.000	0.097	0.019 (3–0)	0.8×	0.019 (3–0)	0.8×	0.000 (0–0)	0.5×	
Terminal (30)	DeepSeek V4.1 Flash	0.100‡	0.307	0.100‡ (3–3)	1.2×	0.100 (2–2)	1.4×	0.000 (0–3)	0.7×	0.433
	GPT-OSS-120B	0.000	0.175	–	–	–	–	n/a	n/a	
	MiniMax M2.5	0.000	0.441	–	–	–	–	n/a	n/a	
	GLM-5	0.000	0.350	0.000 (0–0)	0.6×	0.000 (0–0)	0.6×	n/a	n/a	
	Qwen3-Next-80B	0.000	0.063	0.000 (0–0)	0.9×	0.000 (0–0)	0.7×	0.000 (0–0)	0.6×	

†Opus 5.5 scores 0.990 on code when numpy is available in the grading sandbox; six of its programs import it.
Costs average retained records at configured rates (DeepSeek peak); missing records have no imputed cost.
Holm-adjusted tests are in Table 8.

Table 18: Frozen protocol matrix: accuracy (Acc) over all scheduled tasks and mean logical API cost in USD over retained cell records. Missing records have no imputed cost. Opus 5.5 is the expensive anchor and runs Direct only. Bold marks the best accuracy among the four displayed protocols in each row; "–" means not run; ‡ includes cells that never finished, counted as incorrect.

		Direct		Parallel@2		Parallel@4		Revise@4	
Domain	Model	Acc	$	Acc	$	Acc	$	Acc	$
Math (60)	Opus 5.5	0.950	0.103	–	–	–	–	–	–
	DeepSeek V4.1 Flash	0.767	0.051	0.783	0.098	**0.867**	0.176	0.800	0.096
	GPT-OSS-120B	0.533	0.026	0.533	0.052	**0.583**‡	0.096	–	–
	MiniMax M2.5	0.417	0.038	0.417	0.078	**0.450**	0.151	–	–
	GLM-5	0.650	0.175	0.667‡	0.347	**0.683**‡	0.618	–	–
	Qwen3-Next-80B	**0.517**	0.048	**0.517**	0.091	0.500	0.156	0.500	0.059
Code (100)	Opus 5.5	0.930	0.028	–	–	–	–	–	–
	DeepSeek V4.1 Flash	**0.990**	0.013	**0.990**	0.027	0.960	0.077	0.980	0.017
	GPT-OSS-120B	0.910	0.025	**0.970**	0.052	0.690‡	0.081	–	–
	MiniMax M2.5	0.870	0.039	**0.910**	0.075	0.810	0.169	–	–
	GLM-5	0.660	0.103	**0.810**	0.212	0.550	0.377	–	–
	Qwen3-Next-80B	0.760	0.019	**0.820**	0.038	0.750	0.080	0.760	0.025
Workflow (160)	Opus 5.5	0.925	0.097	–	–	–	–	–	–
	DeepSeek V4.1 Flash	0.487	0.020	**0.512**	0.020	0.506	0.026	0.344	0.030
	GPT-OSS-120B	**0.094**	0.015	0.019	0.003	0.019	0.004	–	–
	MiniMax M2.5	**0.119**	0.094	0.006	0.001	0.000	0.002	–	–
	GLM-5	**0.138**	0.129	0.069	0.074	0.062	0.055	–	–
	Qwen3-Next-80B	0.000	0.097	**0.019**	0.078	**0.019**	0.075	0.000	0.050
Terminal (30)	Opus 5.5	0.433	1.930	–	–	–	–	–	–
	DeepSeek V4.1 Flash	**0.100**‡	0.307	**0.100**‡	0.363	**0.100**	0.438	0.000	0.204
	GPT-OSS-120B	0.000	0.175	–	–	–	–	–	–
	MiniMax M2.5	0.000	0.441	–	–	–	–	–	–
	GLM-5	0.000	0.350	0.000	0.195	0.000	0.224	–	–
	Qwen3-Next-80B	0.000	0.063	0.000	0.056	0.000	0.046	0.000	0.039