WHAT EFFORT LEVEL SHOULD I CHOOSE? ON THE ECONOMICS OF REASONING EFFORT IN LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative models increasingly expose a *reasoning effort* setting that lets users scale inference-time compute, trading higher cost for potentially improved performance. Users must choose an effort level before knowing whether extra reasoning will help, and even a successful run does not reveal whether the effort purchased was necessary. Reasoning effort is therefore a *credence good*: buyers observe the outcome but cannot assess the marginal value of the computation they paid for. In this work, we measure the counterfactual effect of effort levels on both proprietary and open-weight models on four benchmarks, spanning single-call reasoning and long-horizon agentic tasks. *Effort*, it turns out, does not consistently preserve tasks that lower effort solves, can overspend, and can change trajectory behavior. First, we find that although higher effort can raise accuracy in *aggregate*, it frequently fails on tasks that lower effort solved. Second, on the subset of easy tasks that every effort level solves, higher effort is generally more costly, with the highest effort level costing up to 42× as much as the lowest. Lastly, *effort* changes not only how many tokens are used, but also *how* agents solve tasks: at higher effort, agents verify their own work more heavily and more often search for solutions than work through problems directly, both of which lengthen trajectories without necessarily altering the outcome. These findings show that naively choosing the highest level is not only cost-inefficient but also offers no guarantee of the best instance-level performance. A simple alternative, starting at the lowest effort and escalating only after a verified failure, matches or improves on fixed top effort across all models and benchmarks at lower cost for most proprietary models.

1 INTRODUCTION

As large language models (LLMs) are deployed more widely and applied to long-horizon agentic tasks, inference cost has become a central concern. Many reasoning models now expose an *effort* parameter that adjusts inference-time computation without changing the underlying model (OpenAI, 2026b; Anthropic, 2026b). This effort parameter scales *test-time compute*, offering better performance on hard problems without switching to a larger model or fine-tuning for the task (Snell et al., 2025; Muennighoff et al., 2025). In agentic settings, effort also changes how the model acts, including how much it plans and how many tools it calls (Anthropic, 2026b). Because providers bill for input, output, and reasoning tokens alike, cost often rises with effort.

However, users must select an effort level without knowing whether the additional computation will help on their task. This uncertainty makes reasoning effort a *credence good*: buyers cannot readily assess how much of a service they need, even after observing the outcome (Darby & Karni, 1973; Dulleck & Kerschbamer, 2006). A successful run shows that the task was solved, but not that the purchased effort was necessary, and rerunning at another level to find out is rarely worth the cost. Users therefore pay for reasoning whose contribution they cannot observe.

In this paper, we study the following counterfactual question: had the user chosen a different, typically lower, effort level, how would the outcome and its cost have changed? Answering this matters because effort is now the main lever users have over inference spend: as agents run for hundreds of calls and reprocess growing contexts, a single effort setting is multiplied across every step of every

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.

trajectory, and it is selected once, by intuition, with no feedback on whether it was needed. We measure the performance, cost, and behavioral consequences of reasoning effort for proprietary and open-weight models on four benchmarks. Within each model–benchmark configuration, we vary effort while holding tasks, prompts, and the agentic harness fixed where applicable. Higher effort does not dominate lower effort at the task level, yet it often leads to higher costs. We characterize this mismatch from four angles and show how allocating effort per task reduces it:

1. **Effort is not a performance ladder (§3.1).** Higher effort often solves more tasks in aggregate, but it may lose on tasks that lower effort can solve.
2. **Higher effort overspends on tasks that lower effort already solves (§3.2).** On tasks solvable at every effort level, models' mean costs reach up to $42\times$ their `lowest`-effort cost, driven by both more computation per call and longer trajectories that repeatedly reprocess a growing context. Disabling reasoning does not reliably reduce cost either, because agents compensate with longer trajectories.
3. **Different efforts produce different solution strategies (§3.3).** Higher effort shifts agents toward heavier self-verification and shortcut behaviors such as retrieving existing answers, which are not suitable for all tasks.
4. **Effort should be allocated per task (§3.4).** Models cannot reliably choose effort from the task description alone, but *naïve laddering*, which escalates from the lowest to the highest effort only after a failed correctness check, matches or improves the success of fixed top effort across all models and benchmarks, while maintaining a lower median inference cost when considering the user waiting time.

Taken together, higher reasoning effort is not necessarily a reliable purchase: its benefits depend on the task, while its costs rise regardless, a pattern that persists across repeated runs. Neither maximal effort nor disabled reasoning is a safe default, so effort should be dynamic rather than fixed in advance, and evaluations that count accuracy and tokens alone may misjudge which configuration is actually better. As long as users can observe neither the reasoning content they receive nor whether it mattered, they remain credence-good buyers. We argue that reasoning charges should be verifiable, so that paying for more thought becomes an informed choice rather than an act of faith.

2 Experimental setup

We evaluate how reasoning effort affects task success, complete execution cost, and agent behavior across two single-call LLM benchmarks and two agentic benchmarks. Within each model–benchmark configuration, we vary the reasoning-effort setting while holding the tasks, prompts, and, where applicable, the agentic harness fixed. This design allows us to compare effort settings within a configuration and examine how their effects differ across models and execution environments.

Benchmarks and execution environments. AIME 2026 (Dekoninck et al., 2026) evaluates mathematical problem solving, while GPQA Diamond (Rein et al., 2023) evaluates graduate-level science question answering. Both use a single inference call per attempt. Terminal-Bench v2.1 (Merrill et al., 2026) evaluates agents on tasks in a terminal environment using Terminus-2, and SWE-Bench Pro (Deng et al., 2025) evaluates repository-level software engineering using `mini-swe-agent`. These agentic benchmarks involve multi-turn interactions, in which effort can affect tool use, verification, and stopping behavior as well as computation within each call. Processing the accumulated interaction history also makes input-token consumption sensitive to trajectory length. Since the original SWE-Bench Pro containers contain the repository history that can expose the reference fix, we evaluate a corrected variant that removes git history and restricts network access to allowlisted package registries (§3.3). Appendix A summarizes evaluation sizes and execution settings in Table 1.

Models and effort levels. We study proprietary models GPT-5.6-sol, GPT-5.6-terra, GPT-5.6-luna (OpenAI, 2026a), and Claude Sonnet 5 (Anthropic, 2026a), alongside the open-weight models GLM-5.2 (Z.ai, 2026), Inkling (Thinking Machines Lab, 2026), Kimi K3 (Team et al., 2026)[1], and

[1]Kimi K3 is evaluated on AIME and GPQA only, due to budget constraints; the agentic results therefore cover seven models.

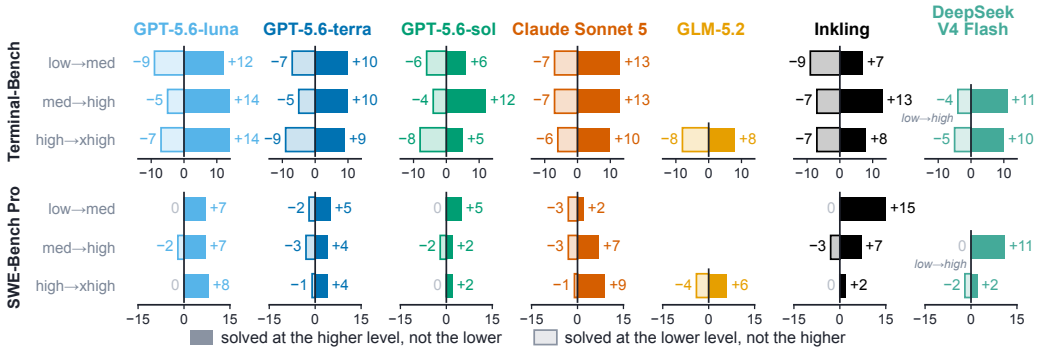


Figure 1: Instances gained and lost between adjacent reasoning-effort settings on the agentic benchmarks. Higher effort can solve additional instances while losing some lower-effort successes. AIME and GPQA are included in Appendix Figure 7.

DeepSeek-V4-Flash-0731 (DeepSeek-AI et al., 2026). Each model has different official reasoning effort settings, as listed in Table 2. We query the proprietary models through their providers' APIs and the open-weight models primarily through Together AI, set reasoning effort through each provider's native control, and leave sampling parameters at the provider or harness defaults (Appendix B).

Measurement and cost accounting. We record task outcomes and capture token usage for every inference call, aggregating usage over the complete execution for agentic tasks. For AIME and GPQA, we run k independent attempts per task at each effort level, with $k = 10$ for AIME and $k = 4$ for GPQA, matching the commonly reported settings. We use pass@k to determine whether a task is solved and mean@k to report aggregate performance. Due to budget constraints, each agentic task is executed once per effort level.[2] To provide additional evidence that quantifies the uncertainty introduced by this single-run protocol, we analyze four independent runs' trajectories on DeepSWE (Huang et al., 2026) in Appendix J. Cost accounting includes input and output tokens, including billed reasoning tokens. We convert usage to dollars using a rate table published by each model provider. Keeping token measurements separate from monetary estimates allows costs to be computed with the most recent pricing data available at the time of submission.

3 Findings

Reasoning effort is exposed as a single ordered dial, with the implicit assumption that by increasing test-time compute, performance will scale accordingly. Here, we test this assumption from the perspectives of accuracy, cost efficiency, and behavior. At a high level, we find that higher effort does not preserve lower-effort successes (§3.1), higher effort overspends on tasks that lower effort already solves (§3.2), and higher efforts change how agents solve tasks rather than simply deliberating longer (§3.3). Together, these findings illustrate the limitations of fixed effort levels and motivate future work on dynamically allocating effort by task (§3.4).

3.1 Model Effort Is Not a Performance Ladder

Reasoning effort settings are presented to users as an ordered sequence, typically spanning `None`, `low`, `medium`, `high`, `extra high`, and `max`. Let s_e denote the set of instances a model solves at effort level e, and let $e_1 < e_2$ order two such levels. One might read this ordering as that raising effort only ever *adds* solutions: the solved sets are nested, $s_{e_1} \subseteq s_{e_2}$, and in particular $|s_{e_1}| \leq |s_{e_2}|$. Higher effort often does solve more instances, so $|s_{e_1}| < |s_{e_2}|$ frequently holds. The nesting, however, does not always hold: models **fail** on tasks they had solved at lower effort,

[2]This is a limitation we accept to cover seven models and five effort levels. While no single run is conclusive on its own, the regressions, cost multipliers, and behavior tend to recur across every model and benchmark.

so the *regression set* $s_{e_1} \setminus s_{e_2}$ is nonempty, and we call these observed reversals "regressions", not established decreases in task-specific success probabilities.

Take Inkling on GPQA as an example. Raising effort from `low` to `medium` gives a net gain of $|s_{\texttt{medium}}| - |s_{\texttt{low}}| = 16$ questions. But this figure is the difference between two **disjoint** sets, and it hides the regression set $s_{\texttt{low}} \setminus s_{\texttt{medium}}$. It turns out that at `medium`, Inkling failed all four attempts on five questions that `low` had solved, $|s_{\texttt{low}} \setminus s_{\texttt{medium}}| = 5$. The aggregated gain (+16 tasks), which is frequently what is reported in the evaluation benchmarks, obscures the questions that the higher level fails to solve (Appendix Figure 7).

Such regressions, although less apparent on non-agentic benchmarks, appear for every model on both agentic benchmarks (Figure 1). On Terminal-Bench, every one of the seven models we evaluated has a nonempty regression set between its lowest and highest effort levels. For GPT-5.6-sol, moving from `high` to `xhigh` loses eight tasks while newly solving only five, so the solved count falls from 76 to 73 of 89 tasks. Aggregate counts $|s_e|$ can therefore make higher effort look like a strict improvement while hiding the lower-effort successes it gives up in $s_{e_1} \setminus s_{e_2}$. In fact, we find that no effort level's solved set contains another's in our experiments. [3] In expectation, higher effort helps, but no user receives the expectation. A user receives one run, and on one run the higher setting's solved set is not a superset of the lower's; the user cannot tell which successes were forfeited (§3.4).

3.2 Higher effort overspends on tasks that lower effort already solves

Beyond regressing on tasks solved at a lower effort level, models exhibit a separate inefficiency: higher-effort executions can cost more than successful lower-effort executions of the same task. One might hope that a model told to spend heavy effort on an easy problem would solve it quickly and stop once there was nothing left to deliberate over. Unfortunately, we find that language model agents show no such tendency. Under higher effort, token usage on easy instances does not contract toward the lower-effort cost; instead, higher effort consumes far more tokens on tasks that a lower level had already solved more cheaply. We observe that *effort* behaves more like a fixed multiplier on computation, rather than an upper-bound budget that can be calibrated to task difficulty.

To isolate this effect, for each model and benchmark, we restrict the comparison to the intersection of tasks solved at every effort level, s_{all}. On AIME and GPQA, membership in s_{all} uses pass@k, while each task's cost is averaged over all k attempts at that effort level. Conditioning on this always-solved subset compares spending across effort levels without conflating cost differences with gains from solving additional tasks. On s_{all}, **higher reasoning effort substantially increases inference cost**. On Terminal-Bench, increasing effort from `low` to `xhigh` raises mean cost by factors of 2.51 for GPT-5.6-sol and 3.04 for Claude Sonnet 5. On SWE-Bench Pro, GPT-5.6-sol's mean cost rises from \$0.27 to \$1.24 over the always-solved tasks ($4.54\times$), while GPT-5.6-luna's rises by $5.34\times$. The increase is not confined to reasoning tokens: for GPT-5.6-sol on SWE-Bench Pro, mean input-token charges rise from \$0.17 to \$0.60 as mean trajectory length increases from 14.7 to 28.4 model calls. Additional effort therefore raises cost through two mechanisms: more computation within individual calls and longer trajectories that repeatedly process an expanding interaction history. Figure 2 reports mean inference cost relative to the lowest reasoning effort available for each model (`low`, or `high` for GLM-5.2, whose reasoning-enabled settings begin at that level).

Models configured with high effort do not recover the efficient route they took at lower effort, and spending scales with effort even on tasks that every level solves. On these tasks, a low effort is sufficient: higher effort buys no additional success, and disabling reasoning shifts the cost into longer trajectories.

"None" effort is not zero additional cost. Our results also highlight that using effort None is not a silver bullet that would eliminate costs. We again use s_{all}, which includes tasks solved at all effort levels, including when the level is set to `None`. The label of `None` reasoning suggests that it might be among the cheapest settings, as it eliminates the additional test-time compute. However, to our surprise, we find that although `None` may not generate more reasoning tokens, it can still spend

[3] Because each agentic task runs once, some reversals reflect sampling noise. Our four-rollout ablation study (Appendix J) shows that such behavior persists at rates comparable to chance, so we report them as observed reversals rather than systematic decreases.

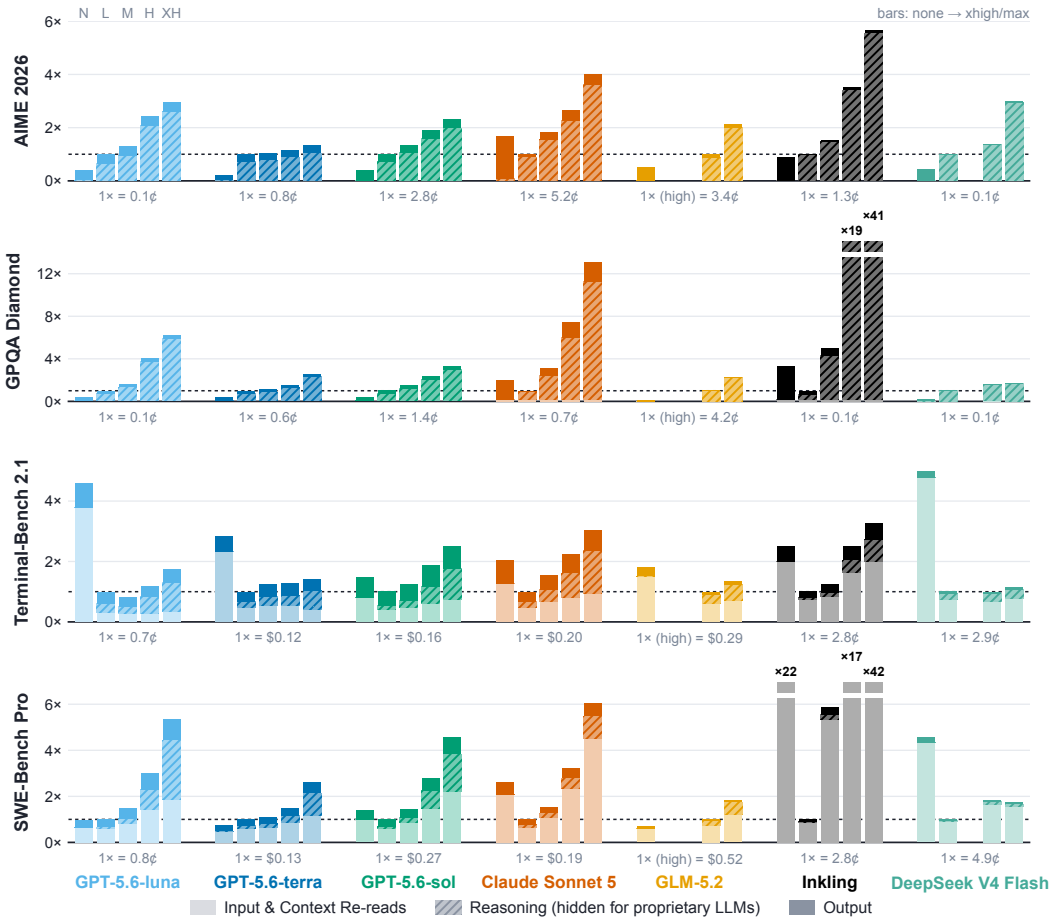


Figure 2: API cost of one task the model solves at every effort level, from `none` to the highest level, relative to the cost at `low` (dashed line; GLM-5.2 is anchored on `high`, its lowest reasoning level). Each bar is split into input, hidden reasoning, and visible output tokens. The always-solved set is fixed per model and benchmark, so bars compare prices within a group and only vary across groups; the footer gives the dollar value of $1\times$. Bars taller than the axis are clipped and labelled.

more money and time than all the other effort levels through increased input-token consumption. On s_{all}, GPT-5.6-luna averages 21.5 model calls at `none`, compared with 5.4 at `low`, and costs $4.58\times$ as much. At `None`, the input-token bill increases from \$0.0021 to \$0.0258, outweighing the removal of reasoning-token charges. Spending less on reasoning within each call can thus be offset by spending more on the trajectory as a whole.

3.3 Different efforts produce different solution strategies

The regressions in §3.1 and the overspending in §3.2 suggest that effort levels do not simply run the same procedure for longer. We examine fine-grained agent trajectories to study what changes. Higher effort shifts agents toward retrieving existing answers and toward heavier self-verification, and neither shift reliably improves outcomes.

Higher effort retrieves reference fixes more often. In the original SWE-Bench Pro environment, the repository history can expose the gold commit that solves the task, allowing an agent to retrieve an existing solution rather than derive one from the task description and code. GPT-5.6-sol reads the reference-fix commit on 15.9% of tasks at `low`, compared with 75.0% at `xhigh` (Figure 3). Part of the apparent improvement with effort may reflect benchmark leakage rather than genuine problem solving.

To separate this retrieval strategy from problem solving, we follow Jain (2026) and create *SWE-Bench Pro-Corrected*, which removes the task repository's original git history while preserving the

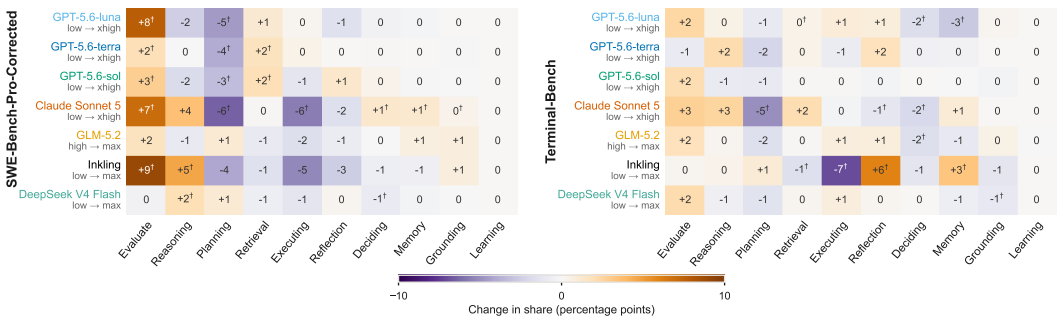
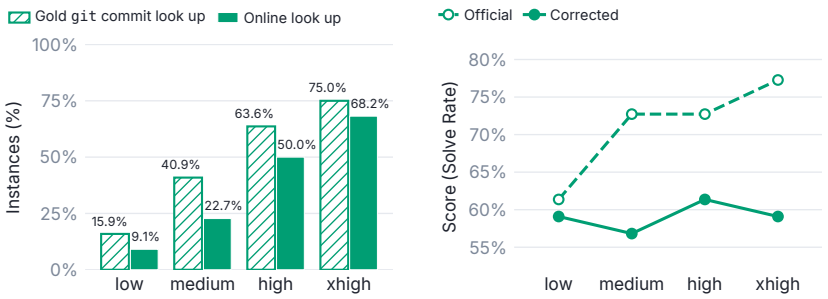


Figure 4: Changes in the mean per-trajectory share of annotated prose assigned to each Act·ONOMY behavior group, in percentage points, from the lowest to the highest annotated reasoning-enabled setting on matched tasks. A dagger (†) marks a change whose unadjusted 90% paired task-bootstrap confidence interval excludes zero (10,000 resamples); markers use unrounded estimates.

starting file tree and working-state changes. We also restrict network access to allowlisted package registries to limit retrieval of upstream fixes, while retaining the same tasks and evaluator. In this corrected environment, GPT-5.6-sol still attempts online answer retrieval on 9.1% of tasks at `low` and 68.2% at `xhigh` even though the restrictions reject these attempts. At `xhigh`, GPT-5.6-sol's solve rate falls from 77.3% in the original environment to 59.1% in the corrected environment. All seven models have lower solve rates in the corrected environment at their highest effort setting (sign test across models, $p = 0.016$), and the decrease is individually significant for four of them (Appendix Table 5). Lower and higher effort levels exhibit different solution strategies, and the high-effort could benefit from a strategy that low effort rarely uses.



(a) Reference-solution lookup

(b) Performance

Figure 3: GPT-5.6-sol on matched SWE-Bench Pro instances across reasoning effort levels. Left: reference-fix reads from official git history and online lookup attempts in the corrected environment, regardless of retrieval success. Right: instances resolved. Corrected removes the original git history and restricts network access to allowlisted package registries.

Higher effort often leads to more verification behavior. We annotate the agent trajectories with the Act·ONOMY taxonomy (Gao et al., 2026), a taxonomy of runtime agent behavior developed through grounded theory, to systematically examine what additional effort buys and inspect the corresponding actions and evaluator outputs. It labels steps of a trajectory (e.g., a tool call or a statement in the model's output) with a category from a three-level hierarchy of 10 top-level actions (e.g., *Evaluation*, *Learning*), 46 subactions, and 120 fine-grained leaf categories. We used Claude Sonnet 5 as the annotator model under the setting of Gao et al. (2026). We compare the lowest and highest annotated reasoning-enabled settings on matched tasks within each model and benchmark, yielding 408 model–task pairs.

Figure 4 shows the mean percentage change across ten behavior groups when switching between effort levels, with evaluation actions showing the largest change. To examine this in more detail, we count annotations in its three leaf categories: *Plan verification step*, *Run verification command*, and *Confirm actual == expected*, which measure evaluation/verification-related behavior among the Act·ONOMY categories. On corrected SWE-Bench Pro, all seven paired models show higher mean verification counts both on tasks passed at both endpoints and on tasks failed at both endpoints. On Terminal-Bench, Claude Sonnet 5's mean counts increase from 6.6 to 12.3 on tasks passed at both endpoints and from 4.3 to 5.5 on tasks failed at both endpoints. Higher effort thus devotes more

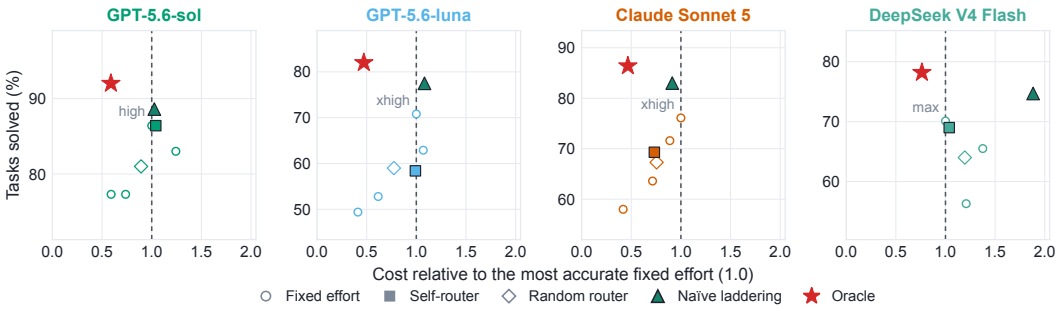


Figure 5: Self-routing and naïve laddering compared with fixed efforts, random allocation, and the oracle on Terminal-Bench 2.1; Appendix I shows the other models with saved self-routing comparisons. Each panel shows success against total API cost, normalized by the cost of that model's fixed effort with the highest performance (dashed line, effort level noted in gray text). Naïve laddering (green triangles) starts at the lowest available reasoning effort and retries at the highest effort after an externally verified failure, charging both executions when it retries. All policies in each panel use the same task cohort. Lower cost and higher success are preferable.

attention to verification, but the increases in both outcome groups show that this additional work need not change the final result.

Self-verification can miss what the external evaluator checks. Individual trajectories show how additional checking can coexist with the regressions in §3.1. In Terminal-Bench's `dna-insert`, GPT-5.6-sol's highest-effort run reports that its output satisfies a numerical acceptance constraint, but the evaluation verifier rejects that constraint, and the low-effort run passes. For GLM-5.2 on `sanitize-git-repo`, the maximum-effort run contains more verification-related annotations but also rewrites repository history, removing a commit required by the external preservation check, while its lower-effort run passes. More verification does not guarantee that the final artifact satisfies the evaluator's requirements.

Effort levels therefore differ in behavior and solution strategy, not merely in the amount of computation. Higher effort favors retrieval and extensive self-checking, while lower effort often reaches a passing solution directly and cheaply. Neither strategy dominates across tasks, so neither a low nor a high fixed setting is uniformly best.

3.4 Effort should be allocated per task

Taken together, the three findings suggest that a single fixed effort level is not optimal for LLM agents. Higher effort, although it usually yields higher aggregate performance, loses tasks that lower effort solves (§3.1), overspends where lower effort suffices (§3.2), and pursues strategies whose value depends on the task (§3.3). A policy that selects effort per task could keep the successes of every level while paying for high effort only where it helps. We quantify this opportunity with an oracle router, then evaluate two deployable policies: self-routing and naïve laddering (Figure 5). The oracle router selects the cheapest recorded successful effort for each task, or the cheapest recorded effort when every setting fails. It uses outcomes and costs known only after execution and is therefore a theoretical bound rather than a deployable policy.

Self-routing: selecting effort before execution remains difficult. We first let the model choose an effort level from the task description on its own, without correctness feedback from an initial or previous attempt. Self-routing offers no consistent cost–accuracy advantage over random allocation or naïve laddering, described in the following paragraph.

Naïve laddering: outcome feedback as a stopgap. Naïve laddering starts at the lowest available reasoning effort and retries at the highest effort only after a failed correctness check, preserving initial successes and charging both attempts when escalation occurs (Appendix H). Because it keeps every initial success, laddering recovers the lower-effort successes that fixed high effort loses and avoids high-effort spending on tasks that lower effort already solves.

We find that naïve laddering matches or improves success over fixed top effort across all evaluated models and benchmarks, while reducing inference cost for every model on AIME and GPQA and six out of seven models on the corrected SWE-Bench Pro. Relative to always using each model's highest available effort setting, median inference-cost savings across models are 25.3% on AIME, 41.5% on GPQA, and 12.3% on corrected SWE-Bench Pro, including the cost of both attempts when laddering retries. On Terminal-Bench, all seven models improve success, with cost reductions for GPT-5.6-sol, GPT-5.6-terra, and Claude Sonnet 5 (Figure 5); the remaining four models gain success at increased cost. For GPT-5.6-sol, success rises from 83.0% to 88.6% while cost falls by 17.4%; for Claude Sonnet 5, success rises from 76.1% to 83.0% while cost falls by 8.7%. Laddering is nonetheless a stopgap rather than a solution. It requires a reliable external correctness check, which many deployments lack, and it pays for a full second execution on every escalation. Its value depends on the quality of that feedback and the complete cost of retries, rather than on an assumption that higher effort is always worth buying in advance.

Counting user waiting time, laddering saves more. The API bill captures inference charges but does not capture the opportunity cost of waiting while the model generates responses. Because reasoning effort can affect both inference charges and generation time, the API bill alone may misrepresent the relative cost of different effort levels from a user's perspective. To account for this additional cost, we define the estimated total cost $C_{\text{full}}(i, e; m)$ for task i, effort level e, and model m:

$$C_{\text{full}}(i, e; m) := C_{\text{API}}(i, e; m) + \frac{w\, N_{\text{out}}(i, e; m)}{3600\, r_m}, \tag{1}$$

where N_{out} counts all generated tokens, including reasoning tokens, and r_m is the model's output throughput in tokens per second. The term $w = \$33.54$ is the mean U.S. hourly wage (Appendix F).[4]

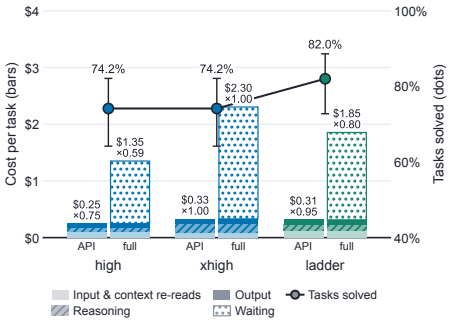


Figure 6: GPT-5.6-terra on Terminal-Bench 2.1 at fixed `high`, fixed `xhigh`, and naïve laddering (green). Bars show mean cost per task (left axis) as the API bill and as full cost (Equation 1), labelled with their ratio to `xhigh`; dots show tasks solved (right axis) with 95% Wilson intervals.

Our estimated total costs make the cost advantage of naïve laddering substantially more pronounced. For instance, on Terminal-Bench with GPT-5.6-terra, the naïve laddering costs 19.5% less than fixed `xhigh` in total cost $C_{\text{full}}(i, e; m)$, compared with only 4.8% less when considering API bill alone. One reason is that API bills do not directly track latency. Naïve laddering may incur additional API cost because caching is not shared across models. In other words, it can still generate fewer tokens overall, thereby reducing waiting time. Because waiting time accounts for over 80% of the total cost at every setting in Figure 6, this latency reduction can translate into a much larger cost advantage than the API bill alone suggests. More broadly, waiting time is the dominant component of our estimated cost of running LLM agents more generally (Appendix F). This makes latency-aware cost accounting particularly important for agentic systems, where long-running trajectories can make waiting costs substantially larger than what API bill alone captures.

4 Related Work

Test-time scaling and reasoning effort. Allocating more computation at inference time improves performance on hard problems (Snell et al., 2025; Muennighoff et al., 2025), and providers now expose this as an ordinal reasoning effort setting (OpenAI, 2026b; Anthropic, 2026b). Their implementation varies across models and sometimes is only partially disclosed. gpt-oss is trained to follow the effort keywords in the system prompt (Agarwal et al., 2025); Qwen3 enforces thinking budgets by interrupting reasoning at inference time (Yang et al., 2025); DeepSeek-V4 distills specialists trained with different length penalties and context windows (DeepSeek-AI et al., 2026); Kimi K3 penalizes trajectories exceeding per-problem token budgets, including reasoning and tool-call arguments in agentic tasks (Team et al., 2026); and Inkling conditions its reinforcement learning

[4]We use the U.S. all-occupations mean rather than a developer wage to avoid assuming who the user is. Lower wages elsewhere would shrink the waiting term but do not change the direction of the comparisons.

stage on a continuous effort value with a per-token cost (Thinking Machines Lab, 2026). Related methods train models to satisfy explicit length constraints (Aggarwal & Welleck, 2025). The heterogeneous mechanisms bring two effects: First, reasoning effort, although identically named across settings, arises from different underlying mechanisms. Second, because the disclosed recipes induce effort largely through length-dependent rewards, nothing in training guarantees that a higher setting dominates a lower one on individual instances. Prior work documents overthinking on easy problems (Chen et al., 2025) and in agentic software engineering (Cuadron et al., 2025). We examine the performance, cost, and behavioral consequences of different reasoning efforts.

Cost-aware evaluation and token economics. Kapoor et al. (2024) advocate cost-controlled agent evaluation using accuracy–cost Pareto frontiers. The Holistic Agent Leaderboard finds that higher reasoning effort reduced accuracy in the majority of its runs (Kapoor et al., 2026), and Erol et al. (2026) measure the expected cost of a correct solution relative to a human-expert baseline. In agentic coding, input tokens rather than output tokens dominate consumption (Bai et al., 2026); failed attempts are also more expensive than successful ones (Fan et al., 2025). Listed token prices can also misrank realized costs because reasoning volume and interaction length differ across models (Chen et al., 2026). Because reasoning tokens are hidden yet billed, pay-per-token pricing creates a moral hazard that users cannot audit (Velasco et al., 2025; Sun et al., 2025).

Prior work either reports aggregate accuracy across effort levels (Kapoor et al., 2026), compares models at a fixed effort (Chen et al., 2026; Bai et al., 2026), or studies a single model on a single-call benchmark (Moon, 2026). We audit the marginal value of reasoning effort at the task level. Our contribution brings these strands together in a task-level audit of the marginal value of reasoning effort. We analyze agent trajectories across effort levels to uncover how reasoning effort reshapes problem-solving strategies, linking these behavioral changes to task outcomes, inference costs, and modeled waiting costs. This sharpens the credence-good interpretation of reasoning: even accurately billed computation may be unnecessary, and observing a successful outcome alone does not establish that the additional effort was worth purchasing.

5 Conclusion

Reasoning effort is presented to users as an ordered dial, but it does not behave as one. Across four benchmarks and eight models, we find that higher effort often raises aggregate success yet can still lose tasks that lower effort solves, and on simple tasks solved at every level, higher effort yields much higher inference cost. Disabling reasoning offers no reliable savings, because agents compensate with longer trajectories that repeatedly reprocess a growing context. In agentic settings, higher effort can bring fundamental change to the solution strategy rather than merely extending deliberation, often leading to more self-verifications that can sometimes be redundant.

For practitioners, these findings suggest that a fixed effort level is not a cost-effective approach. A higher effort level should be justified by evidence on the target task rather than by the intuition that more thinking helps, and disabling reasoning should not be assumed to save money in agentic settings. Where a reliable correctness check exists, naïve laddering, which starts at the lowest effort and escalates only after failure, matches or improves on fixed top effort for every model and benchmark we evaluate, at lower median cost on AIME, GPQA, and corrected SWE-Bench Pro. Practitioners should also evaluate configurations by their full cost, including user waiting time, because token charges alone can misrank them.

More broadly, reasoning effort remains a credence good: users observe neither how much reasoning they receive nor whether it contributed to the outcome. Providers should make reasoning charges verifiable, guiding customers' decisions, and specify what each effort level controls, in token budget, trajectory length, and behavior, so that effort becomes a controlled quantity rather than a named tier. The gap between simple effort routing solutions and the oracle router shows that selecting the optimal effort level for each task remains an open problem, since models cannot yet select effort from the task description and laddering depends on external feedback that many deployments lack. To close this gap, future work can focus on methods that adapt reasoning effort dynamically during execution, spending additional computation only when the difficulty an agent encounters warrants it.

USE OF AI TOOLS

We used LLMs to help write and edit code, improve the clarity of the text, and give critical feedback on our ideas and claims, acting as a simulated reviewer. All AI-assisted content was reviewed by the authors.

REFERENCES

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

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

Anthropic. Claude Sonnet 5 System Card, June 2026a. URL https://www.anthropic.com/claude-sonnet-5-system-card.

Anthropic. Effort. https://platform.claude.com/docs/en/build-with-claude/effort, 2026b. Claude API documentation. Accessed September 16, 2026.

Longju Bai, Zhemin Huang, Xingyao Wang, Jiao Sun, Rada Mihalcea, Erik Brynjolfsson, Alex Pentland, and Jiaxin Pei. How do ai agents spend your money? analyzing and predicting token consumption in agentic coding tasks. *arXiv preprint arXiv:2604.22750*, 2026.

Lingjiao Chen, Chi Zhang, Yeye He, Ion Stoica, Matei Zaharia, and James Zou. The price reversal phenomenon: When cheaper reasoning models cost more. *arXiv preprint arXiv:2603.23971*, 2026.

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.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 9487–9499. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/chen25bx.html.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, et al. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *arXiv preprint arXiv:2502.08235*, 2025.

Michael R Darby and Edi Karni. Free competition and the optimal amount of fraud. *The Journal of law and economics*, 16(1):67–88, 1973.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu

Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence. 2026. URL https://arxiv.org/abs/2606.19348.

Jasper Dekoninck, Nikola Jovanović, Tim Gehrunger, Kári Rögnvaldsson, Ivo Petrov, Chenhao Sun, and Martin Vechev. Beyond benchmarks: Matharena as an evaluation platform for mathematics with llms. 2026. URL https://arxiv.org/abs/2605.00674.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, et al. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Uwe Dulleck and Rudolf Kerschbamer. On doctors, mechanics, and computer specialists: The economics of credence goods. *Journal of Economic literature*, 44(1):5–42, 2006.

Mehmet Hamza Erol, Batu El, Mirac Suzgun, Mert Yuksekgonul, and James Y Zou. Cost-of-pass: An economic framework for evaluating language models. In *International Conference on Learning Representations*, volume 2026, pp. 34103–34127, 2026.

Zhiyu Fan, Kirill Vasilevski, Dayi Lin, Boyuan Chen, Yihao Chen, Zhiqing Zhong, Jie M Zhang, Pinjia He, and Ahmed E Hassan. Swe-effi: Re-evaluating software ai agent system effectiveness under resource constraints. *arXiv preprint arXiv:2509.09853*, 2025.

Jie Gao, Kaiser Sun, Jen-tse Huang, Katherine Van Koevering, Sijie Ji, Heyuan Huang, Weiyan Shi, Zhuoran Lu, Ziang Xiao, Daniel Khashabi, et al. How to interpret agent behavior. *arXiv preprint arXiv:2605.13625*, 2026.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. Deepswe: Measuring frontier coding agents on original, long-horizon engineering tasks. *arXiv preprint arXiv:2607.07946*, 2026.

Naman Jain. Reward hacking is swamping model intelligence gains. Cursor Blog, June 2026. URL https://cursor.com/blog/reward-hacking-coding-benchmarks.

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

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026.

Sophie Leroy. Why is it so hard to do my work? the challenge of attention residue when switching between work tasks. *Organizational behavior and human decision processes*, 109(2):168–181, 2009.

Gloria Mark, Daniela Gudith, and Ulrich Klocke. The cost of interrupted work: more speed and stress. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pp. 107–110, 2008.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Yeabin Moon. The price of thinking: Reasoning effort as a model-specific api contract. *arXiv preprint arXiv:2608.16956*, 2026.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

OpenAI. GPT-5.6 System Card. OpenAI Deployment Safety Hub, July 2026a. URL https://deploymentsafety.openai.com/gpt-5-6.

OpenAI. Reasoning models. https://developers.openai.com/api/docs/guides/reasoning, 2026b. API documentation. Accessed September 16, 2026.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Romain Robbes, Théo Matricon, Thomas Degueule, Andre Hora, and Stefano Zacchiroli. Agentic much? adoption of coding agents on github. *ACM Transactions on Software Engineering and Methodology*, 2026.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Guoheng Sun, Ziyao Wang, Bowei Tian, Meng Liu, Zheyu Shen, Shwai He, Yexiao He, Wanghao Ye, Yiting Wang, and Ang Li. Coin: Counting the invisible reasoning tokens in commercial opaque llm apis. *arXiv preprint arXiv:2505.13778*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He,

Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence. 2026. URL https://arxiv.org/abs/2607.24653.

Thinking Machines Lab. Inkling: Our open-weights model. https://thinkingmachines.ai/news/introducing-inkling/, July 2026. Published July 15, 2026.

Ander Artola Velasco, Stratis Tsirtsis, Nastaran Okati, and Manuel Gomez-Rodriguez. Is your llm overcharging you? tokenization, transparency, and incentives. *arXiv preprint arXiv:2505.21627*, 2025.

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

Z.ai. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, June 2026. Accessed: 2026-09-24.

Jian Zheng and Ge Gao. Fragmented moments, balanced choices: How do people make use of their waiting time? In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–14, 2024.

Table 1: Benchmarks and execution settings. AIME uses ten attempts per problem and GPQA uses four attempts per question; the agentic benchmarks evaluate multi-turn executions with fixed harnesses.

Benchmark	Evaluation size	Execution setting
AIME 2026 Dekoninck et al. (2026)	30 problems, 10 attempts each	Single API call per attempt
GPQA Diamond Rein et al. (2023)	198 questions, 4 attempts each	Single API call per attempt
Terminal-Bench 2.1 Merrill et al. (2026)	89 tasks	Multi-turn terminal agent using Terminus-2
SWE-Bench Pro Deng et al. (2025)	44-instance pilot	Multi-turn repository-level agent using mini-swe-agent

Table 2: Serving API and evaluated reasoning-effort settings for each model.

Model	API	Effort settings
GPT-5.6-sol, GPT-5.6-terra, GPT-5.6-luna	OpenAI	`none`, `low`, `medium`, `high`, `xhigh`
Claude Sonnet 5	Anthropic	`none`, `low`, `medium`, `high`, `xhigh`
GLM-5.2	Together AI	`none`, `high`, `max`
Inkling	Together AI	`none`, `low`, `medium`, `high`, `max`
Kimi K3	Together AI	`low`, `high`, `max`
DeepSeek V4 Flash	Together AI	`none`, `low`, `high`, `max`

A Benchmark details

For SWE-Bench-Pro, we sample 44 instances by deterministically selecting four evenly spaced entries from the lexicographically sorted list of instance IDs for each of the benchmark's 11 repositories.

B Inference setup

Table 2 lists the API through which each model is served and the reasoning-effort settings we evaluate. The available settings vary across models: for example, GLM-5.2 supports only `high` and `max`, while Kimi K3 supports only reasoning-enabled modes. For the proprietary models, we evaluate all settings except for `max` because the extremely high price makes it impractical to evaluate. For the same budgetary reason, we evaluate Kimi K3 only on AIME and GPQA.

Effort control. For the GPT-5.6 models, we specify the selected setting via `reasoning.effort` in the Responses API for AIME and GPQA, and via `reasoning_effort` through LiteLLM for the agentic benchmarks. For Claude Sonnet 5, `none` explicitly disables extended thinking; all other settings enable adaptive thinking, with `output_config.effort` set to the selected level. For models hosted on Together, we include `reasoning_effort` in the request body.

Sampling and output limits. We use the providers' default decoding parameters, except for the Terminus-2 agent, for which we retain its default temperature of 0.7. On AIME and GPQA, we set an output limit of 128,000 tokens per call for all models and allow up to five attempts for calls that encounter API errors, using exponential backoff. On SWE-Bench Pro, we set an output limit of 32,000 tokens and a spending cap of $5 per call. For Terminal-Bench, we retain the default limits of Terminus-2.

Single-call benchmarks. For each AIME attempt, we send the MathArena instruction and the problem in a single user message, without a system prompt, and extract the answer from the final

Table 3: Experimental cost in USD, rounded to one decimal place.

Model / cost category	AIME	GPQA	SWE-Pro	TB	Total (USD)
Benchmark inference					
GPT-5.6-sol	72.4	125.9	1,142.7	165.1	1,506.0
GPT-5.6-terra	29.2	47.6	312.1	131.8	520.8
GPT-5.6-luna	4.9	10.2	15.4	19.3	49.8
Claude Sonnet 5	217.2	230.3	1,045.2	322.3	1,815.0
GLM-5.2	44.7	126.7	804.9	255.7	1,232.1
Inkling	60.0	104.8	416.3	412.3	993.4
Kimi K3	128.5	150.4	–	–	278.9
DeepSeek V4 Flash	3.0	4.4	107.9	510.3	625.6
Inference subtotal	**559.9**	**800.2**	**3,844.6**	**1,816.9**	**7,021.6**
Act·ONOMY annotation (1,917 retained annotations)					3,453.2
Self-routing (12,026 retained selection calls)					44.2
Developmental cost: inference and routing					2,012.3
Developmental cost: annotation					≈446.0
Developmental subtotal					**≈2,458.3**
Total API-equivalent cost					**≈12,977.4**

`\boxed{}` expression in the response. For each GPQA attempt, we use the multiple-choice prompt from OpenAI's `simple-evals`[5], shuffle the answer choices independently, and extract the answer from the `Answer:` line.

Agentic benchmarks. We evaluate Terminal-Bench 2.1 using the `harbor` harness and the Terminus-2 agent under the official configuration, with one trial per task and no turn limit beyond the task-specific wall-clock timeout. We evaluate SWE-Bench Pro using `mini-swe-agent` under the benchmark's official configuration, with one run per instance, a maximum of 250 model calls per instance, and a 60-second timeout per shell command. The GPT-5.6 models, Claude Sonnet 5, and GLM-5.2 use the text-based scaffold, which requires exactly one bash code block per response.

C Experimental cost

Table 3 includes an approximation of the cost of this study, including developmental cost and finalized experiments. To keep the comparison across models and reasoning-effort settings within budget, we restrict the SWE-Bench Pro evaluation to 44 instances spanning all 11 repositories.

D Overall performance across effort levels

Table 4 reports overall accuracy for each model and effort setting, including both the original and corrected SWE-Bench Pro pilots. Terminal-Bench trials without a successful verifier verdict are counted as failures, and provider refusals on GPQA are counted as incorrect attempts. For SWE-Bench Pro, we follow the benchmark's scoring rule: executions that pass the evaluator but cost more than $5 are counted as failures.

E Instances gained and lost on AIME and GPQA

Figure 7 presents the comparison shown in Figure 1 for AIME, GPQA, and the corrected SWE-Bench Pro environments. For each pair of adjacent effort levels, solid bars indicate the number of instances solved at the higher effort level but not at the lower level; hollow bars indicate the number solved at the lower level but not at the higher level.

[5] `https://github.com/openai/simple-evals`

Table 4: Overall accuracy (%) across reasoning-effort settings, with 95% confidence intervals in brackets, rounded to whole percentage points. AIME and GPQA report mean per-attempt correctness, with percentile bootstrap intervals (10,000 resamples) over problems that keep each problem's attempts together; the agentic benchmarks report the fraction of tasks solved, with Wilson score intervals. Intervals describe each setting separately; because all settings of a model share the same tasks, paired tests between settings are more sensitive than a comparison of overlapping intervals. Bold marks the highest accuracy within each model–benchmark row, including ties; dashes indicate settings without a reported result. Effort names retain their model-specific meanings.

Model	none	low	medium	high	xhigh	max
AIME 2026 (30 problems; 10 attempts per problem)						
GPT-5.6-luna	25.3 [14, 38]	75.7 [65, 85]	85.3 [75, 94]	94.7 [89, 99]	**96.7** [93, 99]	–
GPT-5.6-terra	7.3 [2, 14]	89.7 [81, 96]	91.3 [82, 98]	92.3 [86, 98]	**97.0** [94, 99]	–
GPT-5.6-sol	55.3 [42, 68]	96.7 [90, 100]	97.7 [95, 100]	**99.7** [99, 100]	**99.7** [99, 100]	–
Claude Sonnet 5	79.7 [68, 90]	72.3 [60, 83]	86.3 [78, 93]	96.7 [92, 99]	**97.3** [92, 100]	–
GLM-5.2	46.0 [35, 57]	–	–	86.3 [76, 95]	–	**90.7** [81, 98]
Inkling	25.7 [18, 34]	77.7 [67, 87]	90.7 [81, 98]	93.7 [86, 99]	–	**94.3** [86, 100]
Kimi K3	–	65.7 [54, 76]	–	89.0 [84, 93]	–	**90.7** [85, 95]
DeepSeek V4 Flash	65.0 [52, 77]	96.3 [90, 100]	–	**97.3** [95, 99]	–	95.3 [92, 98]
GPQA Diamond (198 questions; 4 attempts per question)						
GPT-5.6-luna	64.0 [58, 70]	81.8 [77, 86]	84.8 [81, 89]	88.9 [85, 92]	**90.0** [86, 94]	–
GPT-5.6-terra	72.6 [67, 78]	86.1 [82, 90]	87.9 [84, 91]	89.0 [85, 92]	**89.8** [86, 93]	–
GPT-5.6-sol	82.4 [78, 87]	91.0 [87, 94]	90.7 [87, 94]	92.3 [89, 95]	**93.1** [90, 96]	–
Claude Sonnet 5	81.8 [77, 86]	78.8 [74, 83]	84.6 [80, 89]	88.8 [85, 92]	**90.0** [86, 94]	–
GLM-5.2	66.7 [62, 71]	–	–	89.4 [86, 93]	–	**91.9** [89, 95]
Inkling	67.4 [62, 73]	65.3 [60, 71]	81.2 [77, 85]	84.3 [80, 88]	–	**87.9** [84, 91]
Kimi K3	–	84.2 [80, 88]	–	91.5 [88, 95]	–	**94.3** [91, 97]
DeepSeek V4 Flash	73.4 [68, 78]	89.8 [86, 93]	–	89.9 [86, 93]	–	**91.4** [88, 95]
Terminal-Bench 2.1 (89 tasks)						
GPT-5.6-luna	39.3 [30, 50]	49.4 [39, 60]	52.8 [43, 63]	62.9 [53, 72]	**70.8** [61, 79]	–
GPT-5.6-terra	60.7 [50, 70]	65.2 [55, 74]	68.5 [58, 77]	**74.2** [64, 82]	**74.2** [64, 82]	–
GPT-5.6-sol	66.3 [56, 75]	76.4 [67, 84]	76.4 [67, 84]	**85.4** [77, 91]	82.0 [73, 89]	–
Claude Sonnet 5	**75.3** [65, 83]	57.3 [47, 67]	64.0 [54, 73]	70.8 [61, 79]	**75.3** [65, 83]	–
GLM-5.2	58.4 [48, 68]	–	–	**71.9** [62, 80]	–	**71.9** [62, 80]
Inkling	**42.7** [33, 53]	31.5 [23, 42]	29.2 [21, 39]	36.0 [27, 46]	–	37.1 [28, 47]
DeepSeek V4 Flash	65.2 [55, 74]	56.2 [46, 66]	–	64.0 [54, 73]	–	**69.7** [59, 78]
SWE-Bench Pro, original (44 instances)						
GPT-5.6-luna	52.3 [38, 66]	38.6 [26, 53]	54.5 [40, 68]	65.9 [51, 78]	**84.1** [71, 92]	–
GPT-5.6-terra	45.5 [32, 60]	56.8 [42, 70]	63.6 [49, 76]	65.9 [51, 78]	**72.7** [58, 84]	–
GPT-5.6-sol	65.9 [51, 78]	61.4 [47, 74]	72.7 [58, 84]	72.7 [58, 84]	**77.3** [63, 87]	–
Claude Sonnet 5	59.1 [44, 72]	54.5 [40, 68]	52.3 [38, 66]	61.4 [47, 74]	**79.5** [65, 89]	–
GLM-5.2	**72.7** [58, 84]	–	–	65.9 [51, 78]	–	70.5 [56, 82]
Inkling	63.6 [49, 76]	18.2 [10, 32]	52.3 [38, 66]	61.4 [47, 74]	–	**65.9** [51, 78]
DeepSeek V4 Flash	52.3 [38, 66]	59.1 [44, 72]	–	**84.1** [71, 92]	–	**84.1** [71, 92]
SWE-Bench Pro, corrected (44 instances)						
GPT-5.6-luna	40.9 [28, 56]	47.7 [34, 62]	47.7 [34, 62]	52.3 [38, 66]	**56.8** [42, 70]	–
GPT-5.6-terra	52.3 [38, 66]	50.0 [36, 64]	52.3 [38, 66]	50.0 [36, 64]	**61.4** [47, 74]	–
GPT-5.6-sol	43.2 [30, 58]	59.1 [44, 72]	56.8 [42, 70]	**61.4** [47, 74]	59.1 [44, 72]	–
Claude Sonnet 5	50.0 [36, 64]	**56.8** [42, 70]	52.3 [38, 66]	47.7 [34, 62]	50.0 [36, 64]	–
GLM-5.2	50.0 [36, 64]	–	–	**54.5** [40, 68]	–	43.2 [30, 58]
Inkling	43.2 [30, 58]	25.0 [15, 39]	38.6 [26, 53]	45.5 [32, 60]	–	**54.5** [40, 68]
DeepSeek V4 Flash	56.8 [42, 70]	54.5 [40, 68]	–	61.4 [47, 74]	–	**72.7** [58, 84]

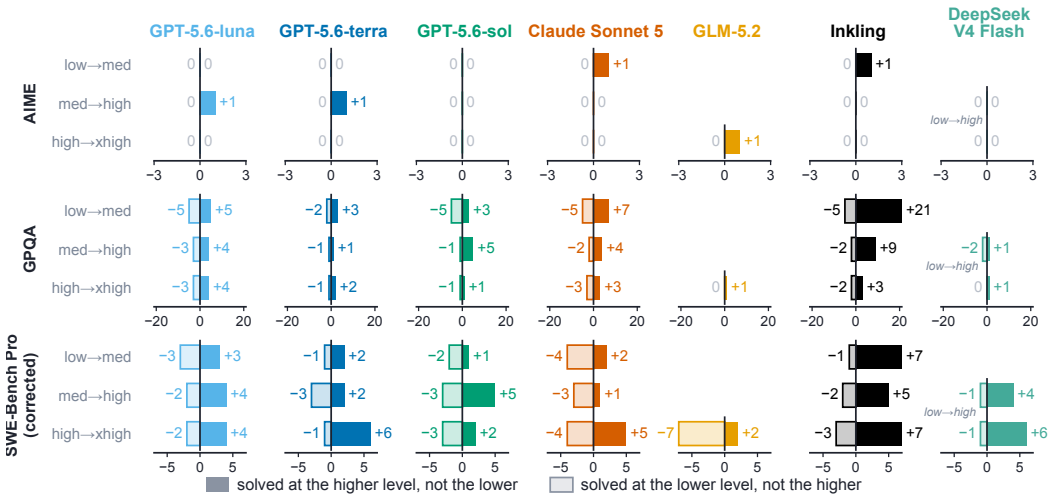


Figure 7: Instances gained and lost between adjacent reasoning-effort settings on AIME, GPQA, and the corrected SWE-Bench Pro environments. An instance counts as solved at a level if any of its attempts at that level is correct.

F User waiting cost dominates the bill

The inference costs reported in §3.2 exclude the opportunity cost of the time users spend waiting for an agent to complete its execution. Figure 9 extends the always-solved cost comparison in Figure 2 by adding this waiting cost to the API charges, following the full-cost formulation in Equation 1. There, C_{API} is the inference bill in U.S. dollars, N_{out} counts all generated tokens across the execution, including reasoning tokens, and r_m is the model's tabulated output throughput in tokens per second. We set the hourly wage $w = 33.54$, the average wage across all U.S. occupations.[6] The factor of 3600 converts decoding time from seconds to hours. This estimate assumes that the user is blocked throughout the generation and accounts only for decoding time; it excludes prompt processing, queueing, and tool-execution latency. If these additional delays also block users, the corresponding waiting cost would be higher.

Under these assumptions, **waiting cost accounts for approximately 58–98% of the combined cost** across the agentic model–effort comparisons in the figure. For example, at the highest effort level, GPT-5.6-sol incurs mean API charges of $1.24 on the always-solved SWE-Bench Pro set; including waiting raises the mean cost to $4.62, of which 73% is attributable to user time. Thus, even when valued at the mean wage across all occupations, the modeled cost of user time can exceed the API charges. In practice, users may perform other activities while an agent is running (Zheng & Gao, 2024). However, idle waiting persists in the use of coding agents (Robbes et al., 2026), and context switching can impose additional cognitive costs, including increased stress and impaired subsequent performance (Mark et al., 2008; Leroy, 2009). These effects are not captured by our estimate.

Figure 8 extends the routing comparisons in Figures 5 and 11 by replacing API charges with full cost. Relative to fixed top effort, including waiting cost increases laddering's savings from 17.4% to 26.3% for GPT-5.6-sol, from 4.8% to 19.5% for GPT-5.6-terra, and from 8.7% to 18.1% for Claude Sonnet 5. For GPT-5.6-luna, an 8.2% cost increase becomes a 5.1% saving. Laddering remains more costly for GLM-5.2, Inkling, and DeepSeek V4 Flash, but the respective cost increases decline from 38.8%, 11.5%, and 88.3% to 20.1%, 7.3%, and 54.2%. Using the mean hourly wage for software developers ($71.20) yields a median saving of 5.3% across the seven models, compared with 5.1% at the mean wage across all occupations.

[6]Source: U.S. Bureau of Labor Statistics, Occupational Employment and Wage Statistics, May 2025 national estimates, Table 1, which reports a mean hourly wage of $33.54 across all occupations. While there are many caveats to this analysis, we believe it is an approximate estimate of the time cost required to solve problem instances.

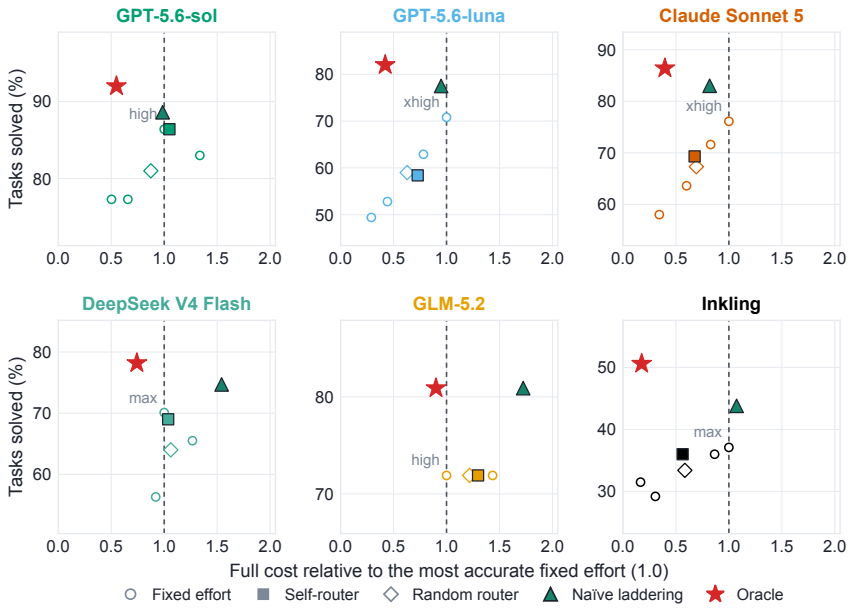


Figure 8: Self-routing and naïve laddering on Terminal-Bench 2.1 with user waiting cost added to the API bill (Equation 1), for the six models with saved self-routing comparisons. Each panel shows success against full cost, normalized by the full cost of that model's fixed effort with the highest performance (dashed line). The oracle selects the successful effort with the lowest full cost; router calls are charged their API cost only. Policies, markers, and task cohorts match Figure 5. Lower cost and higher success are preferable.

G Gold-patch retrieval on SWE-Bench Pro

Table 5 compares performance in the original and corrected environments at each model's highest evaluated reasoning-effort setting. All seven models achieve lower solve rates in the corrected environment, with declines of 11.4–29.5 percentage points. Because the two environments use the same instances, we assess each decline using an exact McNemar test on discordant outcomes, i.e., instances solved in one environment but not the other. The declines are significant for four of the seven models at $\alpha = 0.05$ and remain significant after Holm correction. A sign test across the seven models yields $p = 0.016$. These results indicate that the environment corrections substantially affected the agents' performance, raising questions about the validity of evaluation results using the official SWE-Bench-Pro release.

Figure 10 illustrates an instance in which the original SWE-Bench Pro container retains the gold fix commit in its Git history. At the highest effort level, the agent locates this commit and copies its patch; at the lowest effort level, it solves the same task without using the commit. Both runs are scored as resolved, yet the higher-effort run costs $5.3\times$ as much and reproduces an existing fix rather than independently deriving one.

H Evaluation of naïve laddering

We reconstruct laddering success rates and API costs from our existing executions, counting the costs of both attempts whenever escalation occurs. Each policy execution consists of an initial attempt at the lowest available reasoning-effort setting and, if that attempt fails, at most one fresh retry at the highest setting. On AIME and GPQA, we pair attempts across effort levels by problem and attempt index. The ten attempts per AIME problem and four per GPQA question therefore yield repeated policy evaluations; they are not treated as a batch that succeeds whenever any attempt is correct.

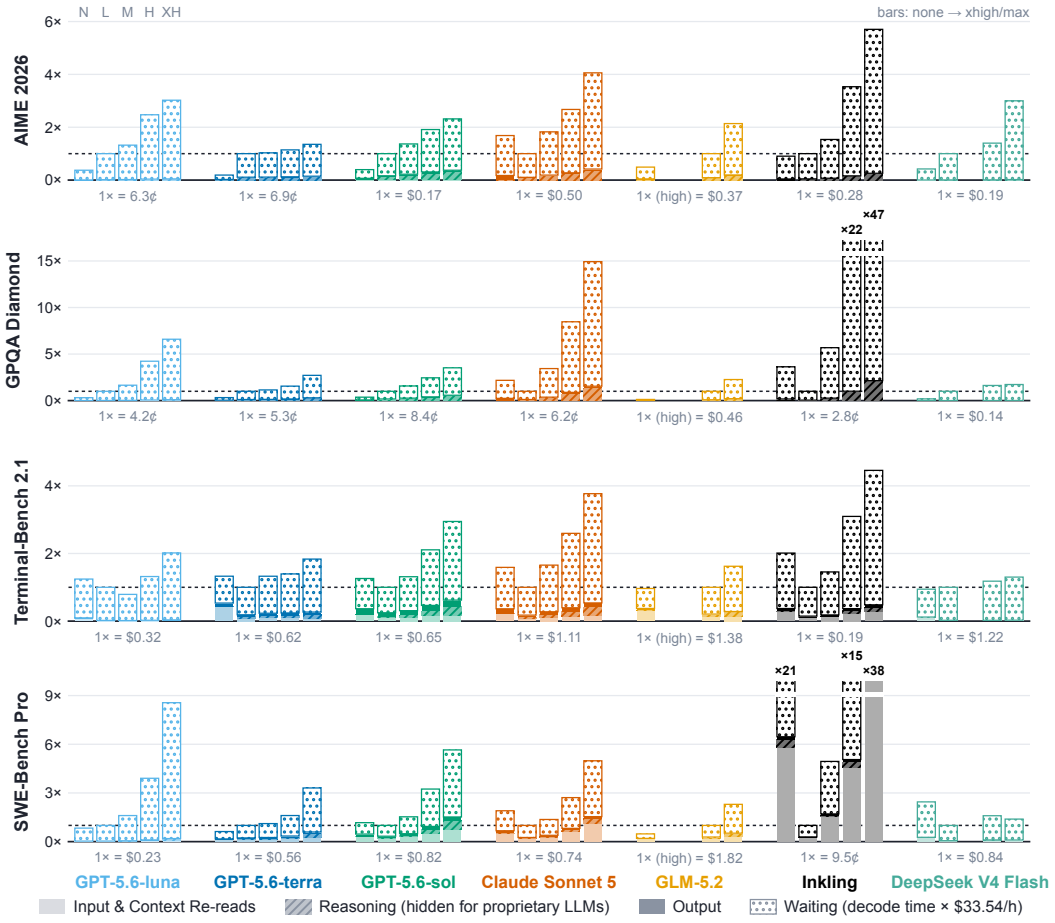


Figure 9: Full cost of one task the model solves at every effort level, from `none` to the highest level, relative to the full cost at `low` (dashed line; GLM-5.2 is anchored on `high`). Full cost adds to the API bill the time a user waits for the output tokens at the model's published throughput, priced at the BLS mean hourly wage of $33.54. Each bar is split into input, hidden reasoning, visible output, and waiting. The always-solved set is fixed per model and benchmark; the footer gives the dollar value of $1\times$. Bars taller than the axis are clipped and labelled.

Laddering includes the full recorded costs and applies no additional cap to the combined cost of an initial attempt and its retry. The analysis assumes a reliable external correctness verdict and evaluates a fresh retry, rather than self-detection of failure or continuation of the initial trajectory. Preserving successful initial attempts makes the recorded success set the union of the starting and highest effort levels' successes. Cost savings depend on whether avoiding high-effort runs on initial successes offsets the cost of failed first attempts.

I SELF-ROUTING ON THE REMAINING MODELS

Figure 11 repeats the self-routing comparison of Figure 5 for GLM-5.2 and Inkling on Terminal-Bench 2.1.

Table 6 gives the prompt for the self-router.

Table 5: Solve rates on the original and corrected SWE-Bench Pro environments at each model's highest reasoning effort setting, using the same 44 instances in both environments. Corrected removes the task repository's original git history and restricts network access to allowlisted package registries to prevent unintended problem-solving approaches. Δ is corrected minus original solve rate in percentage points, computed before rounding. The *Solved only in* columns count discordant instances, and p is the two-sided exact McNemar test on those counts. The decrease is significant at the 5% level for 4 of the 7 models, with all 4 remaining significant after Holm correction across models. Table 4 reports results at all effort levels.

					Solved only in		
Model	Effort	Original (%)	Corrected (%)	Δ (pp)	Original	Corrected	p
GPT-5.6-luna	`xhigh`	84.1	56.8	−27.3	13	1	0.002
GPT-5.6-terra	`xhigh`	72.7	61.4	−11.4	7	2	0.180
GPT-5.6-sol	`xhigh`	77.3	59.1	−18.2	8	0	0.008
Claude Sonnet 5	`xhigh`	79.5	50.0	−29.5	13	0	<0.001
GLM-5.2	`max`	70.5	43.2	−27.3	13	1	0.002
Inkling	`max`	65.9	54.5	−11.4	5	0	0.062
DeepSeek V4 Flash	`max`	84.1	72.7	−11.4	6	1	0.125

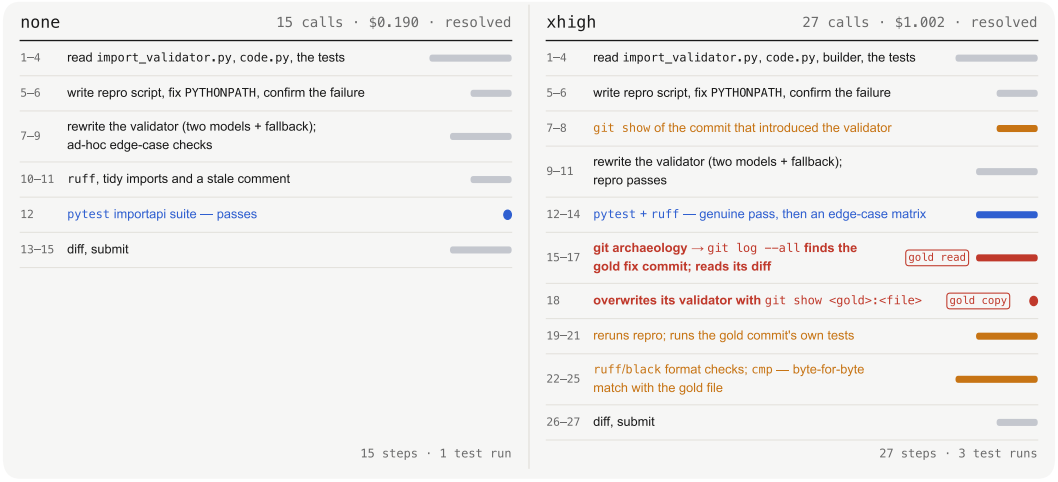


Figure 10: The summarized trajectory of GPT-5.6-sol on the SWE-Bench Pro instance `openlibrary-00bec1e7` at `none` (left, 15 calls, $0.19) and `xhigh` (right, 27 calls, $1.00); both runs resolve the task. Each row is a phase of consecutive agent steps and bar length is the number of steps in that phase. Grey marks routine work, blue a genuine test run, orange a look into git history or validation against the copied file, and red a read or copy of the gold fix commit. At `none` the agent fixes the validator from the issue text and the code. At `xhigh` it does the same work, then runs `git log -all`, finds the gold fix commit still present in the container's clone, overwrites its own file with `git show <gold>:<file>`, runs that commit's tests, and confirms a byte-for-byte match before submitting.

J Ablation: Repeated Agentic Runs

Our agentic benchmarks use one execution per task at each effort level, so individual task outcomes are subject to execution variability. To assess whether our findings hold with repeated executions, we apply the same analyses to the official runs released by DeepSWE (Huang et al., 2026). DeepSWE v1.1 evaluates 113 long-horizon software-engineering tasks using `mini-swe-agent`, with four rollouts per task at each effort level. The official release includes five models from our study: GPT-5.6-luna, GPT-5.6-terra, GPT-5.6-sol, and Claude Sonnet 5 at all settings from `low` to `max`, and GLM-5.2 at `high` and `max`. We exclude DeepSeek V4 Flash because DeepSWE evaluates it only at `max`. For this analysis, a task is considered solved at an effort level if at least one rollout

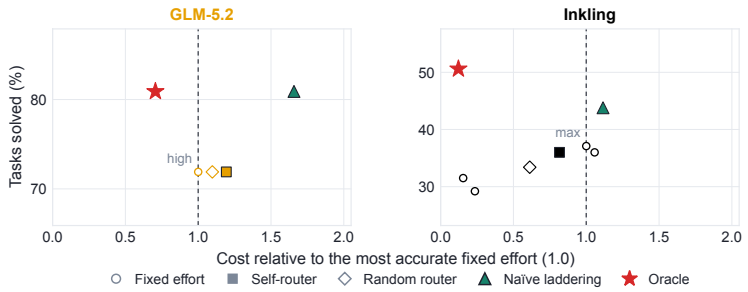


Figure 11: Self-routing and naïve laddering compared with fixed efforts, random allocation, and the oracle on Terminal-Bench 2.1 for GLM-5.2 and Inkling. The layout and policies match Figure 5. Naïve laddering starts at `low` for Inkling and at `high` for GLM-5.2, retrying verified failures at `xhigh` or `max`. Lower cost and higher success are preferable.

passes; rollouts that terminate with an error are counted as failures. We use the per-rollout results downloaded on September 23, 2026.

J.1 Instances gained and lost

Figure 12 extends the comparison in Section 3.1 to DeepSWE. With four rollouts per task, a task solved at the lower effort level is considered lost at the higher level only if all four higher-effort rollouts fail. Regressions persist under this stricter criterion: every pair of adjacent effort levels exhibits at least one lost task for every model. For Claude Sonnet 5, increasing effort from `high` to `xhigh` gains 7 tasks but loses 12, reducing the number solved from 90 to 85 out of 113. For GPT-5.6-sol, the same transition gains 2 tasks and loses 3. GLM-5.2 gains 18 tasks and loses 8 when moving from `high` to `max`. GPT-5.6-luna is the only model whose highest-effort solved set contains its lowest-effort solved set: `max` solves all five tasks solved at `low`. The other four models lose between 2 and 11 tasks when moving from their lowest to their highest effort level. Thus, increasing reasoning effort can still reduce task coverage when success is measured across four executions per task.

J.2 Cost of always-solved tasks

Figure 13 repeats the always-solved cost comparison of Figure 2 on DeepSWE, and Figure 14 adds the waiting cost of Appendix F. DeepSWE does not run `none`, and we plot GPT-5.6 and Claude Sonnet 5 from `low` to `xhigh`, the highest level of our agentic runs, and GLM-5.2 at `high` and `max`. We price each rollout from its token counts at the same rates as our own runs. Claude Sonnet 5's trajectories do not record reasoning tokens, so its bars show input and output only.

The finding in Section 3.2 also holds on DeepSWE: among tasks solved at every plotted level, mean API cost increases with each successive effort level for all five models. From `low` to `xhigh`, mean API cost rises by $4.16\times$ for GPT-5.6-sol, $4.45\times$ for GPT-5.6-terra, and $5.20\times$ for Claude Sonnet 5, compared with $2.51\times$ and $3.04\times$ for GPT-5.6-sol and Claude Sonnet 5 on Terminal-Bench. GPT-5.6-luna's cost rises by $13.58\times$, although only 4 of the 113 tasks are solved at every level. GLM-5.2's cost rises by $1.44\times$ from `high` to `max`. As in our evaluations, input tokens account for most of the API cost, contributing 69–91% of every bar, and reasoning accounts for at most 15% of the runs. Adding waiting cost preserves the ordering, and waiting accounts for 48–95% of the full cost; from `low` to `xhigh`, full cost rises by $3.87\times$ for GPT-5.6-sol and $3.88\times$ for Claude Sonnet 5.

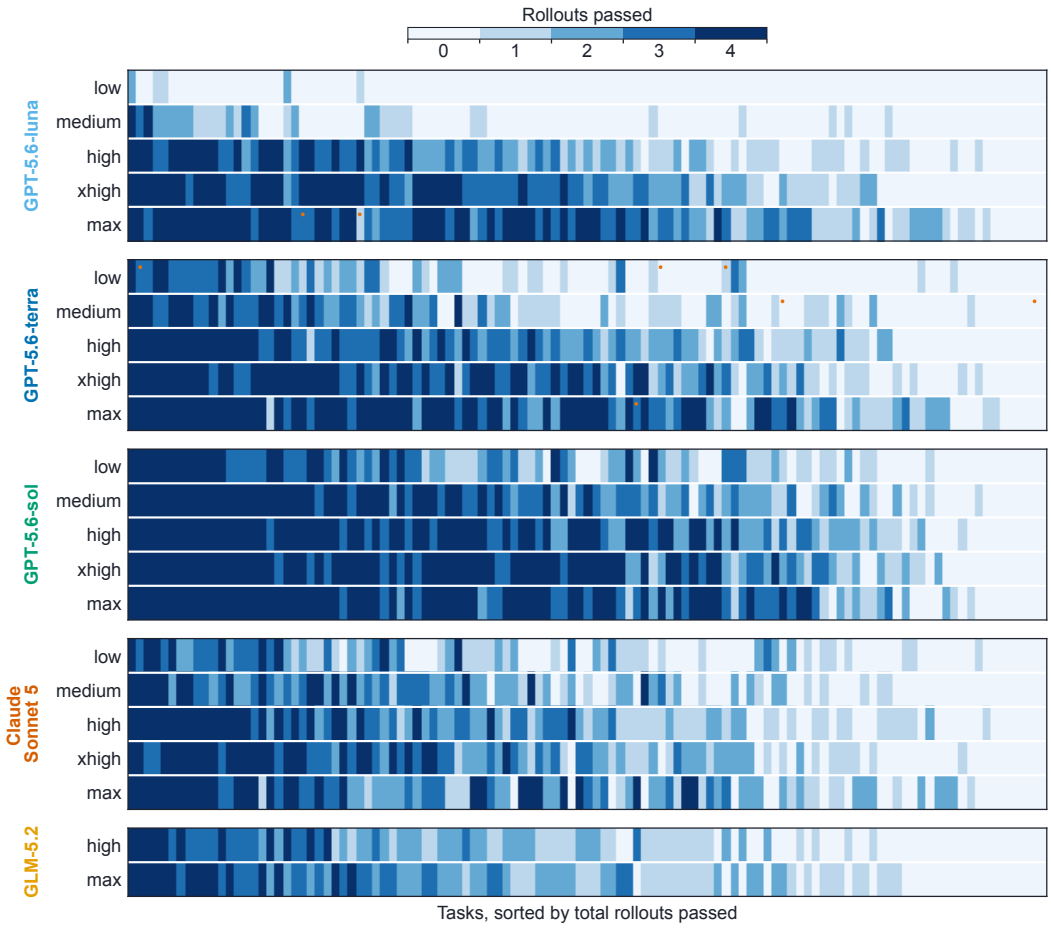


Figure 12: Rollouts passed on each DeepSWE task at every reasoning-effort setting. Each row is an effort setting and each column is a task; columns are sorted separately for each model by total rollouts passed, so a column position does not identify the same task across models. A task solved at a lower setting but not at a higher one appears as a shaded cell above an unshaded cell in the same column. Orange dots mark the eight cells with fewer than four successful rollouts in the official DeepSWE release.

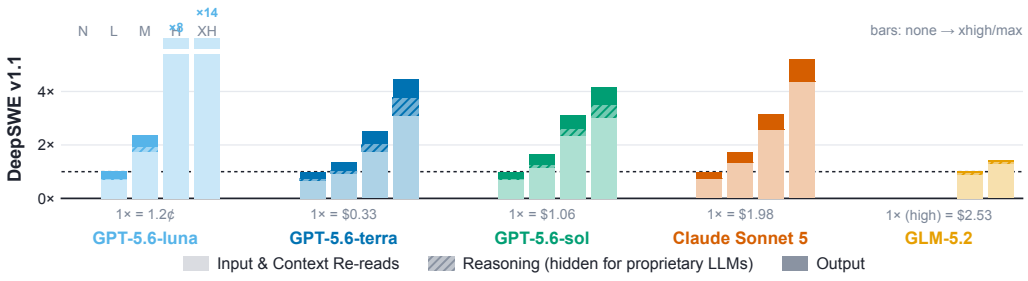


Figure 13: API cost of one DeepSWE task the model solves at every plotted effort level, relative to the cost at `low` (dashed line; GLM-5.2 is anchored on `high`). A task's cost at a level is the mean over all its passing rollouts. The footer gives the dollar value of 1×. Bars taller than the axis are clipped and labelled.

Table 6: Prompt used for self-routing. Each selection sends the template in (a) as a single user message, with the fields in angle brackets filled in and the unit name written in capitals in the task delimiters. The task description is the text the model would receive at the start of the run. The selection call runs at `low` effort, or at `high` for GLM-5.2, which has no `low` setting. The chosen level is read from the `EFFORT:` line, or otherwise from the last option named in the response.

(a) Prompt template

You are about to be given one ⟨*unit*⟩ to solve on your own.

Setting: ⟨*setting*⟩

Before you start, choose the reasoning effort you will run at. Your options, from least to most thinking:
⟨*option 1*⟩ — ⟨*description 1*⟩
...
⟨*option k*⟩ — ⟨*description k*⟩

Thinking longer costs more money and more time, and on a hard task it is often what makes the difference between solving it and not. Thinking longer on an easy task buys nothing. Choose the LOWEST level you believe will still let you solve this particular ⟨*unit*⟩.

Do not attempt the task now. Do not explain your choice.

--- BEGIN ⟨*UNIT*⟩ ---
⟨*task description*⟩
--- END ⟨*UNIT*⟩ ---

Reply with exactly one line, nothing else:
EFFORT: <one of: ⟨*option 1*⟩, ..., ⟨*option k*⟩>

(b) Effort options

Option	Description
`low`	the least thinking — fastest and cheapest
`medium`	moderate thinking
`high`	extended thinking
`xhigh`, `max`	the most thinking this model offers — slowest and dearest

(c) Benchmark-specific fields

Benchmark	Unit	Setting
Terminal-Bench 2.1	task	You will solve a command-line task inside a Linux container, as an autonomous agent sending batches of shell keystrokes to a terminal and reading the screen back (the Terminus-2 scaffold). You get one attempt. You succeed only if the task's own verifier script passes afterwards.
SWE-Bench Pro	instance	You will work on a real GitHub issue inside the repository's Docker container, as an autonomous agent driving a shell one command at a time (the mini-swe-agent scaffold). You get one attempt. You succeed only if the repository's own fail-to-pass and pass-to-pass tests both pass on the patch you leave behind.
AIME 2026	problem	You will answer this AIME competition problem in a single response, with no tools, no code execution and no second chance to revise. The answer is an integer between 0 and 999 and is marked right or wrong on that integer alone.
GPQA Diamond	question	You will answer this graduate-level multiple-choice science question in a single response, with no tools, no search and no second chance to revise. Only the letter you pick is marked.

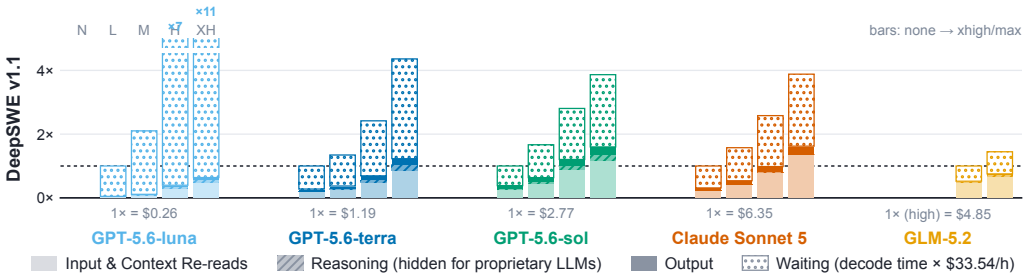


Figure 14: Full cost of one DeepSWE task the model solves at every plotted effort level, relative to the full cost at `low`.