ByteDance | Seed

EdgeBench: Unveiling Scaling Laws of Learning from Real-World Environments

ByteDance Seed

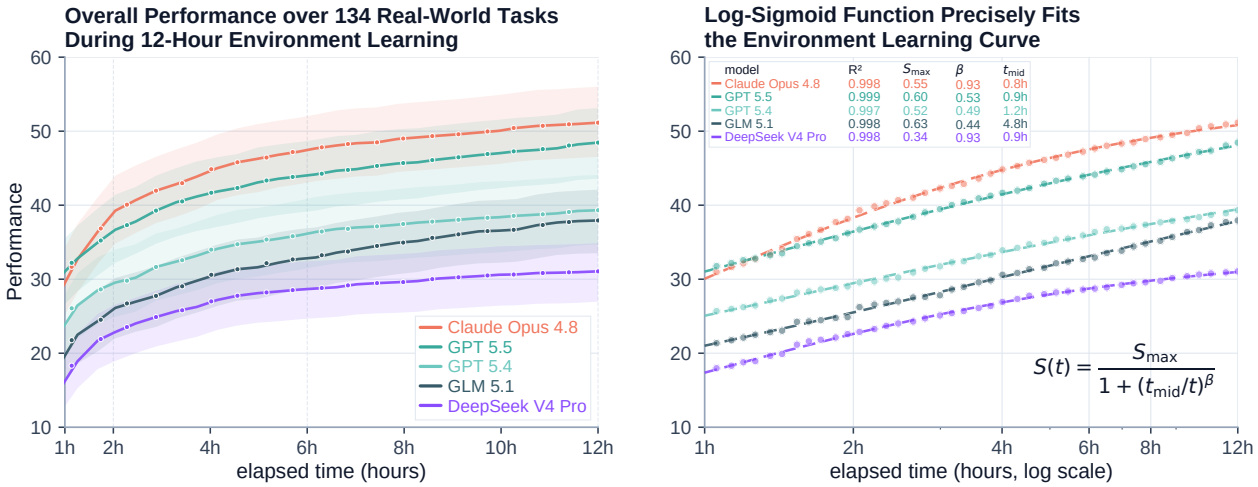


Figure 1 Overall performance follows a precise log-sigmoid relationship with environment interaction time when averaged over all 134 tasks (mean $R^2 = 0.998$).

Abstract

Pretraining scaling laws reveal that model capability improves predictably with data and compute. But learning from real world environments after deployment remains far less understood. Analyzing roughly 38,000 hours of agent interaction with the environment across 134 real world tasks, we find, to the best of our knowledge, the first evidence that overall performance during environment learning follows a log-sigmoid scaling law with remarkably high precision, reaching $R^2 = 0.998$. Across model generations, we also find that agent learning speed roughly doubles every three months. This discovery stems from EdgeBench, a suite of 134 real world tasks with ultra-long horizons, spanning scientific discovery, software engineering, combinatorial optimization, professional knowledge work, formal mathematics, and interactive games. Each task sustains at least 12 hours of continuous agent operation under rich, multilevel feedback, and is built through substantial expert effort. We publicly release 51 tasks and our full evaluation framework to accelerate the study of how agents learn from real world experience.

Date: July 2, 2026
Correspondence: Shu Zhong at zhongshu@bytedance.com
Project Page: https://edge-bench.org

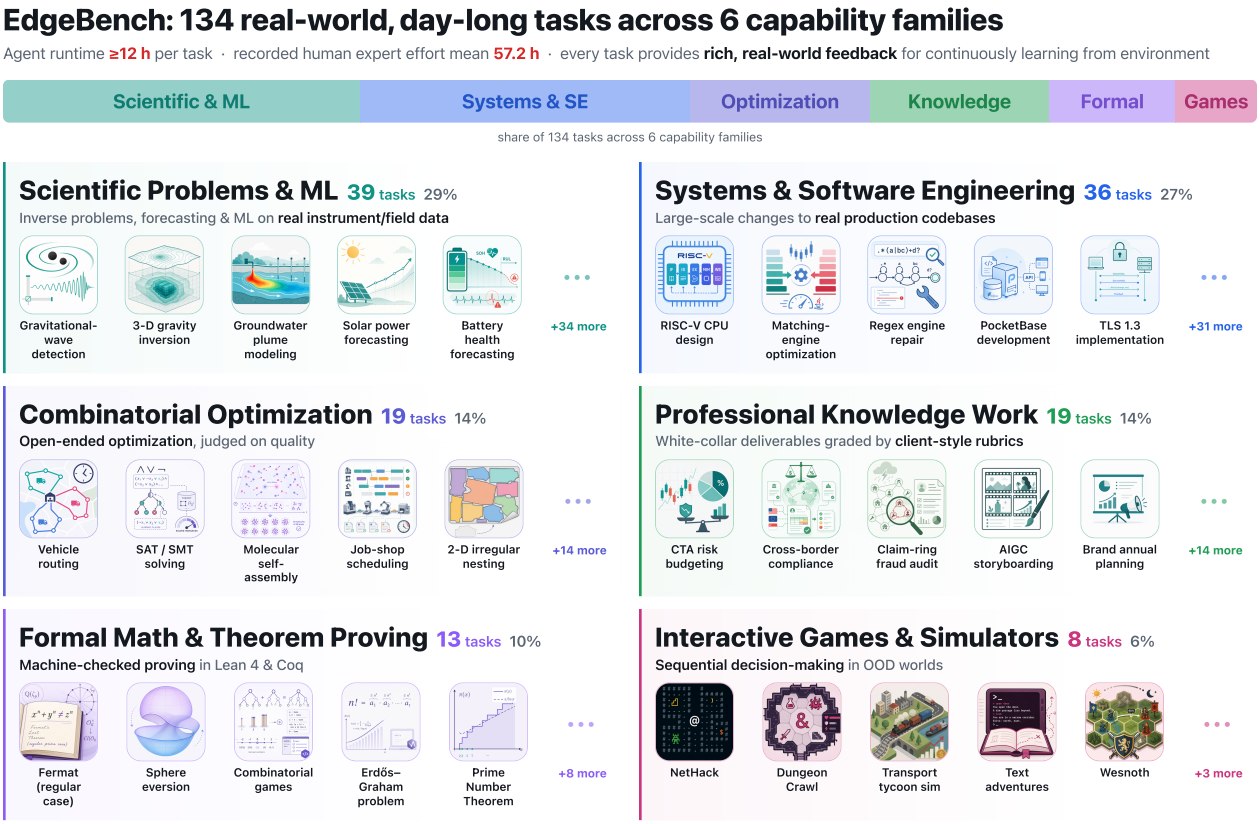


Figure 2 EdgeBench task taxonomy. 134 real world tasks across six capability families, with feedback channels designed to support within-run improvement. Recorded human expert effort estimates: mean 57.2 h.

1 Introduction

Scaling laws for pretraining [32, 36] revealed that model capability improves predictably with data and compute, an insight that guided much of the progress in the years that followed [2, 3, 22, 24, 25, 49, 53–55, 57, 58]. Now, as large language models enter the agent era, they are being deployed into a growing range of real world environments where they can try to learn from interaction, yet whether learning from such environments obeys a clean scaling law remains unknown. Analyzing roughly 38,000 hours of agent environment interaction across 134 diverse real world tasks, we find, to the best of our knowledge, the first evidence that **when agents learn from real world environments, overall performance follows a log-sigmoid scaling law as a function of environment interaction time**, achieving remarkably high precision with $R^2 = 0.998$.

Why study agents' ability to learn from their environments? Real world use of AI depends on more than what a model learned during training. Some needed knowledge never appears in training data, such as private records and internal tools. Even when raw data exists, it omits the human process behind it: the trial-and-error, the interpretation of evidence, and the adaptation to feedback through which experts actually reach results. The real world also never stands still: human knowledge keeps advancing, and new tools, discoveries, and problems continually emerge that no fixed training corpus can anticipate. Therefore, an agent's ability to learn from its environment and improve task performance is central to deploying AI systems at scale in the real world.

Studying this ability requires task environments that resemble real use, provide informative feedback for learning, and allow agents enough time to learn through interaction. However, existing benchmarks often lack such feedback or limit agents to only minutes or a few hours, making them not well suited to this

goal [10, 15, 35, 46, 69, 81]. To address this gap, we created **EdgeBench**. Our benchmark contains **134 realistic and diverse tasks spanning six capability families**, from scientific research and software engineering to formal mathematics and interactive games, as shown in Figure 2. Each task runs in an executable workspace that combines fast local exploration with slower judge feedback on submitted artifacts, mirroring real-world workflows. Agents can work for at least **12 hours** on each task (by contrast, Agents' Last Exam [69] averages roughly one hour per task), while we record their submissions and track how performance changes throughout the run. These tasks are substantial even for human experts: recorded expert effort averages **57.2 hours** per task and reaches up to **320 hours**. This makes EdgeBench a natural testbed for studying how agents learn from their environments over long horizons.

Using EdgeBench, we evaluate frontier agents over roughly 38,000 hours of environment interaction. Our study makes four main observations:

- **Environment learning exhibits a precise log-sigmoid scaling law.** Averaged performance follows the same functional form across the full benchmark, across task families, under longer interaction horizons up to 72 hours, and when forecasting later performance from early trajectories.
- **A theoretical derivation of the log-sigmoid law.** We propose a theory that models environment learning as a frontier expansion process on latent task graphs, which explains why benchmark-averaged progress takes the observed log-sigmoid form.
- **Agent learning speed doubles roughly every three months.** Studying frontier models released since September 2025, we find a rapid scaling trend in how quickly frontier agents learn from their environments.
- **Learning dynamics strongly shape long-horizon performance.** Long-horizon performance depends on how agents use accumulated experience, not only on how many attempts they make. Continuous experience outperforms independent restarts, longer context improves retention, and detailed case studies show that feedback can turn many failed probes into a few durable gains.

2 EdgeBench

2.1 Design Goals

EdgeBench aims to measure whether an autonomous agent can learn from experience in an unfamiliar real world environment. This requires two properties from a benchmark that existing evaluations lack:

- **Ultra-long-horizon, diverse tasks.** Learning behaviors such as exploration, strategy revision, and experience accumulation need time and complexity to emerge. Short tasks are usually solved from memory rather than learning, so measuring learning calls for long-horizon tasks. Because learning is a general capability, these tasks must also span diverse domains.
- **Realistic, multi-level feedback.** In practice, human experts learn from rich feedback: test failures, experimental results, unexpected phenomena, authoritative judgments, and more. A benchmark that cannot offer such rich feedback cannot measure learning, and leaves the agent guessing what the evaluation actually rewards. We need feedback that approximates the real world, so we can measure true, general-purpose learning.

The first requirement motivates our task taxonomy (§2.2): 134 tasks across six capability families, each designed as a day-scale challenge that supports frontier models running for at least 12 hours. The second motivates our evaluation protocol (§2.3): each task individually simulates its own slice of the real world, providing isolated work and judge environments, local agent-driven feedback, submission-gated judge feedback, and host-side trajectory measurement.

2.2 Task Taxonomy

We searched for real world tasks that satisfy two criteria: a performance ceiling high enough that no current agent can saturate it, and a workflow that supports continuous learning rather than one-shot completion.

This search, conducted in collaboration with domain experts across fields, identified six capability families and yielded 134 curated tasks (Figure 2):

- **Scientific Problems & ML** (39 tasks). Each task uses real world research data and experimental settings sourced from working scientists. Domain expertise is essential: agents must formulate hypotheses, choose models, validate against noisy observations, and refine iteratively. Many problems are open-ended, with no known optimal solution.
- **Systems & Software Engineering** (36 tasks). Agents work on production-grade codebases where a single task may require thousands of lines of change, with over 100,000 lines in the largest cases. Because the code spans interdependent modules, an agent must reason about cross-module coupling while meeting both correctness and performance targets.
- **Combinatorial Optimization** (19 tasks). These are open-ended, predominantly NP-hard problems where exact methods are intractable and progress depends on designing, tuning, and iterating on heuristic search strategies. Even strong solvers have room to improve with additional time and feedback.
- **Professional Knowledge Work** (19 tasks). These tasks reproduce real white-collar deliverables across finance, education, healthcare, and legal domains, matching work that would take a human professional with three or more years of experience roughly three full days to complete. Many tasks feature carefully designed rubrics and multi-round delivery feedback that approximate real client review cycles, so agents can learn from structured critique and revise iteratively.
- **Formal Math & Theorem Proving** (13 tasks). These tasks sit at the frontier of mathematical difficulty and require building large-scale machine-checked proofs in Lean, coupling deep mathematical insight with substantial formal-verification engineering. Most are newly created for EdgeBench and designed to support iterative progress: agents receive structured intermediate guidance and can extend partial proofs incrementally.
- **Interactive Games & Simulators** (8 tasks). These are real games designed for human players, where proficient humans typically invest tens of hours to master the mechanics. The state spaces are enormous and each run is procedurally distinct, so agents face strong out-of-distribution pressure. Agents must develop and refine strategies through high-frequency interaction across many episodes.

Tasks whose primary difficulty lies in visual understanding, especially GUI operation, are excluded. When success depends on the vision backbone rather than iterative reasoning, learning ability and perceptual capability are hard to separate.

2.3 Feedback Loop and Evaluation Protocol

Real world engineering and research workflows rarely provide a single final answer check. Instead, practitioners iterate through **two complementary feedback loops**: a fast local loop for exploration, debugging, and refinement, and a slower external loop that provides authoritative calibration through deployment, peer review, benchmark evaluation, or stakeholder feedback. The local loop enables rapid progress, while the external loop guards against overfitting to visible checks and exposes failures that are not captured by the developer's own tests.

EdgeBench adopts this dual-loop structure to measure learning rather than endpoint success (Figure 3). The inner loop is local and agent-driven: agents can inspect a writable workspace, run tests or simulators, observe errors, and revise their artifacts. The outer loop is judge-mediated: submitted artifacts are evaluated against hidden cases or private grading criteria, returning calibrated scores, verdicts, or diagnostics. Across task families, this pattern is instantiated through different mechanisms: tests and profilers for software tasks, development splits and validators for scientific tasks, local testers and hidden seeds for optimization tasks, proof-checker states for theorem proving, episode scores for games, and rubrics for professional knowledge work.

The protocol is implemented with an isolated work–judge evaluation harness. During a run, the agent works inside a work container holding the task materials and local validation tools, but no hidden evaluation assets.

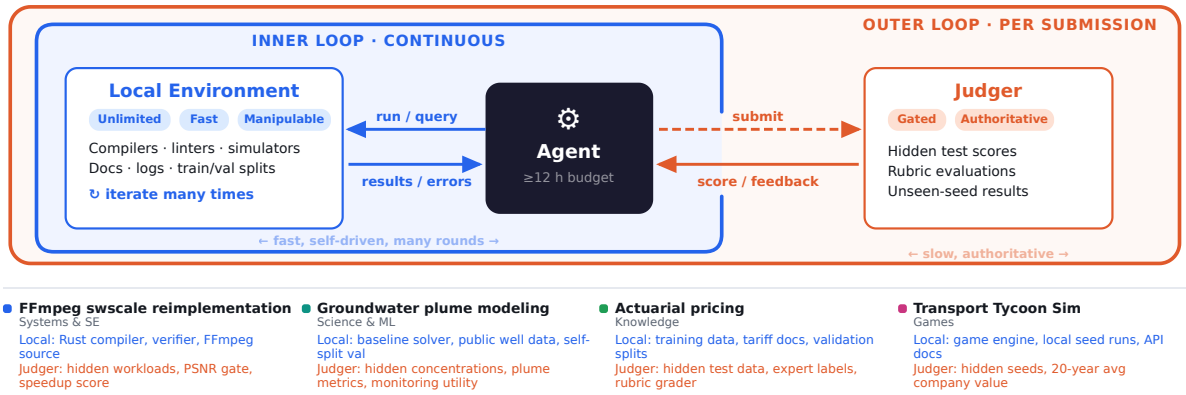


Figure 3 The informative feedback loop in EdgeBench. The inner loop (blue) lets agents iterate freely with local feedback; the outer loop (orange) gates authoritative judge feedback behind submissions. Bottom: representative tasks showing how both feedback channels are instantiated across capability families.

The agent can actively submit its current artifacts to a separate judge container, which runs the hidden evaluation and returns the feedback specified by the task. A host-side judge server mediates this outer loop, including submission queues, cooldowns, authentication, and support for asynchronous grading on long-running evaluations, allowing agents to continue working while submitted jobs are being judged. Appendix C gives examples of task-design failure modes that motivated this isolation and submission-mediated design.

For trajectory measurement, the evaluation harness also performs host-side auto-evaluation at fixed intervals. These snapshots are scored through the hidden judge and recorded for analysis, but the results are not shown to the agent. This lets EdgeBench measure improvement even between explicit submissions, while preserving the distinction between agent-visible feedback and evaluator-only measurement.

3 Scaling Laws of Learning from Real World Environments

Pretraining scaling laws classically model language-model loss as a power-law function of training scale, including the amount of pretraining data [32, 36]. By contrast, benchmark performance is a task-level readout of model capability, shaped by the difficulty thresholds induced by tasks, examples, and scoring criteria. Prior work has found that it is well described under pretraining scale-up by a log-sigmoid curve [7, 22, 59, 64].

Beyond learning from human-collected training data, modern agents such as GPT-5.5 and Claude Opus 4.8 can continue to learn from their environments after deployment. On a task, they can acquire new information through interaction and improve their performance over time. Yet it remains unclear whether this form of learning obeys any similarly simple scaling law. EdgeBench provides 134 diverse real world tasks with executable environments, informative feedback, and at least 12-hour interaction windows, allowing us to study how agents improve through interaction with their environments. Analyzing five frontier agents over roughly **38,000 hours of environment interaction** across 134 diverse real world tasks, we find that **a log-sigmoid curve fits environment learning performance precisely**. Thus, learning from the environment and learning from pretraining data induce the same mathematical scaling form.

3.1 From Task Trajectories to Predictable Scaling Curves

Experimental setting. We evaluate 134 EdgeBench tasks with five frontier models: Claude Opus 4.8 [3], GPT-5.5 [58], GPT-5.4 [57], GLM-5.1 [26, 84], and DeepSeek-V4-Pro (preview) [17]. For each task–model pair, we run three independent 12-hour trials and record the full submission trajectory. GPT models are run with Codex using a 256k compact window, while GLM-5.1 and DeepSeek-V4-Pro are run with Claude Code using

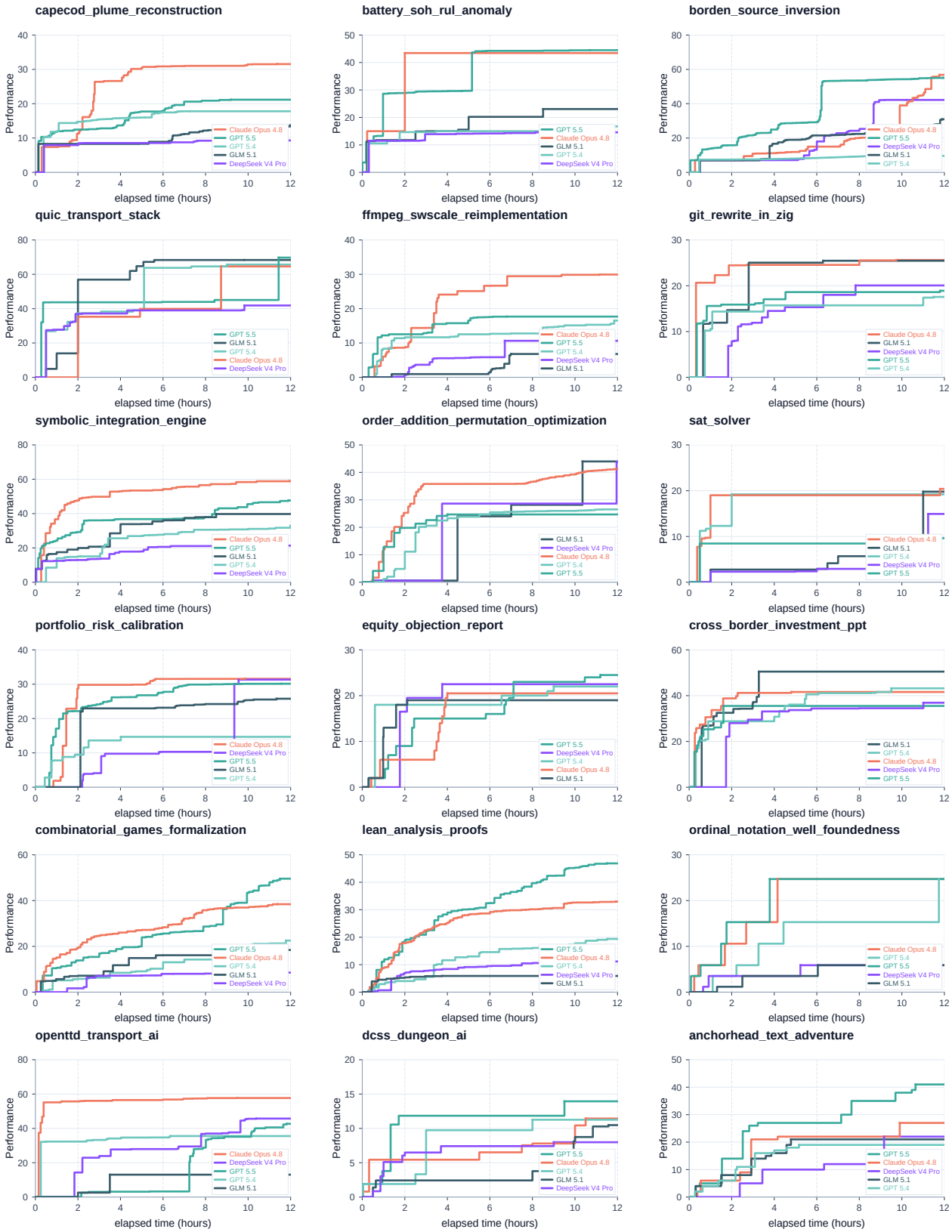


Figure 4 Learning curves over 12 hours for 18 representative tasks across six capability families. The remaining task curves are provided in Figures 15–35.

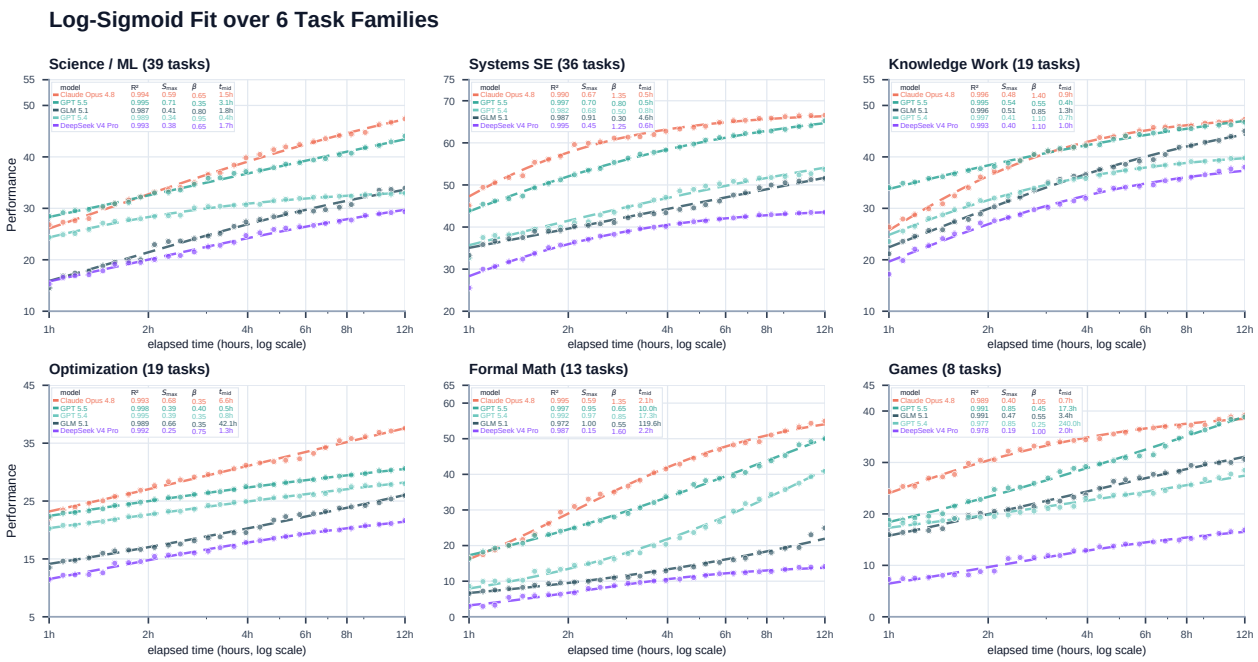


Figure 5 Task-family-level log-sigmoid fits across the six EdgeBench task families. Despite large differences in task type and scoring function, the same log-time sigmoidal form fits the average trajectory in each task family well.[1]

200k compact windows. Claude Opus 4.8 is primarily run with a 1M Claude Code compact window; we additionally include a 200k versus 1M Opus ablation in Section 5.3.

Per-task trajectory diversity. Figure 4 shows per-task learning curves for 18 representative tasks, illustrating how different models improve on the same task over time. The full set of per-task curves is provided in Appendix G.5 for all 134 tasks. The tasks span diverse domains and exhibit heterogeneous learning dynamics, with trajectories ranging from smooth incremental gains to long plateaus, abrupt breakthroughs, and irregular regressions.

Aggregate curves reveal a common structure. Although these individual curves are heterogeneous across both tasks and models, their cross-task averages are unexpectedly smooth and share a common structure. Motivated by prior log-sigmoid fits of benchmark performance under pretraining scale-up, we fit the averaged environment learning curves with the following three-parameter log-sigmoid model:

$$S(t) = \frac{S_{\max}}{1 + (t_{\text{mid}}/t)^{\beta}}, \tag{1}$$

where t is elapsed interaction time and $S(t)$ is best-so-far performance. The fitted parameters serve as empirical descriptors of the aggregate learning curve: $S_{\max}$ is the attainable score ceiling, t_{mid} is the interaction time at which the curve reaches half of that ceiling, and β controls how sharply progress concentrates in log time. Thus a smaller t_{mid} means the model reaches the bulk of its attainable score sooner, while a larger β corresponds to a steeper learning transition. Section 3.2 shows that this simple functional form fits the empirical learning curves surprisingly well.

3.2 Log-Sigmoid Curves Fit Environment Learning with Remarkable Precision

We find that the log-sigmoid fit is precise and robust across every setting we tested:

[1] Fit precision improves as the number of fitted tasks increases; see Figure 8.

[2] GPT-5.4 service availability dropped in the latter half of the experiment, which may partly explain its forecast deviation; see Appendix B.

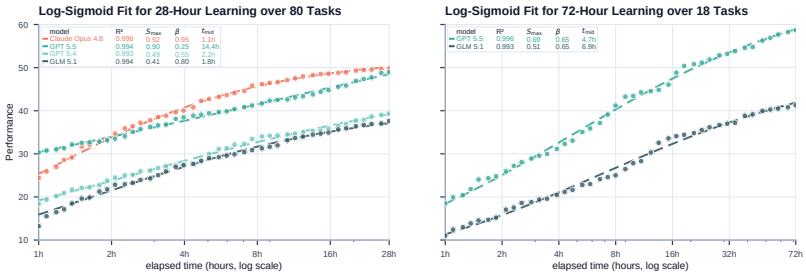


Figure 6 Long-horizon log-sigmoid fits beyond the main 12-hour benchmark window. The left panel fits 28-hour trajectories averaged over 80 tasks; the right panel fits 72-hour trajectories averaged over 18 tasks. All fitted curves remain highly precise, with $R^2 \geq 0.993$.

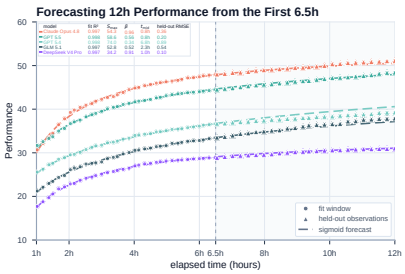


Figure 7 Forecasting 12-hour performance from the first 6.5 hours. Log-sigmoid fits on this early window accurately predict the held-out remainder for all five models.[2]

- **Log-sigmoid curves precisely fit the 134-task average for all five models.** As shown in Figure 1, after averaging over all 134 tasks, the fitted log-sigmoid curve closely tracks each model's 12-hour learning trajectory. The fit is uniformly tight across all five models, with $R^2 \geq 0.997$ in every case.
- **The same log-sigmoid form persists across heterogeneous task families.** Figure 5 repeats the fit separately across the six capability families in EdgeBench. These families require different knowledge, exercise different capabilities, and produce visibly different aggregate learning curves. Yet each family is still well described by the same log-sigmoid form, including smaller families where fewer tasks make the averaged trajectories noisier.
- **The fit remains precise under substantially longer interaction horizons.** Figure 6 extends the analysis beyond the main 12-hour window to 28-hour and 72-hour interaction horizons. Due to resource limits, the 28-hour fit covers 80 tasks and four models, while the 72-hour fit covers 18 tasks and two models. The same log-sigmoid structure remains stable across both longer settings, with every fitted curve reaching $R^2 \geq 0.993$.
- **The log-sigmoid law exhibits predictive power.** Figure 7 tests this by fitting each 12-hour aggregate curve using only the first 6.5 hours, then evaluates the forecast on held-out observations from 6.5 to 12 hours. The extrapolated curves remain close to the later observed trajectories across all five models, with $R^2 \geq 0.997$ and RMSE below 1.0 performance point in every case.

These precise fits raise two questions: is the log-sigmoid genuinely the right curve, or would any S-shape do, and where does so clean a law come from?

Log-sigmoid fits best among common S-curves. Many saturating growth processes are S-shaped [77]. We therefore compare the log-sigmoid against other common S-curves, each viewing an S-shaped learning curve as a cumulative distribution over a time coordinate: log-probit [8], log-Gompertz [27], and a Weibull CDF [74] on raw time, together with a two-parameter log-linear baseline; for the log-time families the independent variable is $x = \ln t$. We fit every family on the 12h, 28h, and 72h full windows and pool the error. As Table 1 shows, the log-sigmoid family attains the lowest RMSE, while the log-linear baseline is substantially worse. The empirical signal is thus robustly sigmoidal rather than tied to a single link function, and is not explained by mere linear improvement in $\ln t$. Appendix E discusses why we nevertheless prefer the log-sigmoid on mechanistic grounds.

The law emerges from a population of tasks. A single task's trajectory is noisy and idiosyncratic, yet Figure 8 shows that the log-sigmoid fit becomes steadily more precise as we average over more tasks: the residual error falls monotonically as tasks accumulate, from 1 task to all 134. The clean scaling law is therefore an *emergent, population-level* regularity, sharp only across many diverse tasks rather than within any one of them.

Family	Functional form	RMSE
Log-Sigmoid	$S(t) = \dfrac{S_{\max}}{1 + (t_{\text{mid}}/t)^{\beta}}$	**0.390**
Log-Probit	$S(t) = S_{\max}\,\Phi\left(\dfrac{\ln t - \mu}{\sigma}\right)$	0.398
Log-Gompertz	$S(t) = S_{\max}\exp[-\exp\{-c(\ln t - x_0)\}]$	0.402
Weibull CDF	$S(t) = S_{\max}\left(1 - \exp\{-(t/\lambda)^{\beta}\}\right)$	0.404
Log-Linear	$S(t) = a + b\ln t$	0.717

Table 1 Full-window fit error for three-parameter S-curve families and a two-parameter log-linear baseline. RMSE is measured in performance points on the 0–100 score scale; lower is better.

Figure 8 Log-sigmoid fit error decreases as more tasks are averaged.

3.3 A Theory of the Log-Sigmoid Law

The empirical fits show a robust log-sigmoid shape, but do not by themselves explain why this form should appear. We propose a theoretical model: environment learning is a frontier expansion process on the underlying task graphs. In this view, each task is modeled by a latent graph of score units, the already-unlocked units exert influence on the locked neighbors to unlock them, and progress occurs when the frontier between unlocked and locked score nodes advances. Appendix D gives the full derivation; here we summarize the mechanism. Throughout the section, we use

$$u = \log t - \log t_{\text{mid}}$$

as a change of coordinate of the time axis.

Environment learning is a frontier expansion process. For a single task, let us consider the task score is composed of many score units, representing nodes i on the task graph G with score w_i and normalized score weights $\mu_i = w_i / \sum_i w_i$. Let $n_i(u) \in \{0, 1\}$ indicate whether unit i has been unlocked at time u, the normalized score obtained is

$$x(u) = \sum_i \mu_i n_i(u).$$

On the task graph G, an edge weight $K_{ij} \geq 0$ measures how much an unlocked source unit j helps unlock a target unit i. Thus a locked unit i receives an influence field

$$h_i(u) = \sum_j K_{ij} n_j(u).$$

If locked units unlock randomly at an expected rate proportional to this field, then conditioned on the current state,

$$\frac{\mathrm{d}}{\mathrm{d}u}\mathbb{E}[x(u) \mid n(u)] = \eta \sum_{i \in L(u)} \sum_{j \in U(u)} \mu_i K_{ij}. \tag{2}$$

The expected score-growth rate is therefore exactly the weighted frontier cut from unlocked units $U(u)$ to locked units $L(u)$.

Frontier process moves at a speed proportional to $x(1-x)$. The exact frontier cut still depends on the task graph structure. The mean-field approximation is to assume that, at the aggregate level, every macroscopic unlocked–locked cut has approximately product-measure influence:

$$\sum_{i \in L} \sum_{j \in U} \mu_i K_{ij} \approx \kappa \mu(L)\mu(U).$$

where $\mu(A) = \sum_{i \in A} \mu_i$ for any set $A \subseteq G$. This, along with (2), gives

$$\frac{\mathrm{d}x}{\mathrm{d}u} = \beta x(1-x), \qquad \beta = \eta\kappa. \tag{3}$$

The two factors have direct interpretations: unlocked score mass supplies reusable capability, while locked score mass measures the remaining opportunity for improvement. Appendix D.2 formalizes this approximation using a weighted cut-mixing condition, which is weaker than assuming that all edge weights are individually equal.

The effective time coordinate is logarithmic. The frontier equation is written in an effective task-graph coordinate u. A natural reason for u to be approximately $\log t$ is self-similar graph structure[3]. If each additive increase in task difficulty exposes a multiplicatively larger amount of relevant graph structure, then search volume needed to traverse the graph grows exponentially with difficulty scale. If the search effort is approximately constant across time horizon, then the difficulty scale reached by time t grows as

$$u \sim \log t$$

Substituting this coordinate into the frontier equation (3) gives

$$\frac{\mathrm{d}x}{\mathrm{d}\log t} = \beta x(1-x). \tag{4}$$

Solving the frontier equation. Separating variables in (4) gives

$$\log \frac{x(t)}{1-x(t)} = \beta \log \frac{t}{t_{\mathrm{mid}}},$$

where t_{mid} is chosen so that $x(t_{\mathrm{mid}}) = 1/2$. Hence

$$x(t) = \frac{1}{1+(t_{\mathrm{mid}}/t)^{\beta}}, \qquad \Longrightarrow \qquad S(t) = \frac{S_{\max}}{1+(t_{\mathrm{mid}}/t)^{\beta}}.$$

Benchmark average is smoother than individual tasks. The argument above describes the limiting frontier drift. It does not imply that every finite task should visibly follow a smooth sigmoid. A task with a small number of score units can have long plateaus and sudden jumps. The empirical scaling law is instead a statement about the task-aggregate curve. If many independently evaluated tasks each follow approximate frontier dynamics, then averaging removes finite-task jaggedness. The aggregate becomes a single log-sigmoid when residual task midpoints and task speeds concentrate:

$$x_M(u) = \frac{1}{M}\sum_{b=1}^{M} x_b(u) \approx \frac{1}{M}\sum_{b=1}^{M} \frac{1}{1+e^{-\beta_b(u-\delta_b)}} \xrightarrow{P} \frac{1}{1+e^{-\beta u}}.$$

Appendix D.3 makes this statement precise under assumptions of blockwise cut-mixing, vanishing average jump noise, midpoint alignment, and speed concentration.

Interpretation of the fitted parameters. The fitted rate β has a natural interpretation as an effective frontier-propagation speed in log time. A larger β means that score unlocks over a narrower range of interaction scales, producing a steeper transition from low to high performance. A smaller β means that progress is spread over more multiplicative time, producing a more gradual learning curve. The fitted ceiling $S_{\max}$ should be interpreted as the attainable score support over the fitted regime, not necessarily as an absolute upper bound on performance.

Applicability and limitations. The log-sigmoid law is not expected to hold for every environment-learning process. It can fail when task graphs contain strong bottlenecks, dispersed task midpoints, heterogeneous

[3]This resembles scale-free dynamics in physical complex systems, such as self-organized criticality and critical phenomena, where behavior is not governed by a single characteristic scale [5, 48].

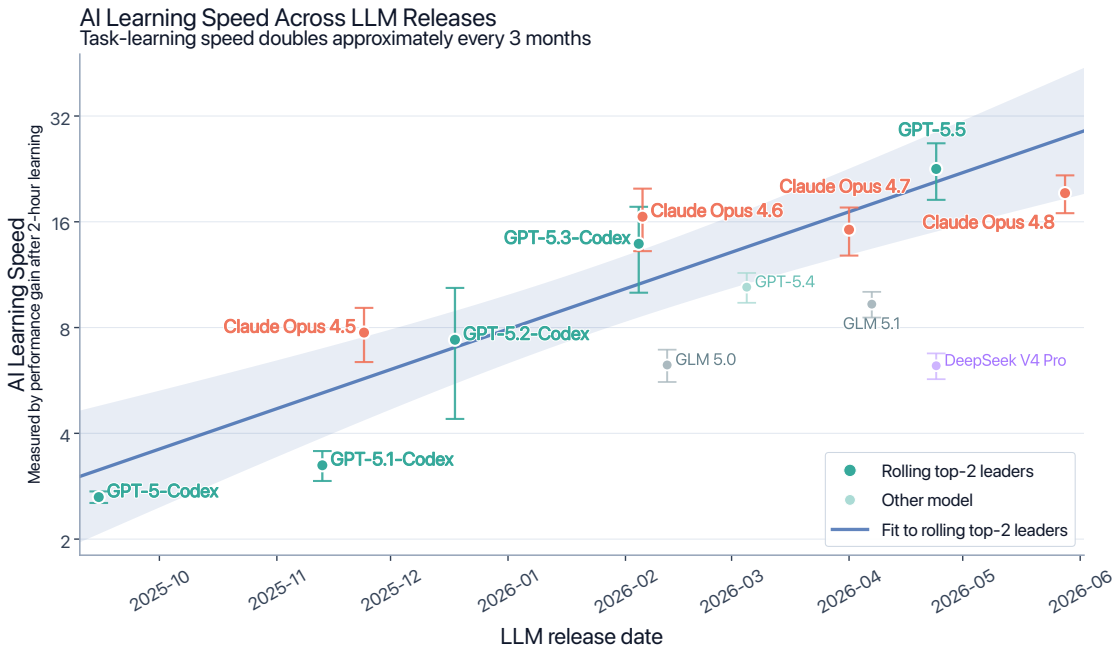


Figure 9 Learning speed across evaluated LLM releases. Learning speed denotes the two-hour performance gain on the fixed 18-task slice. The blue line fits the rolling top-2 leaders by release date and indicates an approximately three-month doubling trend.

frontier speeds, or non-scale-free graph structures. These failure modes clarify the scope of the theory: the log-sigmoid is most natural when progress behaves like a sufficiently mixed frontier expansion process on an approximately fractal structured graph. In this sense, learning from environments tests whether an agent can convert diverse feedback into reusable structure that accelerates subsequent discovery. We view this as central to why environment learning is worth measuring and scaling.

4 Agent Learning Speed Doubles Approximately Every Three Months

Frontier agents can now improve through interaction with task environments. This raises a separate question: are newer models learning from their environments faster? We measure this by comparing how much performance each agent gains over a fixed interaction budget across model release dates.

4.1 Experimental Design

Disentangling prior knowledge from environment learning. A high score may reflect what the model already knew rather than what it learned during the run. To reduce this confound, we select an 18-task slice from EdgeBench where models show similar first-attempt performance. With comparable starting points, later gains provide a cleaner measure of environment learning. We measure task-learning speed as the **average performance gain** over a fixed two-hour budget.

Evaluation protocol. We evaluate frontier open- and closed-source AI systems released from September 2025 through the current evaluation window, when frontier systems became capable of sustained autonomous runs. Each model is run three times per task. GPT models are run with Codex; all other models are run with Claude Code. As shown in Figure 10 (left), models start from comparable first-attempt performance on this 18-task slice, with an average of 6.87 ± 0.97.

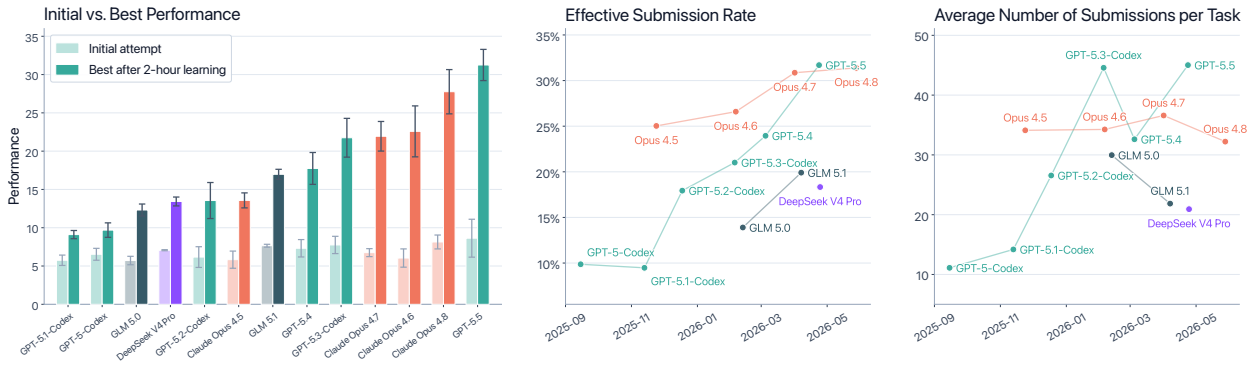


Figure 10 Learning outcomes and agent effort on the 18-task slice. Left: normalized initial performance and best performance after two hours. Middle: fraction of submissions that improve the best-so-far score. Right: average number of submissions per task. Error bars in the left panel show uncertainty over three replicate runs.

Model	Overall Score ↑						Category Score@12h ↑					
	@2h	@4h	@6h	@8h	@10h	**@12h**	Science	Code	Optimize	Knowledge	Math	Games
Opus 4.8	**39.0**	**45.7**	**48.1**	**49.8**	**50.9**	**51.3**	**48.5**	**67.4**	**36.5**	**47.0**	**55.0**	**39.3**
GPT-5.5	36.8	42.1	44.5	46.3	47.6	48.4	44.3	65.0	33.6	45.7	50.0	39.1
GPT-5.4	29.7	34.0	36.5	38.0	38.9	39.3	33.5	54.1	27.9	38.8	40.8	29.0
GLM-5.1	26.0	30.4	32.9	34.9	36.5	37.4	33.8	50.9	26.4	43.5	24.6	29.3
DS-V4-Pro	23.3	27.1	29.0	29.9	30.9	31.0	30.0	43.0	21.5	37.0	14.1	16.9

Table 2 Aggregate leaderboard. Overall scores are reported at each time budget, while category scores are reported at the 12-hour budget. Bold marks the best value in each score column and underlining marks the second-best value.

4.2 Learning Speed across Model Generations

Figure 9 reports the two-hour evaluation results on the fixed 18-task slice. The y-axis measures agent learning speed, defined as performance gain over two hours, on a log scale. To estimate the frontier trend, we use darker markers to highlight the top two models at each release date. We then fit a linear trend to these frontier points, with the fitted 95% confidence interval shown as the shaded band.

Figure 9 shows a rapid increase in learning speed across recent model generations. From GPT-5-Codex in September 2025 to GPT-5.5 in April 2026, learning speed increases by roughly 8× over 221 days. A log-linear fit to the frontier models captures this trend well, corresponding to **an approximate doubling every three months**. Figure 10 shows that this improvement is not simply explained by more frequent submissions. Submission frequency (right panel) changes unevenly: newer GPT models submit more actively, while other families do not. The middle panel tells a different story: later models turn a larger fraction of submissions into best-so-far improvements. This trend therefore reflects more effective learning from each interaction, not merely more attempts.

5 Analysis of Environment Learning Dynamics

We study how frontier models perform and learn from environmental feedback over long horizons. We evaluate five frontier models: Claude Opus 4.8, GPT-5.5, GPT-5.4, GLM-5.1, and DeepSeek-V4-Pro (DS-V4-Pro). This section contains four analyses. Section 5.1 compares frontier models at the aggregate, family, task, and submission levels. Section 5.2 tests whether accumulated experience adds value beyond independent restarts. Section 5.3 examines whether longer context still improves long-horizon interaction. Section 5.4 traces a single scientific task to show how an agent's improvements unfold. Appendix G.3 reports additional harness-level continuation ablations for /goal mode and the Ralph loop.

5.1 Comparison across Frontier Models

Setup. We follow Section 3.1 and analyze 12-hour trajectories from two angles: **(1)** aggregate, family, and task performance, and **(2)** submission efficiency, i.e., how often submissions improve the current best result.

Performance Comparison. Table 2 gives the aggregate leaderboard over the 2 to 12 hour budget and reports family-level means at 12 hours. Claude Opus 4.8 leads throughout the time budget and reaches 51.3 at 12 hours, followed by GPT-5.5 at 48.4. GPT-5.4 and GLM-5.1 form the next tier at 39.3 and 37.4, while DS-V4-Pro obtains 31.0. Family-level results are broadly consistent with the aggregate ranking: Claude Opus 4.8 leads each family mean, with GPT-5.5 especially close in Games and second overall. Detailed per-task performance is reported in Appendix G.6, with $*$ marking cells based on fewer than three valid runs.

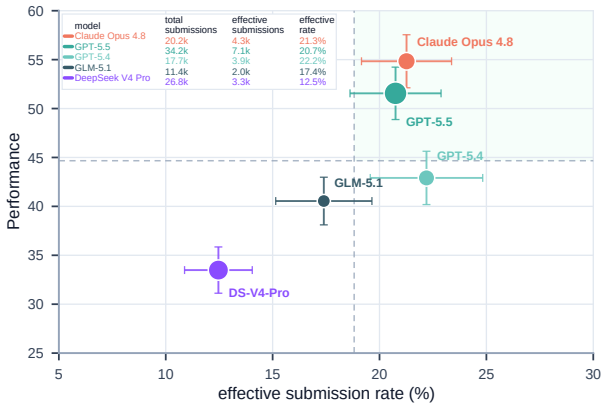


Figure 11 12-hour effective-submission rate versus final performance. Marker area shows total submissions; error bars show task-level standard errors, and shading marks above-average rate and performance.

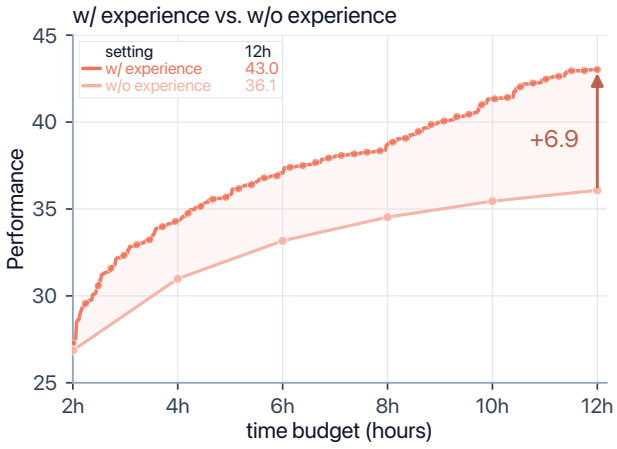
Submission efficiency. We count each agent submission and mark it as effective when it sets a new best-so-far score. Figure 11 compares each model's 12-hour effective-submission rate with its final performance. Models that make more effective submissions usually perform better, but more submissions do not automatically lead to a better final result. **Claude Opus 4.8 achieves the best final performance despite submitting less often than GPT-5.5**, and GPT-5.4 has the highest effective-submission rate but still trails the top two models. Progress therefore depends not only on how often an agent improves its score, but also on whether those improvements are large, reliable, and reusable. Stronger agents use feedback more deliberately: they build a submit-ready baseline, preserve the current best solution, make focused changes, and use feedback to keep gains or roll back failures. Weaker agents more often over-trust local proxies, bundle unrelated edits, or continue broad exploration after feedback has ruled out a direction, reducing sample efficiency.

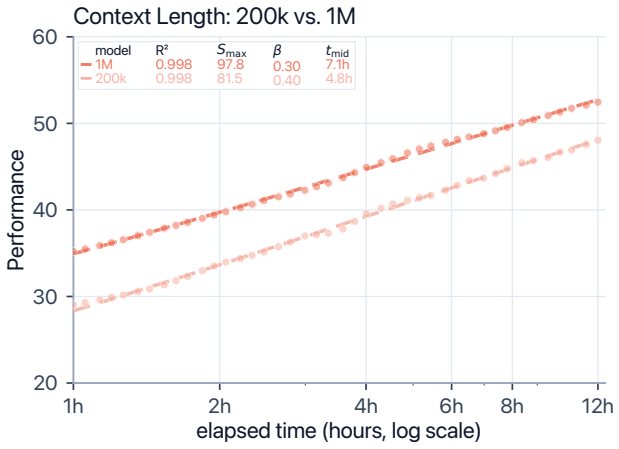
5.2 Agents Do Learn from Experience beyond Repeated Sampling

Motivation. A rising best-so-far curve does not by itself prove that an agent is learning. Running longer also gives more chances to stumble on a good solution: with enough independent tries, the best of them climbs by luck alone. We therefore test whether the agent's accumulated experience adds value beyond such repeated sampling under the same total time budget.

Setup. We give Opus 4.8 the same 12-hour budget on each of 17 tasks and compare two ways of spending it, *with* versus *without* accumulated experience. **With experience**: the agent runs once and continuously, keeping its workspace, artifacts, and feedback history throughout, so experience builds up across the whole run. **Without experience**: the same budget is split into $n = 6$ independent attempts of $\tau = 2$ hours, with all state discarded between attempts and only the best result kept, so each attempt starts from scratch and any gain can come only from repeated sampling. Comparing the two at elapsed time $t = k\tau$ contrasts one continuous run after t hours against the best of k independent attempts using the same total time. Appendix G.1 details how each curve is estimated.

Result. Figure 12a shows a clear gain from experience: the with-experience curve stays above the without-experience baseline under the same time budget. At 12 hours it reaches 43.0 versus 36.1, a gain of +6.9. The improvement is therefore not explained by repeated sampling alone: accumulating and reusing task experience drives progress beyond what independent restarts achieve.





	2h	6h	12h
w/o experience	26.9	33.2	36.1
w/ experience	$27.2_{+0.4}$	$37.1_{+3.9}$	$43.0_{+6.9}$

	2h	6h	12h
Opus 4.8 200k	33.8	42.6	48.0
Opus 4.8 1M	$39.6_{+5.8}$	$48.0_{+5.5}$	$52.5_{+4.4}$

Figure 12 **Left:** Gain from accumulated experience for Opus 4.8: the w/ experience run minus the w/o experience baseline (independent restarts) under the same total time budget. **Right:** Context-length ablation: 200k vs. 1M on Opus 4.8. Each curve shows the average performance over time; dashed lines are log-sigmoid fits.

5.3 How Much Does a Longer Context Improve Performance

Motivation. Section 5.2 shows that accumulated experience can improve performance. A remaining question is how this experience should be retained during a long run. Using long context is a natural way to do so. However, frontier-agent harnesses can also maintain state outside the model context through workspace files, compaction, progress notes, and memory-like artifacts. It is therefore unclear whether extending the context window still provides additional benefits once these external state channels are available, and if so, by how much.

Setup. We compare 200k-context Opus 4.8 with 1M-context Opus 4.8 on the 42-task subset under the same long-horizon evaluation protocol.

Result. A longer context yields a consistent multi-point gain throughout the 12-hour window (Figure 12b). The 1M-context Opus 4.8 stays above the 200k variant at every checkpoint, and the two trajectories run roughly parallel: the gap is +5.8 at 2h and only edges down to +4.4 by 12h, with both curves well described by the same log-sigmoid form. Thus, even with identical external workspace and harness state, a longer context window gives a stable advantage over the horizon, with at most a slight tendency to narrow.

5.4 Case Study

Setup. We examine the gravitational-wave reconstruction task. Based on the first GW150914 detection paper [1], the agent must recreate the published signal analysis from LIGO strain data. The target has three output groups: H1/L1 waveforms, H1/L1 spectrograms, and velocity/separation curves for the source dynamics. The judge weights the five component scores at 0.15 for each waveform, 0.20 for each spectrogram, and 0.30 for velocity/separation. We run a Codex agent with periodic auto-evaluation, auto-resume, no Internet access, a 30-minute evaluation interval, and a 120-second submission cooldown. The agent made 224 explicit submissions, the harness added 23 auto-evaluations, and the run timed out at the 12-hour budget.

The trajectory reveals a sparse but structured diagnose-edit-evaluate loop. The loop is sparse because most submissions are exploratory probes: only 27 of 224 agent submissions improve the best-so-far score by at least 0.1 percentage points. It is structured because feedback repeatedly changes what the agent searches for. The agent first makes the task measurable, then breaks unresolved errors into smaller searches, then identifies a main bottleneck and keeps searching around it, and finally keeps a working solution while repairing the

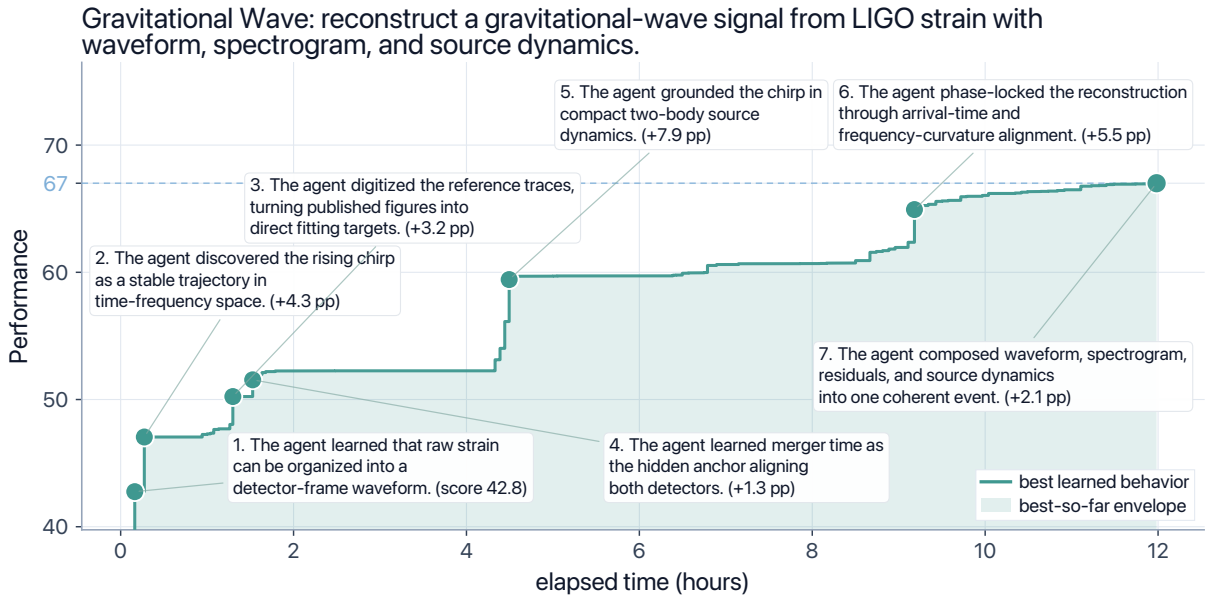


Figure 13 Gravitational-wave reconstruction task case study. The trajectory shows how a GPT-5.5 agent improves over a 12-hour run, with numbered milestones marking representative best-so-far updates.

remaining errors. These patterns show how many failed trials can still produce a small number of cumulative improvements.

- **The agent first makes the problem measurable before making it better.** The first valid submission turns an underspecified analysis task into a scoreable pipeline, but feedback exposes weak source dynamics. The agent then spends the first 11 submissions stabilizing the pipeline and replacing a noisy frequency estimate, producing three meaningful updates and a +4.5 pp gain.
- **When direct repair stalls, the agent decomposes the failure into searchable subproblems.** Instead of treating waveform mismatch as one opaque error, the agent separates reference anchoring, time-frequency localization, and detector alignment. Across 40 signal-search submissions, this decomposition yields seven meaningful updates and lifts the best score to 52.3.
- **Identifying a main bottleneck lets the agent keep searching productively.** After broad signal-processing edits plateau, component feedback identifies velocity/separation as the dominant gap. The agent keeps searching within source-mass calibration rather than rewriting the whole pipeline; in the 4–5h window, 17 submissions and five useful updates raise source dynamics from 64.2 to 89.0, creating the largest jump in the run.
- **After finding a stable solution, the agent keeps the core and repairs only the remaining errors.** In the final hours, most residual edits fail to transfer. Instead of restarting the pipeline, the agent keeps the source model fixed and tests targeted residual corrections, phase alignment, and narrow-band corrections. These useful updates raise the H1 waveform component score from roughly 47 to 95, while the aggregate best score reaches 67.0.

The run improves through uneven jumps rather than a smooth climb. Across 247 scored evaluations, the best score rises from 42.8 to 67.0 on the 0–100 scale. Figure 13 shows seven representative milestones. These jumps correspond to the behavior patterns above: the agent first makes the task scoreable, then localizes the signal, improves source dynamics, and finally repairs the remaining H1 waveform errors. Detailed milestone phases and final subscore composition are reported in Appendix G.2.

6 Related Work

Benchmark	# Tasks	Horizon	Scenario	Self-evolution
MMLU [29]	15,908	Short	Knowledge QA	No
AIME [45]	30	Short	HS math competition	No
GDPval Gold [60]	220	Short	Professional deliverables	No
SWE-bench verified [35, 51]	500	Short	Issue repair	No
Terminal-Bench 2.0 [46]	89	Short	Terminal workflows	No
Continual Learning Bench [4]	6	Short	Controlled task sequences	Yes
CL-bench [21]	1,899	Short	Context learning	No
FrontierCode [41]	150	N/A	Production code changes	No
NL2Repo-Bench [18]	104	Medium	Repo generation	No
Agents' Last Exam [69]	152 public	Medium	Computer-use work	No
MLE-bench [10]	75	Long	ML engineering	Yes
MLS-Bench [42]	140	Long	ML research	Subset
Frontier-Eng [14]	47	Long	Engineering design	No
Frontier-CS [43]	156	Long	Open-ended CS tasks	No
AutoLab [81]	36	Long	Research/engineering optimization	No
FrontierSWE [15]	17	Long	SWE/performance tuning	No
EdgeBench (ours)	134	Ultra-long	SWE, science/ML, professional work, optimization, formal proving, games	Yes

Table 3 Comparison with representative benchmarks. Horizon describes the expected single-instance task contract rather than total suite runtime. A benchmark is counted as measuring self-evolution only when performance is explicitly plotted against a resource axis such as time or sample count.

Existing benchmarks cover major parts of agent capability but rarely measure how an agent improves within a run. Closed-form QA, math, and coding benchmarks such as MMLU [29], AIME [45], and HumanEval [11], together with endpoint patch benchmarks such as SWE-bench [35], evaluate final answers or final patches. More agentic and work-oriented benchmarks, including GDPval [60], Agents' Last Exam [69], and Frontier-Code [41], broaden the task interface to professional deliverables, computer-use workflows, or production code changes. Their central reported quantities, however, remain end-state measures: task success, artifact quality, pass rate, or readiness for production. EdgeBench instead measures the improvement trajectory over time.

Some benchmarks study learning, but they focus on restricted domains and shorter horizons. CL-bench [21], EvaLearn [19], and Continual Learning Bench [4] evaluate whether models improve from a static context or static information streams whose content is not primarily shaped by the agent's actions within a single task. Iterative optimization benchmarks such as MLE-bench [10], MLS-Bench [42], Frontier-Eng [14], Frontier-CS [43], and ALE-Bench [34] further include repeated attempts, empirical feedback, or visible metrics. The closest long-horizon agentic comparators are AutoLab [81] and FrontierSWE [15]: both evaluate agents that repeatedly edit executable artifacts and incorporate feedback. EdgeBench differs by covering a broader range of executable domains, using a day-scale task contract, and applying the same trajectory metrics consistently across all tasks.

Scaling laws have been widely studied in prior work, but learning from diverse real-world environments remains underexplored. Classical scaling laws study pretraining, relating loss or benchmark performance to model size, data, and compute [32, 36]; later work studies bounded benchmark performance curves as model scale increases [7, 59, 64, 85]. Test-time scaling laws have also been studied across several inference-time methods: plain repeated sampling [9, 11, 40], search, revision, or verifier-guided compute allocation [67, 79], and long-chain-of-thought inference in reasoning models [16, 50]. Agentic test-time scaling has been observed in computer-use and browsing agents [52, 72], studied through interaction-length scaling [65], and revisited through long-horizon human-agent comparisons [44]. These studies are closer to our setting but cover narrower domains or fewer tasks and do not establish a cross-domain scaling law. Reinforcement-learning scaling is also relevant because it studies learning from environment feedback [31, 37], while EdgeBench measures

elapsed interaction time in deployed agent trajectories. In-context learning from task feedback is relatively inexpensive to evaluate repeatedly across many executable environments, while large reinforcement-learning runs are costly and typically cover fewer environments. This broader environment coverage may explain why the aggregate curves are stable enough to reveal a scaling law.

A more detailed benchmark-by-benchmark discussion is provided in Appendix F.

7 Conclusion

This paper introduced EdgeBench, a benchmark for studying how agents learn from real-world environments over day-long horizons. Across 134 diverse executable tasks and roughly 38,000 hours of environment interaction, we find that aggregate learning trajectories follow a precise log-sigmoid relationship with interaction time. The same form appears across task families, remains stable over longer horizons, and supports forecasting later performance from early trajectories. We also find that agent learning speed has improved rapidly across recent frontier model generations.

These results suggest that learning from environments is not merely a collection of idiosyncratic task outcomes, but a measurable scaling object. Unlike many aggregate capability curves, environment-learning trajectories expose the intermediate attempts, feedback, and revisions through which progress occurs. This makes EdgeBench useful not only for ranking agents, but for studying how agents acquire and reuse experience. More broadly, the regularity we observe suggests that post-deployment learning from rich environments may deserve the same systematic scaling attention that pretraining has received.

Author Contributions

Core Contributors

Deyao Zhu[*]
Xin Zhou[*]
Shengling Qin[*]
Xuekai Zhu[*]
Hangliang Ding[*]
Shu Zhong[*†]

Theory

Zixin Wen

Data Collection and Curation

Zhonglin Xie
Chenhui Gou
Linxuan Ren
Yueyang Wang
Junfeng Zhong
Rui Liu
Tian Gao
Yangguang Lin
Jingyuan Zhang
Maojia Song
Xuan Qi[‡]
Jinhong Wu[‡]
Chenyang Zhang[‡]
Yinzhu Piao
Ziru Niu
Hongbin Lin
Lingxiang Meng
Pengtang Tang
Chengyao Tang
Shanyu Wu
Huanyu Zheng
Yu Liu
Liya Zhu
He Wang
Ming Ding

Data Infrastructure

Ziyu Wan
Hao Liu
Sibo Wang
Haotian Zhu
Xintian Zhang
Nan Chai
Yipeng Liu
Panhao Lai

Referrals

Sihang Yuan
Zixin Su
Ge Zhang
Wangchunshu Zhou
Yantao Du

Advisors

Wenhao Huang
Guang Shi

[*]Equal contribution [†]Project Lead [‡]External contributor

References

[1] B. P. Abbott et al. Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, 116(6):061102, 2016. doi: 10.1103/PhysRevLett.116.061102.

[2] Anthropic. The Claude Model Family. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.5_Addendum.pdf, 2024.

[3] Anthropic. Claude Opus 4.8 System Card. https://www-cdn.anthropic.com/0b4915911bb0d19eca5b5ee635c80fef830a37ea.pdf, May 2026. System card. Released May 28, 2026; updated June 3, 2026.

[4] Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, et al. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026.

[5] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality: An explanation of 1/f noise. *Physical Review Letters*, 59(4):381–384, 1987.

[6] Joseph Berkson. Application of the logistic function to bio-assay. *Journal of the American Statistical Association*, 39(227):357–365, 1944. doi: 10.1080/01621459.1944.10500699.

[7] Akshita Bhagia, Jiacheng Liu, Alexander Wettig, David Heineman, Oyvind Tafjord, Ananya Harsh Jha, Luca Soldaini, Noah A. Smith, Dirk Groeneveld, Pang Wei Koh, Jesse Dodge, and Hannaneh Hajishirzi. Establishing task scaling laws via compute-efficient model ladders. *arXiv preprint arXiv:2412.04403*, 2024.

[8] Chester I. Bliss. The method of probits. *Science*, 79(2037):38–39, 1934.

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

[10] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025.

[11] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[12] Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, et al. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025.

[13] Ching-An Cheng, Andrey Kolobov, Dipendra Misra, Allen Nie, and Adith Swaminathan. LLF-bench: Benchmark for interactive learning from language feedback. *arXiv preprint arXiv:2312.06853*, 2023.

[14] Yizhe Chi, Deyao Hong, Dapeng Jiang, Tianwei Luo, Kaisen Yang, Boshi Zhang, et al. Frontier-Eng: Benchmarking self-evolving agents on real-world engineering tasks with generative optimization. *arXiv preprint arXiv:2604.12290*, 2026.

[15] Evan Chu, Rajan Agarwal, Abishek Thangamuthu, Brendan Graham, and Justus Mattern. FrontierSWE: Benchmarking coding agents at the limits of human abilities. https://www.frontierswe.com/blog, 2026.

[16] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[17] DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf, April 2026. Technical report.

[18] Jingzhe Ding, Shengda Long, Changxin Pu, et al. NL2Repo-Bench: Towards long-horizon repository generation evaluation of coding agents. *arXiv preprint arXiv:2512.12730*, 2025.

[19] Shihan Dou, Ming Zhang, Chenhao Huang, Jiayi Chen, Feng Chen, Shichun Liu, Yan Liu, Chenxiao Liu, Cheng Zhong, Zongzhang Zhang, et al. EvaLearn: Quantifying the learning capability and efficiency of LLMs via sequential problem solving. *arXiv preprint arXiv:2506.02672*, 2025.

[20] Shihan Dou, Yujiong Shen, Chenhao Huang, Junjie Ye, Jiayi Chen, Junzhe Wang, Qianyu He, Shichun Liu, Changze Lv, Jiahang Lin, et al. CL-bench Life: Can language models learn from real-life context? *arXiv preprint arXiv:2604.27043*, 2026.

[21] Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. CL-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

[22] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[23] Darius A. Faroughy, Sofia Palacios Schweitzer, Ian Pang, Siddharth Mishra-Sharma, and David Shih. Collider-Bench: Benchmarking AI agents with particle physics analysis reproduction. *arXiv preprint arXiv:2605.13950*, 2026.

[24] Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[25] Gemini Team. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[26] GLM-5 Team. GLM-5: from vibe coding to agentic engineering. https://arxiv.org/abs/2602.15763, 2026.

[27] Benjamin Gompertz. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. *Philosophical Transactions of the Royal Society of London*, 115: 513–583, 1825.

[28] Harry F. Harlow. The formation of learning sets. *Psychological Review*, 56(1):51–65, 1949. doi: 10.1037/h0062474.

[29] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

[30] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems*, 2021.

[31] Jacob Hilton, Jie Tang, and John Schulman. Scaling laws for single-agent reinforcement learning. *arXiv preprint arXiv:2301.13442*, 2023.

[32] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[33] Geoffrey Huntley. Everything Is a Ralph Loop. https://ghuntley.com/loop/, January 2026. Blog post, January 17, 2026.

[34] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-bench: A benchmark for long-horizon objective-driven algorithm engineering. *arXiv preprint arXiv:2506.09050*, 2025.

[35] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[36] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[37] Devvrit Khatri, Lovish Madaan, Rishabh Tiwari, Rachit Bansal, Sai Surya Duvvuri, Manzil Zaheer, Inderjit S. Dhillon, David Brandfonbrener, and Rishabh Agarwal. The art of scaling reinforcement learning compute for LLMs. *arXiv preprint arXiv:2510.13786*, 2025.

[38] Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring AI ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 2025.

[39] Nathaniel Leibowitz, Barak Baum, Giora Enden, and Amir Karniel. The exponential learning equation as a function of successful trials results in sigmoid performance. *Journal of Mathematical Psychology*, 54(3):338–340, 2010. doi: 10.1016/j.jmp.2010.01.006.

[40] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *arXiv preprint arXiv:2203.07814*, 2022.

[41] Eric Lu, Ben Pan, Deniz Birlikci, Sam Lee, Ray Wang, Rohan Choudhury, Fermi Ma, TC Qin, Carlo Baronio, Silas Alberti, et al. Introducing FrontierCode. https://cognition.ai/blog/frontier-code, 2026.

[42] Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, et al. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI. *arXiv preprint arXiv:2605.08678*, 2026.

[43] Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, et al. FrontierCS: Evolving challenges for evolving intelligence. *arXiv preprint arXiv:2512.15699*, 2025.

[44] Qiuyang Mang, Kaiyuan Liu, Bo Peng, Shreyas Pimpalgaonkar, Alex Dimakis, and Alvin Cheung. Humans still beat AI in the long horizon: Revisiting test-time scaling in the agent era. https://joyemang33.github.io/blog/2026/humans-dont-just-sample/, 2026.

[45] Mathematical Association of America. MAA invitational competitions: American invitational mathematics examination (AIME). https://maa.org/maa-invitational-competitions/, 2026.

[46] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *The Fourteenth International Conference on Learning Representations*, 2026.

[47] Jaap M. J. Murre. S-shaped learning curves. *Psychonomic Bulletin & Review*, 21(2):344–356, 2014. doi: 10.3758/s13423-013-0522-0.

[48] Mark E. J. Newman. Power laws, pareto distributions and zipf's law. *Contemporary Physics*, 46(5):323–351, 2005.

[49] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[50] OpenAI. Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, 2024.

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

[52] OpenAI. Computer-using agent. https://openai.com/index/computer-using-agent/, 2025.

[53] OpenAI. GPT-4.5 System Card. https://cdn.openai.com/gpt-4-5-system-card-2272025.pdf, 2025.

[54] OpenAI. Update to GPT-5 System Card: GPT-5.2. https://cdn.openai.com/pdf/3a4153c8-c748-4b71-8e31-aecbde944f8d/oai_5_2_system-card.pdf, 2025.

[55] OpenAI. GPT-5 System Card. *arXiv preprint arXiv:2601.03267*, 2025.

[56] OpenAI. Follow a Goal. https://developers.openai.com/codex/use-cases/follow-goals, 2026. Codex documentation. Accessed June 18, 2026.

[57] OpenAI. GPT-5.4 Thinking System Card. https://deploymentsafety.openai.com/gpt-5-4-thinking/gpt-5-4-thinking.pdf, March 2026. System card. Published March 5, 2026.

[58] OpenAI. GPT-5.5 System Card. https://deploymentsafety.openai.com/gpt-5-5/gpt-5-5.pdf, April 2026. System card. Published April 23, 2026.

[59] David Owen. How predictable is language model benchmark performance? *arXiv preprint arXiv:2401.04757*, 2024.

[60] Tejal Patwardhan, Rachel Dias, Elizabeth Proehl, Grace Kim, Michele Wang, Olivia Watkins, Simon Posada Fishman, Marwan Aljubeh, Phoebe Thacker, Laurance Fauconnet, et al. GDPval: Evaluating AI model performance on real-world economically valuable tasks. *arXiv preprint arXiv:2510.04374*, 2025.

[61] Shi Qiu, Junyi Deng, Yiwei Deng, Haoran Dong, Jieyu Fu, Mao Li, Zeyu Li, Zhaolong Zhang, Huiwen Zheng, Leidong Bao, et al. PRBench: End-to-end paper reproduction in physics research. *arXiv preprint arXiv:2603.27646*, 2026.

[62] 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.

[63] David Rein, Joel Becker, Amy Deng, Seraphina Nix, Chris Canal, Daniel O'Connel, Pip Arnott, Ryan Bloom, Thomas Broadley, et al. HCAST: Human-calibrated autonomy software tasks. *arXiv preprint arXiv:2503.17354*, 2025.

[64] Yangjun Ruan, Chris J. Maddison, and Tatsunori Hashimoto. Observational scaling laws and the predictability of language model performance. *arXiv preprint arXiv:2405.10938*, 2024.

[65] Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. Thinking vs. doing: Agents that reason by scaling test-time interaction. *arXiv preprint arXiv:2506.07976*, 2025.

[66] Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024.

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

[68] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 56843–56873. PMLR, 2025.

[69] Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, et al. Agents' last exam. *arXiv preprint arXiv:2606.05405*, 2026.

[70] Minh V. T. Thai, Tue Le, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios. *arXiv preprint arXiv:2512.18470*, 2025.

[71] Louis Leon Thurstone. *The Learning Curve Equation*. Psychological Review Company, Princeton, NJ, 1919.

[72] Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. BrowseComp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*, 2025.

[73] Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, et al. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

[74] Waloddi Weibull. A statistical distribution function of wide applicability. *Journal of Applied Mechanics*, 18(3): 293–297, 1951.

[75] Pierre Weiss. L'hypothèse du champ moléculaire et la propriété ferromagnétique. *Journal de Physique Théorique et Appliquée*, 6(1):661–690, 1907.

[76] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. RE-bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

[77] Wikipedia contributors. Sigmoid function. https://en.wikipedia.org/wiki/Sigmoid_function, 2025.

[78] Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung yi Lee. StreamBench: Towards benchmarking continuous improvement of language agents. In *Advances in Neural Information Processing Systems*, 2024.

[79] Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

[80] Xinbo Xu, Ruihan Yang, Haiyang Shen, et al. RoadmapBench: Evaluating long-horizon agentic software development across version upgrades. *arXiv preprint arXiv:2605.15846*, 2026.

[81] Zhangchen Xu, Junda Chen, Yue Huang, Dongfu Jiang, Jiefeng Chen, Hang Hua, Zijian Wu, Zheyuan Liu, Zexue He, Lichi Li, et al. AutoLab: Can frontier models solve long-horizon auto research and engineering tasks? *arXiv preprint arXiv:2606.05080*, 2026.

[82] John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. ProgramBench: Can language models rebuild programs from scratch? *arXiv preprint arXiv:2605.03546*, 2026.

[83] Christine Ye, Sihan Yuan, Suchetha Cooray, Steven Dillmann, Ian L. V. Roque, Dalya Baron, Philipp Frank, Sergio Martin-Alvarez, Nolan Koblischke, Frank J. Qu, et al. ReplicationBench: Can AI agents replicate astrophysics research papers? *arXiv preprint arXiv:2510.24591*, 2025.

[84] Z.ai. GLM-5.1: Towards long-horizon tasks. https://z.ai/blog/glm-5.1, 2026. Model release blog and model card.

[85] Hanlin Zhang, Jikai Jin, Vasilis Syrgkanis, and Sham Kakade. Prescriptive scaling reveals the evolution of language model capabilities. *arXiv preprint arXiv:2602.15327*, 2026.

[86] Zhirui Zhang, Hongbo Zhang, Haoxiang Fei, et al. SWE-AGI: Benchmarking specification-driven software construction with MoonBit in the era of autonomous agents. *arXiv preprint arXiv:2602.09447*, 2026.

[87] Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Appendix

Appendix Contents

A Evaluation Harness

A standard unit-test harness is not enough for day-long runs. The benchmark must hide evaluation assets, support repeated submissions over many hours, measure progress even when agents do not explicitly submit, and run reliably on both local machines and clusters. We built **SForge** around three mechanisms: isolated work and judge environments, a feedback loop modeled on online judges, and progress tracking on the host.

Two-container architecture. Each task is materialized as two task-specific Docker images:

- The **work image** contains the skeleton codebase, documentation, and local validation tools, but no hidden evaluation assets. The agent operates only in this environment.
- The **judge image** contains the hidden evaluation assets, grading scripts, and evaluation commands. It is never exposed to the agent.

At evaluation time, the agent's code is copied into an ephemeral judge container, which runs the hidden tests, returns only task-defined structured feedback, and is then destroyed. This separation prevents agents from inspecting or modifying the hidden grader while still allowing realistic iterative development.

Judge server and the outer loop. The outer loop is mediated by a host-side HTTP judge server. The agent submits its current solution to the server, which queues the submission, runs the judge container, parses the result, and returns feedback such as pass rate, score, per-test verdicts, or diagnostics. For long-running evaluations, the server supports asynchronous grading, allowing agents to continue working while submitted jobs are being judged. The server also enforces submission budgets, cooldown intervals, and session authentication.

Long-horizon execution and measurement. SForge combines host-side auto-eval with stop-hook and auto-resume mechanisms to support day-scale runs. Auto-eval periodically snapshots and evaluates the agent workspace through a privileged channel; these results are recorded for trajectory analysis but are not shown to the agent. The stop hook and auto-resume mechanism reduce premature truncation from voluntary exits, crashes, context limits, or transient API failures.

Operational support and safeguards. SForge also provides practical infrastructure for running and auditing large-scale experiments. A web dashboard records score trajectories, submission histories, diffs, and conversation traces for live monitoring and post-hoc comparison. The same task specification can run on either a local Docker backend or a Kubernetes backend for cluster-scale evaluation. To preserve benchmark integrity, SForge combines work–judge isolation with submission filtering, network isolation, and session-scoped authentication, so agents can receive feedback without access to hidden tests, privileged history, or auto-eval results. Appendix C describes task-design failure modes we observed during benchmark development and the corresponding mitigations.

B Serving and API Stability

Sustaining a single agent for 12 hours or more stresses API serving far more than short evaluations do: a day-long run must keep the model, its context, and its tool calls available without interruption, and any serving-side incident in that window can truncate or degrade the trajectory. Our long-horizon tasks therefore unavoidably fold serving stability into what they measure. We view this as appropriate rather than a confound to be engineered away: an agent deployed to work for hours in the real world is subject to exactly these serving realities, so a benchmark for long-horizon learning should reflect them too.

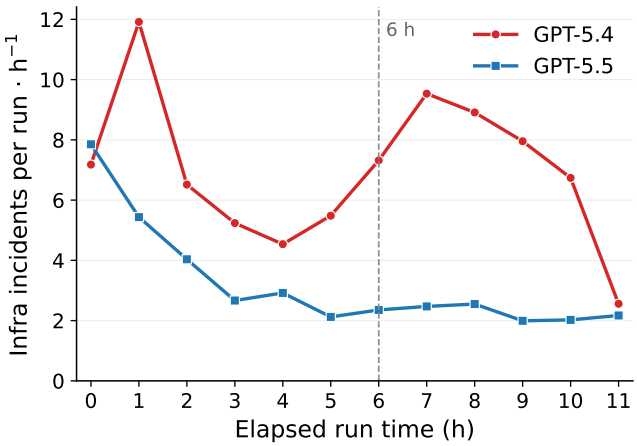


Figure 14 Serving-side incident rate during long-horizon GPT-5.4 and GPT-5.5 runs. Incidents are normalized by active run time; the dashed line marks six elapsed hours. GPT-5.4 experienced substantially more infrastructure and API interruptions, especially in the later part of the run.

Figure 14 quantifies this for the two GPT models, plotting the serving-side incident rate (normalized by active run time) against elapsed time: GPT-5.4 saw substantially more infrastructure and API interruptions than GPT-5.5, especially after the six-hour mark. This carries two caveats for GPT-5.4. First, several of the per-task cells with fewer than three valid runs (marked with $*$ in the score tables) involve GPT-5.4, so its lower run coverage partly reflects serving reliability rather than model behavior, though the reported scores still use only valid trajectories. Second, it offers an operational explanation for the forecast deviation flagged in Section 3.2 (Figure 7): with availability dropping in the second half of the run, GPT-5.4's later trajectory is noisier and pulls away from the log-sigmoid forecast fit to its first 6.5 hours.

C Evaluation Hacking

Long-horizon evaluation gives agents many opportunities to adapt to evaluator feedback. During task construction and adversarial stress tests, we audited traces for strategies that raised measured scores without exercising the intended capability. These observations are development diagnostics, not official model results: affected tasks were revised or excluded, and the cases below motivated safeguards in the final benchmark.

Feedback as an oracle for hidden answers. In `cylinder_wake_prediction`, an agent treated per-case absolute errors as equations and reconstructed the hidden targets through more than 400 submissions. A lookup table then scored 1.000 without solving the underlying fluid dynamics problem, whereas the best observed submission based on a physics model scored 0.165.

Optimizing stochastic upper tails. In `nethack_dungeon_agent`, an agent removed a fixed random seed after recognizing that higher variance increased the chance of a favorable evaluation. Across 311 submissions, its best score was 1,501 while its mean was 484, showing how best-of-N selection can reward repeated sampling as well as policy quality.

Overfitting evaluator seeds. In `bipedalwalker_locomotion_rl`, the development judge initially used a single deterministic episode, allowing agents to infer and optimize for the reused seed rather than expected return. One run reached a Hardcore return of 301.5 on the judge seed but averaged about 12 over a local 100-episode evaluation; another searched a small hand-coded controller directly against the judge formula. This motivated hidden multi-seed evaluation for stochastic control tasks.

Crossing a trust boundary. In `autolifter`, an agent discovered that anti-cheat checks exempted the repository's `baseline/` directory and moved an implementation based on an oracle into that trusted path. It

solved 82 of 84 hidden cases and scored 0.980; the strongest observed submission that followed the intended synthesis route scored 0.121.

Online answer lookup. Publicly indexed data, task artifacts, or reference implementations can turn an intended reasoning problem into retrieval. In `stock_momentum_backtest`, an agent attempted Web search for target data, but network isolation for the task blocked the requests. We include this as a prevented risk rather than a successful exploit.

Mitigation. These findings led us to harden both task design and infrastructure. We revised or excluded affected tasks, reduced or aggregated feedback that could reveal hidden targets, enforced submission budgets and cooldowns, evaluated stochastic behavior over hidden seed sets, extended integrity checks across writable paths, and disabled network access for tasks where public data or reference solutions could reveal the answer. These controls do not eliminate every adaptive attack, but they reduce the channels identified during task development.

D A Comprehensive Derivation of the Log-Sigmoid Law

This appendix gives a comprehensive derivation of a sufficient mechanism for the empirical log-sigmoid law observed in the *task-aggregate* environment-learning curve. We assume the score of a single task is observed through finitely many visible *score units*, so its best-so-far curve can be a jagged staircase with long plateaus and abrupt breakthroughs. These irregularities are the expected finite-resolution behavior of individual tasks. Under explicit cut-mixing, granularity, midpoint-alignment, and speed-concentration assumptions, we show how averaging many independently evaluated task scores can yield a smooth log-sigmoid limit.

Our central modeling assumption: **environment learning is a frontier expansion process on the task graph**. We model a task environment as a latent graph whose nodes are small score units, and the environment learning process can be viewed as frontier expansion on this graph. Our theory provides a clear path from frontier expansion process to the observed log-sigmoid law.

The exposition of the derivation follows five steps:

1. **Preliminaries.** Define the model-conditioned task graph, score units, score measure μ, influence matrix K, its support graph $\mathcal{E}_K$, and the capability field.
2. **Environment learning as a frontier expansion process.** The model improves its outputs based on feedback it has previously received. Within one task, the conditional expected score-growth rate is a weighted cut from unlocked score nodes to locked score nodes of the task graph.
3. **Single-task curves range from jagged to a smooth limit.** Weighted cut mixing and small score units turn the jagged process into a logistic ordinary differential equation in the many-unit limit.
4. **Many-task aggregate reveals the log-sigmoid law.** Even though the expansion process on tasks with finite score units remain jagged processes, the benchmark-aggregated curve converges to a smooth log-sigmoid law.
5. **Graph self-similarity induces a log scale for time.** Finally, we show that when the task graph displays self-similar structure at different resolutions, frontier expansion naturally exhibits a log-time scale.

Throughout the section, we denote raw interaction time by $t > 0$, and the raw task/benchmark score by $S(t)$. Let $t_{\mathrm{mid}} > 0$ and $S_{\max} > 0$ be fitted parameters. The corresponding log-time coordinate and normalized score are

$$u = \log t - \log t_{\mathrm{mid}}, \qquad x(u) = \frac{S(t)}{S_{\max}}$$

We show how the following normalized log-sigmoid dynamics, with frontier speed β (also a fitted parameter),

can arise as a natural many-task limit of the averaged benchmark score:

$$\frac{\mathrm{d}x(u)}{\mathrm{d}u} = \beta x(u)(1 - x(u)), \qquad x\left(\log \frac{t}{t_{\mathrm{mid}}}\right) = \frac{1}{1 + (t_{\mathrm{mid}}/t)^{\beta}}.$$

The value at raw time $t = 0$ is understood as the boundary limit $u \to -\infty$.

D.1 Preliminaries: Task Graph and the Attainable Support

We start with one task and one fixed model. The main assumption is that score units are the nodes of an *agent-conditioned task graph*: unlocked nodes supply an influence field, locked nodes receive this field through directed influence edges, and a locked node can become unlocked at a rate proportional to the field strength.

Definition D.1 (Task graph)**.** Let E be the set of visible score-unit nodes for the task. The language model induces a nonnegative influence matrix

$$K = (K_{ij})_{i,j \in E}, \qquad K_{ij} \geq 0.$$

Its directed support graph has edge set $\mathcal{E}_K = \{j \to i : K_{ij} > 0\}$ where $j \to i$ means that unlocked source node j can help unlock target node i. We write $G_K = (E, \mathcal{E}_K)$ for the resulting task graph.

If one begins with a larger ambient graph, the restriction to the model-relevant part is made before writing E and K; empirically, that restriction is absorbed into the fitted ceiling $S_{\max}$. Next we introduce the score units and the corresponding task influence field.

Definition D.2 (Score units and normalized score measure)**.** On the latent graph $G_K = (E, \mathcal{E}_K)$, each node $i \in E$ represents a score unit with weight $\omega_i \geq 0$. The normalized score measure is the probability measure

$$W = \sum_{i \in E} \omega_i > 0, \qquad \mu_i = \frac{\omega_i}{W}, \qquad \mu(A) = \sum_{i \in A} \mu_i, \quad \forall A \subseteq E.$$

Definition D.3 (Unlock state, unlocked and locked sets)**.** The unlock state of node i at log-time u is $n_i(u) \in \{0, 1\}$, and once a node is unlocked, it cannot be locked again. The normalized score, unlocked set, and locked set are defined by

$$x(u) = \sum_{i \in E} \mu_i n_i(u), \qquad U(u) = \{j \in E : n_j(u) = 1\}, \qquad L(u) = \{i \in E : n_i(u) = 0\}. \tag{5}$$

For a static state $n = (n_i)_{i \in E}$, we also write

$$U(n) = \{j \in E : n_j = 1\}, \qquad L(n) = \{i \in E : n_i = 0\}, \qquad x(n) = \mu(U(n)).$$

Definition D.4 (Task influence field)**.** The entry K_{ij} is the influence strength from source node j to target node i. The capability field on target node i is the incoming-edge sum

$$h_i(n) = \sum_{j : j \to i} K_{ij} n_j = \sum_{j \in E} K_{ij} n_j.$$

When stating many-unit limits, we restore the index N: the objects above become E_N, $\mu^{(N)}$, $K^{(N)}$, $n^{(N)}(u)$, $x_N(u)$, $U_N(u)$, $L_N(u)$, and

$$\mu_N(A) = \sum_{i \in A} \mu_i^{(N)}, \qquad x_N(u) = \sum_{i \in E_N} \mu_i^{(N)} n_i^{(N)}(u), \qquad h_i^{(N)}(u) = \sum_{j \in E_N} K_{ij}^{(N)} n_j^{(N)}(u).$$

For a static state $n \in \{0, 1\}^{E_N}$, we write

$$U_N(n) = \{j \in E_N : n_j = 1\}, \qquad L_N(n) = \{i \in E_N : n_i = 0\}, \qquad x_N(n) = \mu_N(U_N(n)).$$

D.2 Environment Learning as a Frontier Expansion Process

With the latent graph and score measure fixed, we now turn the story into a process. The reason to use a frontier picture is that long-horizon environment learning is cumulative. In a software task, a runnable baseline makes later debugging meaningful; in a proof task, a lemma makes later proof obligations easier; in a scientific task, a calibrated model or validation loop makes later parameter search more useful. More generally, each partial success can become a tool for later progress.

The graph model encodes which already-solved score units can help unlock which remaining ones. At log time u, let

$$U(u) = \{j : n_j(u) = 1\}, \qquad L(u) = \{i : n_i(u) = 0\}$$

denote the unlocked and locked sets. A locked unit i can only become usable through influence arriving from units in $U(u)$. Thus the relevant quantity is not the total amount of unlocked score, but the amount of influence crossing the frontier from $U(u)$ into $L(u)$.

Probabilistic frontier expansion. For a locked score unit i, its task field is given by $h_i(u) = \sum_{j \in E} K_{ij} n_j(u)$. We turn the influence of the field into a continuous-time stochastic process by assuming that unit i unlocks with hazard proportional to its accumulated field:

$$\eta(1 - n_i(u))h_i(u) = \eta(1 - n_i(u)) \sum_{j \in E} K_{ij} n_j(u),$$

where $\eta > 0$ converts influence strength into progress per unit log time. Thus the field does not deterministically unlock i; it determines the instantaneous probability that i unlocks. This is the sense in which learning expands the frontier: locked nodes become available at rates determined by the influence they receive from the currently unlocked side of the graph. We formalize this process in the definition below.

Definition D.5 (Single-task frontier process)**.** Fix a finite task graph $G_K = (E, \mathcal{E}_K)$, score weights μ_i, influence matrix K, and conversion constant $\eta > 0$. The single-task frontier process is the continuous-time process on $\{0,1\}^E$ in which, for each score unit i, the instantaneous unlocking intensity is

$$\begin{aligned} \lambda_i(u) &:= \lim_{\Delta u \downarrow 0} \frac{1}{\Delta u} \mathbb{P}\left(n_i(u + \Delta u) - n_i(u) = 1 \mid n(u)\right) \\ &= \eta(1 - n_i(u)) \sum_{j \in E} K_{ij} n_j(u). \end{aligned} \tag{6}$$

Importantly, once a node unlocks, it remains unlocked.

To express the resulting score growth, define for $A, B \subseteq E$ the weighted frontier cut

$$C(A, B) = \sum_{i \in A} \sum_{j \in B} \mu_i K_{ij}.$$

This is the total influence from sources in B to targets in A, with each target weighted by the score gained if it unlocks. The active frontier at time u is therefore $C(L(u), U(u))$.

Lemma D.1 (Exact single-task frontier growth rate)**.** *For the single-task frontier process,*

$$\frac{\mathrm{d}}{\mathrm{d}u} \mathbb{E}\big[x(u) \mid n(u)\big] = \eta C(L(u), U(u)).$$

Proof. Fix u and condition on the state $n(u) = n$. For $i \in E$, let $A_i(\Delta u)$ denote the event that node i unlocks during $[u, u + \Delta u]$. By the definition of the unlocking intensity,

$$\Pr(A_i(\Delta u) \mid n(u) = n) = \eta(1 - n_i) \sum_{j \in E} K_{ij} n_j \, \Delta u + o(\Delta u).$$

Since E is finite, the probability of two or more unlocking events in $[u, u+\Delta u]$ is $o(\Delta u)$. Therefore, to first order in Δu, the expected score increment is obtained by summing the score gain from each possible single-node unlock:

$$\begin{aligned}\mathbb{E}[x(u+\Delta u) - x(u) \mid n(u) = n] &= \sum_{i \in E} \mu_i \Pr(A_i(\Delta u) \mid n(u) = n) + o(\Delta u) \\ &= \eta \sum_{i \in E} \mu_i (1 - n_i) \sum_{j \in E} K_{ij} n_j \, \Delta u + o(\Delta u).\end{aligned}$$

Dividing by Δu and letting $\Delta u \downarrow 0$ gives

$$\frac{\mathrm{d}}{\mathrm{d}u} \mathbb{E}[x(u) \mid n(u) = n] = \eta \sum_{i \in E} \mu_i (1 - n_i) \sum_{j \in E} K_{ij} n_j.$$

Since $1 - n_i$ restricts the outer sum to $i \in L(n)$, and n_j restricts the inner sum to $j \in U(n)$, this becomes

$$\frac{\mathrm{d}}{\mathrm{d}u} \mathbb{E}[x(u) \mid n(u) = n] = \eta \sum_{i \in L(n)} \sum_{j \in U(n)} \mu_i K_{ij} = \eta C(L(n), U(n)).$$

This proves the claim. □

Lemma D.1 is the key reduction. It says that, to obtain a logistic expected growth rate, we do not need every microscopic edge weight to be identical. Instead, we need the boundary influence to depend mainly on two coarse quantities: the unlocked mass $\mu(U)$ and the locked mass $\mu(L)$. Then the field is roughly the product of the two: available reusable capability times remaining score opportunity.

As the number of score units increases, the same fixed-resolution objects are now applied to the size-N graph. In particular,

$$C_N(A, B) = \sum_{i \in A} \sum_{j \in B} \mu_i^{(N)} K_{ij}^{(N)}, \qquad A, B \subseteq E_N.$$

From frontier cuts to product frontiers. The remaining modeling question is when this boundary influence should look like a product of the two measures. The following quantity controls the deviation of the frontier process from the complete-mixing case.

Definition D.6 (Single-task weighted cut error)**.** Given $(E_N, \mu^{(N)}, K^{(N)})$ and an arbitrary $\kappa > 0$, define

$$\varepsilon_N(\kappa) = \sup_{A, B \subseteq E_N} \left| \sum_{i \in A} \sum_{j \in B} \mu_i^{(N)} \left(K_{ij}^{(N)} - \kappa \mu_j^{(N)} \right) \right|.$$

Condition D.1 (Single-task weighted cut mixing)**.** There exists a $\kappa > 0$ such that, as $N \to \infty$ for the task graph, the weighted cut error in Definition D.6 satisfies

$$\varepsilon_N \to 0.$$

Weighted cut mixing is weaker than entrywise complete mixing. It does not say that each $K_{ij}^{(N)}$ is close to $\kappa \mu_j^{(N)}$. It says that, for every possible frontier, the aggregate capability crossing from unlocked graph nodes into locked graph nodes is close to the product-measure value.

Lemma D.2 (Cut mixing gives logistic frontier growth rate)**.** *Under Condition D.1, the following holds for every static state $n \in \{0,1\}^{E_N}$. Let $U_N = U_N(n)$, $L_N = L_N(n)$, and $x_N = x_N(n)$. Then*

$$|C_N(L_N, U_N) - \kappa x_N (1 - x_N)| \le \varepsilon_N.$$

Consequently,

$$b_N(n) = \eta \kappa x_N (1 - x_N) + r_N(n), \qquad |r_N(n)| \le \eta \varepsilon_N.$$

Proof. Apply Definition D.6 to $A = L_N$ and $B = U_N$. Since $\mu_N(U_N) = x_N$ and $\mu_N(L_N) = 1 - x_N$,

$$|C_N(L_N, U_N) - \kappa x_N(1 - x_N)| = \left| \sum_{i \in L_N} \sum_{j \in U_N} \mu_i^{(N)} \big(K_{ij}^{(N)} - \kappa \mu_j^{(N)}\big) \right| \leq \varepsilon_N.$$

The expected-growth-rate statement follows from Lemma D.1. □

Cut mixing controls the deterministic expected growth rate. A separate condition controls the visible jump size. This distinction is crucial for the interpretation of the paper: a finite task with only a few large score units can follow the right frontier mechanism and still look very jagged.

Condition D.2 (Small score units and controlled fields)**.** Let

$$q_N = \sum_{i \in E_N} \big(\mu_i^{(N)}\big)^2.$$

Assume $q_N \to 0$. Assume also that there is a deterministic H_N such that

$$\sup_{i \in E_N,\, n \in \{0,1\}^{E_N}} \sum_{j \in E_N} K_{ij}^{(N)} n_j \leq H_N, \qquad H_N q_N \to 0.$$

For equal score units, $q_N = 1/N$. Thus Condition D.2 captures the smoothening process from a task with few bulky score units to a task with many small score units. The theorem below should be read exactly in that way. It is not a claim that the realized best-so-far curve of a finite benchmark task must be visually smooth. Rather, it says that if the same task-level mechanism were observed at increasingly fine score resolution, the staircase process would converge to a logistic frontier.

Theorem D.1 (Single-task frontier, many-unit limit)**.** *Consider the single-task frontier process of Definition D.5. Let $[u_0, u_1]$ be a compact log-time interval and let $x_0 \in [0, 1]$. Suppose Conditions D.1 and D.2 hold, and suppose $x_N(u_0) \to x_0$. Then x_N converges uniformly in probability on $[u_0, u_1]$ to the solution of*

$$\frac{dx}{du} = \eta\kappa x(1 - x), \qquad x(u_0) = x_0. \tag{7}$$

If t_{mid} is chosen so that the limiting midpoint satisfies $x(0) = 1/2$, then, with $x(t) := x(\log(t/t_{\mathrm{mid}}))$, we obtain the log-sigmoid law:

$$x_N \xrightarrow{P} x(t) = \frac{1}{1 + (t_{\mathrm{mid}}/t)^{\beta}}, \qquad \beta = \eta\kappa. \tag{8}$$

Proof. Let $M_N(u)$ denote the martingale increment in the Doob-Meyer decomposition, normalized so that $M_N(u_0) = 0$. The Doob-Meyer decomposition and Lemma D.2 give, for $u \in [u_0, u_1]$,

$$x_N(u) = x_N(u_0) + M_N(u) + \eta\kappa \int_{u_0}^{u} x_N(s)(1 - x_N(s))\, ds + R_N(u), \tag{9}$$

where

$$\sup_{u \in [u_0, u_1]} |R_N(u)| \leq \eta(u_1 - u_0)\varepsilon_N.$$

A jump of unit i changes x_N by $\mu_i^{(N)}$. The predictable quadratic variation accumulated over the interval $[u_0, u_1]$ is therefore bounded by

$$\langle M_N \rangle_{[u_0, u_1]} = \int_{u_0}^{u_1} \sum_{i \in E_N} \big(\mu_i^{(N)}\big)^2 \eta\big(1 - n_i^{(N)}(s)\big) \sum_{j \in E_N} K_{ij}^{(N)} n_j^{(N)}(s)\, ds \leq \eta(u_1 - u_0) H_N q_N.$$

Doob's L^2 inequality therefore yields

$$\mathbb{E}\left[\sup_{u\in[u_0,u_1]}|M_N(u)|^2\right] \le 4\eta(u_1-u_0)H_N q_N \to 0.$$

Thus the martingale term vanishes uniformly in probability.

Let $f(x)=\eta\kappa x(1-x)$. On $[0,1]$, f is Lipschitz with constant at most $\eta\kappa$. Comparing (9) with the integral equation for (7) gives

$$\sup_{u_0\le r\le u}|x_N(r)-x(r)| \le |x_N(u_0)-x_0| + \sup_{u_0\le r\le u}|M_N(r)| + \eta(u_1-u_0)\varepsilon_N + \eta\kappa\int_{u_0}^{u}\sup_{u_0\le r\le s}|x_N(r)-x(r)|\,ds.$$

Gronwall's inequality proves uniform convergence in probability on $[u_0,u_1]$. Solving (7) and setting $x(0)=1/2$ gives (8). □

Interpreting the single-task curve shapes. For finite N, the task score is still a jagged jump process. The theorem says that a smooth logistic curve appears only after two effects become negligible: mixing cut error $\varepsilon_N \to 0$ and jump noise from large score units $H_N q_N \to 0$. Thus a jagged finite-task curve is compatible with the logistic frontier mechanism. Moreover, the growth-rate $x(1-x)$ has a direct interpretation: unlocked score measure x supplies capability to help unlock new score, while locked score measure $1-x$ delimits the remaining opportunities for score improvement. In the next subsection, we explain why averaging many such finite-task curves is the natural place to expect a clean scaling law.

D.3 Many-task Aggregation Reveals the Smooth Log-Sigmoid Law

We now return to the quantity fit in the benchmark: the average over many independently evaluated tasks. As we have observed empirically, the per-task curves are heterogeneous and often visibly jagged, while the cross-task averages are much smoother and become better fit by the log-sigmoid as more tasks are included. In our language, each task is its own finite frontier expansion process on a task-specific task graph. The aggregate theorem therefore answers the central question:

How can many jagged task curves produce a clean log-sigmoid scaling law after task averaging?

The answer has two layers. First, because different task environments do not interact with each other, each task can have its own approximate logistic frontier. Second, the benchmark average removes finite-task roughness and becomes a single log-sigmoid when the task midpoints and task speeds are sufficiently concentrated in the many-task limit.

Moving from a single task to a full benchmark introduces three additional sources of variation.

- **Task-level logistic frontier dynamics.** Each task has its own task graph, score atoms, influence matrix, cut error, and finite-jump noise. These quantities determine how closely that task's frontier expansion follows its own logistic limit.
- **Time-axis alignment.** Each task may have its own model-dependent midpoint $t_{\mathrm{mid},b}$. A benchmark-level fit, however, has only one common midpoint t_{mid}. Therefore the task-specific midpoint shifts must concentrate across tasks.
- **Learning-speed homogeneity.** Each task may also have its own frontier speed β_b. A benchmark-level fit uses one common speed β, so the model-dependent task learning speeds must concentrate around a shared value.

The aggregate theorem says that the log-sigmoid law appears as a population-level limit: finite-task roughness is washed out by averaging, and the remaining task-level logistic frontiers combine into a single curve when their midpoints and speeds are sufficiently aligned.

Definition D.7 (Benchmark task graph and state)**.** For a benchmark with M tasks, the tasks are indexed by $b = 1, \dots, M$. Task b has its own task graph

$$G_b = (E_b, \mathcal{E}_b), \qquad \mathcal{E}_b = \{j \to i : K^b_{ij} > 0\},$$

where E_b is the task's visible score-unit node set for the model being evaluated. It has normalized score measurees μ^b_i for $i \in E_b$, influence matrix $K^b = (K^b_{ij})_{i,j\in E_b}$, and field-to-progress conversion constant $\eta_b > 0$.

The state vector of task b is

$$n^b(u) = (n^b_i(u))_{i\in E_b},$$

and the capability field on node i is

$$h^b_i(u) = \sum_{j\in E_b} K^b_{ij} n^b_j(u).$$

Definition D.8 (Task score and benchmark average)**.** The normalized score of task b at log-time u is

$$x_b(u) = \sum_{i\in E_b} \mu^b_i n^b_i(u).$$

And the benchmark-average normalized score is

$$x_B(u) = \frac{1}{M} \sum_{b=1}^{M} x_b(u).$$

Here $u = \log t - \log t_{\text{mid}}$ is the log-time coordinate. For raw time, we denote the normalized score and benchmark score by

$$x^\dagger_B(t) = x_B(\log(t/t_{\text{mid}})), \qquad S_B(t) = S_{\max} \cdot x^\dagger_B(t),$$

where t_{mid} is the fitted aggregate midpoint and $S_{\max}$ denotes the fitted aggregate ceiling. Individual tasks may have residual midpoint shifts relative to this common coordinate, introduced below as δ_b in (10).

Definition D.9 (Task-specific logistic frontier)**.** In the benchmark-level log-time coordinate u, task b is assigned a task-specific frontier speed $\beta_b \geq 0$ and a residual midpoint shift $\delta_b \in \mathbb{R}$. The corresponding task-specific logistic curve is

$$\ell_b(u) = \frac{1}{1 + e^{-\beta_b(u-\delta_b)}}. \tag{10}$$

Here β_b controls the speed of task-level frontier propagation, while δ_b measures the task midpoint relative to the benchmark-level midpoint t_{mid}. Thus $\delta_b = 0$ means that task b's midpoint is aligned with the aggregate midpoint.

The first condition applies the single-task cut-mixing argument inside each task graph. It is written for the influence matrix $\eta_b K^b$, so the product-frontier coefficient is directly β_b.

Definition D.10 (Blockwise weighted cut error and field bound)**.** For each task b, define the task-conditioned weighted cut error by

$$\varepsilon_b = \sup_{A,B\subseteq E_b} \left| \sum_{i\in A} \sum_{j\in B} \mu^b_i \big(\eta_b K^b_{ij} - \beta_b \mu^b_j\big) \right|. \tag{11}$$

Define the block granularity and score-growth field bound by

$$q_b = \sum_{i\in E_b} (\mu^b_i)^2, \qquad H_b = \sup_{i\in E_b,\, n\in\{0,1\}^{E_b}} \eta_b \sum_{j\in E_b} K^b_{ij} n_j.$$

Condition D.3 (Blockwise weighted cut mixing and vanishing score units)**.** Assume that task speeds are uniformly bounded,

$$0 \leq \beta_b \leq \beta_+ < \infty,$$

and that the average task cut error and granularity error vanish:

$$\Delta_M = \frac{1}{M}\sum_{b=1}^{M}\varepsilon_b \to 0, \qquad A_M = \frac{1}{M}\sum_{b=1}^{M}\sqrt{H_b q_b} \to 0 \tag{12}$$

in probability.

The next condition asks the initial value variation is negligible across different tasks in the benchmark.

Condition D.4 (Blockwise initial alignment)**.** We assume the average initial value of the dynamics is aligned,

$$\frac{1}{M}\sum_{b=1}^{M}|x_b(u_0) - \ell_b(u_0)| \xrightarrow{P} 0.$$

Condition D.3 has a direct consequence: the observed benchmark trajectory is close to an average of task-specific logistic frontiers. Notice the averaging in the condition. We do not require every task to look smooth on its own, but the *average* contribution of cut errors, score units, and jumps to vanish. This is the formal version of the empirical claim that a population of jagged frontier expansion processes can reveal a smooth law. For the compact interval $I \subseteq \mathbb{R}$, define the following blockwise trajectory error supremum:

$$R_M(I) := \frac{1}{M}\sum_{b=1}^{M}\sup_{u\in I}|x_b(u) - \ell_b(u)|. \tag{13}$$

Then we have the following lemma that proves its vanishing property.

Lemma D.3 (Average error supremum, many-task limit)**.** *Under Conditions D.3 and D.4, $R_M(I) \xrightarrow{P} 0$ as $M \to \infty$ for any compact interval $I \subseteq \mathbb{R}$.*

Proof. For task b, the same Doob-Meyer decomposition as in the single-task proof gives

$$x_b(u) = x_b(u_0) + M_b(u) + \int_{u_0}^{u}\{\beta_b x_b(s)(1 - x_b(s)) + \rho_b(s)\}\, ds, \tag{14}$$

where M_b is a martingale and, by (11), $|\rho_b(s)| \le \varepsilon_b$ for all states. The task-level curve (10) solves

$$\frac{\mathrm{d}\ell_b}{\mathrm{d}u} = \beta_b \ell_b(1 - \ell_b).$$

Since $z \mapsto \beta_b z(1 - z)$ is Lipschitz on $[0, 1]$ with constant at most β_+, Gronwall's inequality applied to (14) yields

$$\sup_{u\in I}|x_b(u) - \ell_b(u)| \le e^{\beta_+|I|}\left(|x_b(u_0) - \ell_b(u_0)| + \sup_{u\in I}|M_b(u)| + |I|\varepsilon_b\right), \tag{15}$$

where $|I| = u_1 - u_0$.

The first term converges to zero by Condition D.4. The third term as well by Condition D.3. It remains to average the martingale terms. A jump of unit i in task b changes x_b by μ_i^b, and the total score-growth field is bounded by H_b. Hence the predictable quadratic variation of M_b can be bounded by

$$\langle M_b\rangle_I \le |I| H_b q_b.$$

Doob's L^2 inequality gives the conditional bound

$$\mathbb{E}\left[\sup_{u\in I}|M_b(u)| \,\middle|\, \mathcal{G}_M\right] \le 2\sqrt{|I| H_b q_b}, \tag{16}$$

where $\mathcal{G}_M$ denotes the task-level weights and influence matrices. This conditional form is useful because A_M may itself be random. Let

$$Z_M = \frac{1}{M}\sum_{b=1}^{M} \sup_{u\in I} |M_b(u)|.$$

For any $\varepsilon > 0$ and $a > 0$, (16) implies

$$\begin{aligned}\mathbb{P}(Z_M > \varepsilon) &\le \mathbb{P}(A_M > a) + \mathbb{P}(Z_M > \varepsilon,\, A_M \le a)\\ &\le \mathbb{P}(A_M > a) + \varepsilon^{-1}\mathbb{E}[Z_M \mathbb{1}\{A_M \le a\}]\\ &\le \mathbb{P}(A_M > a) + \frac{2\sqrt{|I|}\,a}{\varepsilon}.\end{aligned}$$

Since $A_M \to 0$ in probability, first let $M \to \infty$ and then let $a \downarrow 0$ to obtain $Z_M \to 0$ in probability. Averaging (15) over b and using (12) proves (13). □

The next two task-distributional assumptions serve to align the scaling curves within the benchmark. They are not consequences of blockwise cut mixing. They are what turns an average of task-level sigmoids into a single benchmark sigmoid.

Assumption D.1 (Uniform log-time bias)**.** There is a single benchmark-level midpoint t_{mid} such that the residual task shifts in (10) satisfy

$$D_M = \frac{1}{M}\sum_{b=1}^{M} |\delta_b| \to 0.$$

The strict uniform-bias case is $\delta_b \equiv 0$.

Assumption D.2 (Concentrated environment learning speed)**.** There is a scalar $\beta > 0$ such that

$$B_M = \frac{1}{M}\sum_{b=1}^{M} |\beta_b - \beta| \to 0.$$

In particular, $M^{-1}\sum_{b=1}^{M} \beta_b \to \beta$.

The reason these assumptions are necessary can be seen from the aggregated frontier. Averaging removes jaggedness, but it does not by itself align task difficulty or environment learning speed. When blockwise cut mixing error tends to zero, the leading expected growth rate is

$$\frac{1}{M}\sum_{b=1}^{M} \beta_b x_b(u)(1 - x_b(u)).$$

This becomes $\beta x_B(1 - x_B)$ only if tasks are aligned enough that the task scores and speeds behave like one population. Even when all speeds are equal, persistent task dispersion subtracts from the scalar frontier through

$$\frac{1}{M}\sum_{b=1}^{M} x_b(1 - x_b) = x_B(1 - x_B) - \frac{1}{M}\sum_{b=1}^{M}(x_b - x_B)^2.$$

The theorem below proves the core convergence once the blockwise frontier condition, midpoint alignment, and speed concentration are in place.

Theorem D.2 (Benchmark aggregate produces the log-sigmoid law in the many-task limit)**.** *Fix a compact log-time interval $I \subseteq \mathbb{R}$. Suppose Conditions D.3-D.4, Assumptions D.1-D.2 hold for some $t_{\text{mid}} > 0$ and $\beta > 0$. Then*

$$x_B(u) \xrightarrow{P} \ell_\beta(u), \quad \forall u \in I.$$

Equivalently, for raw times t whose log coordinate $u = \log(t/t_{\text{mid}})$ lies in I, using the notation from Definition D.8, we have

$$x_B^\dagger(t) = \frac{S_B(t)}{S_{\max}} \xrightarrow{P} \frac{1}{1 + (t_{\text{mid}}/t)^\beta}. \tag{17}$$

Proof. We prove the statement for the normalized score $x_B(u)$. Multiplication by a fixed fitted ceiling $S_{\max}$ gives the raw-score statement.

Step 1: replace observed task trajectories by task-level frontier limits. By Lemma D.3,

$$\sup_{u\in I}\left|x_B(u)-\frac{1}{M}\sum_{b=1}^{M}\ell_b(u)\right|\le\frac{1}{M}\sum_{b=1}^{M}\sup_{u\in I}|x_b(u)-\ell_b(u)|=R_M(I)\to 0 \tag{18}$$

in probability.

Step 2: align the task-level frontiers to the benchmark frontier. Since the logistic link satisfies $|\mathsf{sigmoid}'(z)|\le 1/4$ for all z, writing $\ell_\beta(u)=\mathsf{sigmoid}(\beta u)$, we have

$$\begin{aligned}\sup_{u\in I}|\ell_b(u)-\ell_\beta(u)|&\le\frac{1}{4}\sup_{u\in I}|\beta_b(u-\delta_b)-\beta u|\\&\le\frac{1}{4}\left(\sup_{u\in I}|u|\,|\beta_b-\beta|+\beta_b|\delta_b|\right).\end{aligned} \tag{19}$$

The uniform speed bound in Condition D.3 gives $\beta_b|\delta_b|\le\beta_+|\delta_b|$. Averaging (19) over tasks and using Assumptions D.1 and D.2,

$$\frac{1}{M}\sum_{b=1}^{M}\sup_{u\in I}|\ell_b(u)-\ell_\beta(u)|\le\frac{1}{4}\left(\sup_{u\in I}|u|\,B_M+\beta_+D_M\right)\to 0. \tag{20}$$

Step 3: combine the two approximations. The triangle inequality gives

$$\begin{aligned}\sup_{u\in I}|x_B(u)-\ell_\beta(u)|\le\sup_{u\in I}&\left|x_B(u)-\frac{1}{M}\sum_{b=1}^{M}\ell_b(u)\right|\\&+\frac{1}{M}\sum_{b=1}^{M}\sup_{u\in I}|\ell_b(u)-\ell_\beta(u)|.\end{aligned}$$

The first term converges to zero by (18); the second converges to zero by (20). This proves uniform convergence in probability of x_B to ℓ_β on I. Substituting $u=\log(t/t_{\mathrm{mid}})$ gives (17). □

How does the benchmark aggregate produce the empirical scaling law. This is the theorem that corresponds to the empirical scaling law. The aggregate averages many frontier expansion processes on task-specific graphs to produce an emergent smooth curve in the many-task limit. For the convergence to hold, we need blockwise cut mixing that ensures task-level product frontier growth rates, the small-score-unit condition that makes jump noise vanish in the aggregate, and the midpoint and speed assumptions that prevent a mixture of incompatible sigmoids. Under these conditions, the benchmark average can finally converges to a single log-sigmoid law in (17), instead of being merely S-shaped.

D.4 Graph Self-similarity Induces Log Scale for Time Axis

The previous subsections explain why frontier expansion gives logistic dynamics once progress is measured in an effective learning coordinate. We now explain why this coordinate should often be logarithmic in raw interaction time.

Recall that the frontier process already takes place on the model-conditioned task graph $G_K=(E,\mathcal{E}_K)$. An edge $j\to i$ exists means that an unlocked source node j can help unlock a target node i, with field strength (marginal improvement in scores) K_{ij}. The weighted cut $C(L,U)$ measures the total influence crossing from the unlocked set U to the locked set L.

The remaining question is how raw interaction time moves the agent through the latent task graph. The answer we propose is graph-dependent and geometric: harder regions of the task graph require more search effort to expose. If this search geometry is approximately self-similar across difficulty scales, then equal additive increases in difficulty require multiplicative increases in raw time. This is why the natural frontier coordinate becomes $\log t$.

Definition D.11 (Difficulty scale on task-graph edges)**.** For each influence edge $j \to i \in \mathcal{E}_K$, let

$$\rho_{ij} \geq 0$$

be the difficulty scale of that edge. Smaller ρ_{ij} means that the influence from j to i becomes usable at lower search effort; larger ρ_{ij} means that the influence requires deeper search or longer interaction to become usable.

For $r \geq 0$, define the exposed influence matrix

$$K_{ij}^{(\leq r)} = \begin{cases} K_{ij}, & \text{if } j \to i \in \mathcal{E}_K \text{ and } \rho_{ij} \leq r, \\ 0, & \text{otherwise.} \end{cases}$$

The corresponding exposed field is

$$h_i^{(\leq r)}(n) = \sum_{j \in E} K_{ij}^{(\leq r)} n_j.$$

Increasing r reveals more of the same task graph. At small r, only low-difficulty edges contribute to the field. At larger r, higher-difficulty edges also contribute to the frontier cut. Thus r is an effective coordinate for how much of the task graph has become usable.

Definition D.12 (Search volume at difficulty scale)**.** Let $k \in \mathbb{N}^+$ index difficulty levels, with level k corresponding to the scale band $[k\Delta r, (k+1)\Delta r)$ where $\Delta r > 0$. Define the set of task-graph edges at level k by

$$\mathcal{E}_k = \{j \to i \in \mathcal{E}_K : k\Delta r \leq \rho_{ij} < (k+1)\Delta r\}.$$

Let $N_k = |\mathcal{E}_k|$ be the number of edges whose difficulty lies in this scale band. We interpret N_k as a discrete proxy of search-volume: it counts how much edge structure must be exposed at difficulty level k.

Search volume is different from score measure. score measure μ measures how much benchmark value is obtained once nodes are unlocked. Search volume measures how much task-graph structure must become usable before the corresponding frontier influence can appear.

Assumption D.3 (Self-similar task-graph edge growth)**.** There exist constants $b > 1$ and $c_-, c_+ > 0$ such that, throughout the scale range of interest,

$$c_- b^k \leq N_k \leq c_+ b^k.$$

This assumption says that each additive increase in difficulty level multiplies the amount of relevant task-graph edge structure by a factor comparable to b. Since level k corresponds to scale $r_k = k\Delta r$, this means

$$N_k \asymp b^{r_k/\Delta r} = \exp\left(\frac{\log b}{\Delta r} r_k\right).$$

We write

$$h = \frac{\log b}{\Delta r}$$

for the corresponding search-entropy parameter. Informally, h measures how quickly the amount of task-graph structure grows as difficulty scale increases.

The discrete assumption motivates the continuum approximation

$$V(r) = V_0 e^{hr}, \qquad V_0 > 0,$$

where $V(r)\,\mathrm{d}r$ is the search effort needed to expose the task-graph edge structure in the difficulty band $[r, r+\mathrm{d}r]$. Hence the cumulative search volume up to difficulty scale r is

$$\mathcal{V}(r) = \int_0^r V(s)\,\mathrm{d}s = \frac{V_0}{h}(e^{hr} - 1).$$

Assumption D.4 (Linear search-effort supply)**.** Let $A(t)$ be the cumulative search effort supplied by raw interaction time t. Assume

$$A(t) = \nu t$$

for some $\nu > 0$.

This assumption says that raw interaction time is proportional to available search effort. The search may be adaptive, but one unit of raw time does not by itself expose exponentially many edges of the task graph.

Proposition D.1 (Self-similar task geometry gives logarithmic time)**.** *Under Assumptions D.3 and D.4, the difficulty scale exposed by raw time t satisfies*

$$r(t) = \frac{1}{h}\log\left(1 + \frac{h\nu}{V_0}t\right).$$

In particular, in the scale-free regime $t \gg V_0/(h\nu)$,

$$r(t) = \frac{1}{h}\log t + O(1).$$

The proposition is just the inversion of cumulative search volume. The scale $r(t)$ is defined by $\mathcal{V}(r(t)) = A(t)$. Since $\mathcal{V}(r)$ grows exponentially in r while $A(t)$ grows linearly in t, the exposed difficulty scale grows logarithmically in raw time.

Connection to the frontier law. The earlier frontier law describes score growth once progress is measured in the coordinate that makes task-graph influence usable. Here that coordinate is r. As r increases, more entries of K are exposed through $K^{(\le r)}$, the field $h_i^{(\le r)}$ grows, and the frontier cut from unlocked nodes to locked nodes expands through the same mechanism as before.

If the exposed task graph is approximately weighted cut-mixed at each scale, with a scale-stationary effective frontier coefficient, then the coarse frontier dynamics in the scale coordinate take the product form

$$\frac{\mathrm{d}x}{\mathrm{d}r} = \gamma x(1-x),$$

where $\gamma > 0$ is the frontier speed per unit difficulty scale. This is the earlier weighted-cut frontier mechanism written in the coordinate r.

Since $r(t) = h^{-1}\log t + O(1)$ in the scale-free regime, a unit increase in $\log t$ corresponds to approximately $1/h$ units of difficulty-scale progress. Therefore

$$\frac{\mathrm{d}x}{\mathrm{d}\log t} = \frac{\gamma}{h}x(1-x).$$

Thus the fitted log-time frontier speed is

$$\beta = \frac{\gamma}{h}.$$

Solving the log-time logistic equation gives

$$x(t) = \frac{1}{1 + (t_{\mathrm{mid}}/t)^{\beta}}.$$

Why self-similar task geometry gives log time. The weighted-cut argument explains the frontier factor $x(1-x)$, as previously shown. Self-similar task-graph geometry explains why the time coordinate is measured by log scale. If each additive increase in difficulty scale multiplies the number of relevant task-graph edges by a factor b, then search volume grows like e^{hr}, where $h = (\log b)/\Delta r$. Since raw interaction time supplies only $O(t)$ search effort, the exposed difficulty scale satisfies $r(t) = h^{-1} \log t + O(1)$. Therefore a logistic frontier law in task-graph scale becomes a logistic law in $\log t$.

D.5 Discussion and Limitations

The derivation above gives a sufficient mechanism for the empirical log-sigmoid law. Its assumptions are useful precisely because they identify concrete ways in which the log-sigmoid limit can fail. We therefore treat it as a mechanistic account of the observed regime rather than a claim that all environment-learning curves must be logistic.

Finite score granularity. The single-task theory allows finite tasks to remain visibly jagged. Real tasks may contain a few large hidden tests, decisive proof obligations, or high-weight rubric cells. When such score units remain macroscopic, the martingale error term need not vanish for that task, and the realized best-so-far curve can exhibit long plateaus followed by sudden jumps. The aggregate theorem only requires that such coarse units do not dominate the benchmark average. If a non-negligible fraction of benchmark score measure is carried by coarse tasks, the aggregate curve may retain visible jumps or large run-to-run variance even when the average drift is approximately logistic.

Weighted cut mixing. Weighted cut mixing is the core condition of the scaling law. It says that every macroscopic unlocked–locked frontier cut sees approximately product-measure influence. If the task graph has persistent bottlenecks, modules, prerequisite chains, or separated high-transfer and low-transfer regions, then the frontier remembers where it is in the graph, not only how much score measure has been unlocked. In that case, the natural limit is no longer a one-dimensional logistic equation. One should instead expect multi-type dynamics, delayed takeoff, multiple inflection regions, long plateaus, or a sum of sigmoids corresponding to different task modules.

Attainable support and the fitted ceiling. The analysis treats $S_{\max}$ as the stable score measure of an effective reachable support. This is appropriate when the set of practically attainable score units is fixed over the fitted time window. However, if longer interaction changes what is effectively reachable—for example, if weak transfer routes become usable only at much longer horizons—then the denominator of the normalized score is itself moving. A short-window fit may still provide a useful effective ceiling, but that ceiling should not be interpreted as an indefinite upper bound.

Task midpoint alignment. The aggregate theorem assumes that a single benchmark-level midpoint removes most task-level log-time bias. If residual midpoint shifts δ_b remain widely dispersed, the benchmark average becomes a convolution of shifted task frontiers. Such a curve may still be smooth and S-shaped, but it need not satisfy the scalar logistic ODE. In this regime, the fitted midpoint and slope are window-dependent summaries of the task-midpoint distribution rather than intrinsic benchmark constants.

Learning-speed concentration. The aggregate theorem also assumes that task frontier speeds concentrate around a common value. If some task families have persistently steep frontiers while others have persistently shallow frontiers, then the average of task-level sigmoids is generally not itself a sigmoid. Early progress may be dominated by fast tasks, while later progress may be governed by slower tasks. A single fitted β may therefore reflect task-family composition rather than a true scalar environment-learning speed.

Choice of time coordinate. The log-time coordinate is justified by the graph self-similarity hypothesis in which unit increments of difficulty require multiplicative increases in search effort. Some environments, however, have characteristic raw-time cycles: fixed evaluation delays, daily data refreshes, hard deadlines,

staged curricula, or batch feedback. In such cases, another time coordinate, or a piecewise time model, may be more appropriate than a single log-time transformation.

Together, these limitations clarify the scope of the proof. The appendix gives one route from frontier expansion to the observed population-level log-sigmoid law. A fuller theory would classify the non-logistic limits produced by coarse score units, non-mixing graphs, moving attainable supports, dispersed midpoints, heterogeneous learning speeds, and non-scale-free feedback schedules.

E More Discussion on the Scaling Law Shapes

A mechanistic reading selects the log-sigmoid. Because the candidate S-curves fit almost equally well (Table 1; the log-sigmoid/log-probit degeneracy has been known since Berkson [6]), the choice cannot rest on fit; we make it on mechanism. Written as a growth law for the normalized score $y = S/S_{\max}$, each candidate's rate makes a different statement about what drives progress and what limits it.

- **Log-sigmoid:** $\frac{dy}{d\tau} = \beta\, y(1-y)$ in log time $\tau = \ln t$. The detailed derivation is given in Section 3.3 and Appendix D; here the key mechanistic reading is that y is the unlocked attainable score mass and $1-y$ is the remaining locked score mass. If the aggregate influence crossing the unlocked–locked frontier is approximately proportional to the product of these masses, then progress follows $y(1-y)$. The log-time coordinate comes from the self-similar search geometry of the task graph. This reading is consistent with two empirical checks: Section 5.2 shows that continuous stateful runs outperform equal-budget repeated sampling, so progress is not explained by independent attempts alone; and Section 5.4 shows a finite task advancing through sparse but cumulative breakthroughs, where an early working pipeline makes later repairs searchable. Inflection occurs at $y = 0.5$ (symmetric).
- **Log-Gompertz:** $\frac{dy}{d\tau} = c\, y \ln(1/y)$. The $\ln(1/y)$ term is the signature of an engine that winds down multiplicatively as the system matures (e.g., tumour growth, where the proliferating fraction shrinks). It is front-loaded, with inflection at $y = 1/e \approx 0.37$: the relative growth rate diverges as $y \to 0$, so a tiny system grows explosively because the limitation it will eventually hit does not yet exist. Experience acquisition is the opposite—its early phase is *slow* because a foothold must first be bootstrapped, not fast for lack of a brake.
- **Log-Probit:** $\frac{dy}{d\tau} = \frac{1}{\sigma}\,\varphi\big(\Phi^{-1}(y)\big)$, a sweep across a *log-normal* distribution of difficulties—difficulties formed as products of many *independent* factors (a multiplicative central-limit argument). Experience acquisition is path-dependent rather than a product of independent factors, so this microfoundation does not apply, even though probit and the logistic are empirically near-indistinguishable.
- **Weibull CDF:** $y(t) = 1-\exp\{-(t/\lambda)^{\beta}\}$, equivalently $\frac{dy}{dt} = h(t)(1-y)$ with $h(t) = \frac{\beta}{\lambda}(t/\lambda)^{\beta-1}$. Its hazard is a function of raw elapsed time, while accumulated progress enters only through the survival term $(1-y)$. This makes it the natural first-passage or repeated sampling baseline. For a single 0–1 reward task with independent attempts of success probability p, $P_{\text{pass}}(k) = 1-(1-p)^k = 1-\exp[-k(-\ln(1-p))]$, which is an exponential CDF, the $\beta = 1$ Weibull case (approximately $1-\exp(-pk)$ for small p). Such a curve improves because repeated independent attempts shrink the remaining failure mass, not because state carried across attempts makes later attempts more effective. Section 5.2 shows that stateful learning beats this repeated-sampling mechanism under the same total budget, and the Weibull CDF also fits worse than the log-sigmoid in Table 1. Individual improvements may have a first-passage flavor, but the macroscopic best-so-far trajectory accumulates through dependent, stateful steps rather than through a raw-time hazard with no accumulated-progress factor.
- **Log-linear:** $\frac{dy}{d\tau} = b$ (constant). With no saturating term it grows without bound and cannot level off, contradicting the many tasks that reach a ceiling within the budget; it attains by far the worst fit.

We therefore read the log-sigmoid not merely as the best-fitting S-curve but as the curve whose $y(1-y)$ rate law matches the frontier-expansion interpretation: unlocked score mass supplies reusable capability, while locked score mass bounds the remaining opportunity for improvement. This is a mechanistic preference, not an empirical exclusion—the data cannot separate the symmetric families. It is falsifiable through the

inflection: a symmetric peak near $y = 0.5$ is consistent with the logistic (and probit), a front-loaded peak near 0.37 would favor Gompertz, and a back-loaded peak near 0.63 would favor Weibull.

F Additional Related Work

F.1 Benchmarks Not Suitable for Measuring Self-Evolution

Many benchmarks evaluate whether models or agents can answer questions, complete tasks, or produce correct artifacts, without making self-evolution itself the object of measurement. We use this label in a narrow evaluation sense: these benchmarks may still involve reasoning, tools, iteration, or environment interaction, but their primary reported quantity is not related to learning. It is usually final-answer accuracy, task success, pass rate, artifact quality, reproduction fidelity, or human-time horizon. In such settings, interaction is typically a means to produce a final answer or artifact, rather than the quantity being measured.

At the shortest and least interactive end, many classic capability benchmarks, such as MMLU [29], GPQA [62], AIME [45], and closed-form coding or math evaluations [11, 30], ask models to solve static problems and report final accuracy. These benchmarks can be difficult and useful, but the model's actions do not create a changing environment, and feedback from the benchmark is not exposed as experience for later adaptation. They therefore measure static knowledge and reasoning accuracy rather than self-evolution.

Agentic software benchmarks increase realism by placing agents in codebases, development workflows, or larger construction tasks, but many still reduce evaluation to the final result. SWE-bench [35] evaluates issue repair in existing repositories; development and evolution benchmarks such as RoadmapBench [80] and SWE-EVO [70] evaluate incremental changes over existing projects or release histories; construction or reconstruction benchmarks such as NL2Repo-Bench [18] and ProgramBench [82] evaluate complete repository generation or recovery of program behavior; and SWE-AGI [86] evaluates specification-driven system construction under a fixed MoonBit API scaffold. FrontierCode [41] raises the bar on readiness for production by evaluating whether coding agents produce maintainable, mergeable changes under repository-specific quality rubrics. Their task formats differ, but the shared evaluation target is final patch correctness, repository quality, program behavior, maintainability, or test pass rate rather than the trajectory by which an agent improves from feedback.

Some evaluations broaden realism without becoming agentic learning benchmarks. GDPval [60], for example, measures model performance on economically valuable professional deliverables across occupations. Agents' Last Exam [69] is a close comparator in realism and autonomy: it evaluates generalist computer-use agents on economically valuable professional workflows in real Windows or Linux sandboxes, with CLI and GUI access, professional software, hidden references, and verifiable final artifacts. The distinction is the evaluation target rather than agenticity. Agents' Last Exam is oriented toward completion: agents work toward a final deliverable, and the benchmark reports final scores, pass rates, cost, or time after the artifact is graded. Its trajectories are valuable for replay, audit, and failure analysis, but learning from the benchmark's feedback over a run is not the main quantity being measured.

Other benchmarks use realistic or long-horizon workflows that could support learning, but their published protocols mainly measure final outcomes. HCAST [63] and METR time-horizon evaluations [38] calibrate model or agent success by human task duration. Terminal-Bench [46], RE-Bench [76], PaperBench [68], CORE-Bench [66], PRBench [61], ReplicationBench [83], Collider-Bench [23], and ScienceAgentBench [12] involve executable software tasks, terminal workflows, or scientific reproduction. These settings are more compatible with learning than short closed-form tasks, but their central questions are typically whether a task is completed, an artifact works, a paper is reproduced, or a final score is high enough under a budget. PaperBench is a useful example of the distinction: it evaluates agentic replication of 20 ICML papers from scratch using detailed rubrics and many gradable subtasks, but the reported score is still replication quality rather than improvement per unit of feedback.

F.2 Benchmarks Suitable for Measuring Learning or Self-Evolution

Another line of work is more suitable for measuring learning or self-evolution because it exposes new context, serialized streams, iterative feedback, or long executable workspaces. These benchmarks are the closest conceptual comparisons to EdgeBench, but they differ in what creates the experience stream and what quantity is ultimately scored. We group them by evaluation interface: learning from supplied context, learning over serialized streams or task sequences, and iterative optimization. This grouping is descriptive rather than an ordering by interaction strength. Sequential benchmarks can be highly interactive, and optimization benchmarks can expose diagnostic feedback; the key question is whether the agent's own behavior shapes the future experience it receives within a long executable task.

Context-learning benchmarks study the least agentic form of adaptation. CL-bench [21] and CL-bench Life [20] test whether models can use newly provided professional or personal context to answer downstream questions or perform tasks grounded in that context. This counts as learning from external information, but the stream of episodes is usually not shaped by the agent's own actions, and there is no changing environment. The model is asked to use context, not to discover and exploit feedback through sustained interaction.

Sequential learning benchmarks introduce a stream-like structure. EvaLearn [19] groups 648 problems into 182 sequences and evaluates learning capability and efficiency as models solve related problems in order. Continual Learning Bench [4] explicitly reports improvement over sequential experience and introduces a stateful-versus-stateless comparison similar to Section 5.2. However, its synthetic task sequences have explicit subtask boundaries, whereas EdgeBench uses single continuous problems. We therefore partition by time rather than by subtask, which changes how the stateful and stateless settings are defined. StreamBench [78] also reports improvement over feedback streams. LLF-Bench [13], LifelongAgentBench [87], and Evo-Memory [73] study related questions through language feedback, interdependent tasks, experience replay, or evolving memory. These works establish that static capability and learning ability are distinct axes. The difference from EdgeBench is not that their interaction is necessarily weaker; rather, the sequence is usually organized by the benchmark as a series of instances, tasks, feedback events, or memory updates. In EdgeBench, the sequence is endogenous to one long task: the agent plans, edits artifacts, submits attempts, receives diagnostics, and thereby affects what it learns next.

Iterative optimization benchmarks are closer to EdgeBench because they make repeated attempts and empirical feedback central to performance. Frontier-Eng [14] studies improvement over generative optimization cycles, while MLS-Bench [42] and Frontier-CS [43] evaluate test-time discovery or open-ended CS optimization with visible metrics, submissions, or trajectory-level reporting. MLE-bench [10] is also naturally viewed in this family: agents work in a free Kaggle-style workspace, the paper includes studies of how performance scales with resources, and its 100-hour analysis grades snapshots of the agent's best attempt over elapsed time. ALE-Bench [34] is another adjacent long-horizon setting driven by fixed objectives, even though its original protocol primarily reports outcomes on algorithm engineering rather than learning metrics.

FrontierSWE [15] and AutoLab [81] are the closest prior work. Both evaluate long-horizon agentic improvement over executable artifacts through repeated edits, experiments, and empirical feedback. FrontierSWE focuses on software engineering and performance-tuning tasks, with reported average agent runtime of roughly 3–4 hours across models and task categories. AutoLab gives agents working but deliberately suboptimal programs across systems, CUDA, model development, and puzzle-style optimization tasks; it directly measures iterative improvement and should not be treated as a non-learning benchmark, although performance curves are not reported as the main result and most current tasks run for 2–4 hours, with only a small minority exceeding 6 hours. Under the shared premise of long-horizon agentic tasks, the main distinction is domain coverage: FrontierSWE is concentrated on software engineering, AutoLab emphasizes research and engineering tasks centered on optimization, and EdgeBench spans a broader set of executable domains. EdgeBench also uses a day-scale task contract and makes the time-aligned trajectory itself the primary object of measurement, with metrics for improvement area, regression, and active learning span.

Taken together, these benchmarks cover important pieces of the problem: realistic executable work, learning over organized streams, and iterative optimization under feedback. EdgeBench targets their intersection. It evaluates within-run self-evolution in long-horizon executable environments where the experience stream

arises from the task itself, the agent can influence what it observes next, and evaluation uses a general-purpose agent harness rather than a benchmark-specific learning scaffold. Its trajectory-level metrics report improvement, regression, active learning span, and final performance over the same continuous run.

F.3 Scaling Laws for LLMs and Agents

Classical neural and language-model scaling laws mainly study pretraining, modeling reducible loss as a function of model size, data, and training compute. Kaplan et al. [36] showed that language-modeling loss follows power-law relationships over several axes of scale, and Chinchilla-style work refined the compute-optimal allocation between model parameters and training tokens [32]. More recent work studies how benchmark performance, rather than loss, changes with scale; because accuracy-like metrics are bounded, these curves are often better described by sigmoidal or other saturating forms [7, 59, 64, 85]. Our log-sigmoid environment learning curves in Section 3 are closest in mathematical form to this bounded-performance line of work, but the independent variable is elapsed environment interaction within a task rather than pretraining compute or model scale.

Test-time and inference-time scaling provide a second point of comparison: different test-time methods exhibit their own empirical scaling behavior. One line scales plain repeated sampling: Evaluating Large Language Models Trained on Code introduced pass@k as a repeated-sampling evaluation for code generation [11], AlphaCode used large-scale sampling, filtering, and selection to improve competitive-programming solve rates [40], and Large Language Monkeys showed that repeated sampling can scale coverage across orders of magnitude when outputs are automatically verifiable [9]. A second line studies how test-time compute should be allocated across search, revision, voting, and verifier- or reward-guided strategies [67, 79]. A third line scales long-chain-of-thought inference in reasoning models: OpenAI's o1 report made train-time reinforcement learning compute and test-time thinking compute explicit scaling axes [50], and DeepSeek-R1 studies how reinforcement learning can elicit reasoning behaviors in LLMs [16]. These studies mostly scale the compute spent producing or selecting answers on traditional math, code, proof, or game-style tasks, and the reported functional forms are often local log-linear trends, exponentiated power laws, or frontiers for allocating compute. EdgeBench instead scales *interaction time*: it measures how an agent's best-so-far performance changes as elapsed time in an executable environment increases and as the agent repeatedly reads, acts, receives feedback, and revises. This gives broader task coverage and a different scaling object than sampling from a static prompt or selecting among answers.

Recent agentic test-time scaling work moves closer to our setting by allowing agents to acquire information from an environment over time. OpenAI's CUA report observed that computer-use performance improves when more steps are allowed [52], and BrowseComp reports smoother gains as browsing effort increases [72]. Thinking vs. Doing explicitly frames interaction length as a test-time scaling axis for web agents [65]. Closest in spirit, Mang et al. [44] compare agents and humans on a long-horizon coding contest setting where agents can try, observe, and revise over time. They report an informative human reference curve and find that current agents plateau while strong humans continue improving. However, the measurement is concentrated in one contest-style domain and a limited task set. The human improvement curve is also difficult to extrapolate: the observed segment appears closer to linear improvement over time, but it may cover only the early portion of a longer sigmoid-like trajectory.

Scaling has also been studied in reinforcement learning. Hilton et al. [31] introduced intrinsic performance to obtain smooth power-law relations between environment interactions, model size, and RL performance. Recent LLM RL scaling work fits sigmoidal compute-performance curves for post-training and uses smaller runs to predict larger RL runs [37]. Both RL and our evaluation measure learning from environment feedback rather than from human-labeled data. The practical difference is resolution and coverage. RL learning runs are expensive, so scaling evidence is usually gathered on narrower task distributions, fewer distinct environments, and fewer long trajectories. In EdgeBench, the learning algorithm is in-context learning (ICL): the agent absorbs observations, diagnostics, and prior attempts into its context and uses them to improve subsequent actions. Because ICL is cheaper to repeat than RL training, EdgeBench can measure environment learning across 134 executable task environments, multiple domains, full time-aligned trajectories, and repeated trials, yielding substantially broader environment coverage than is typical in RL scaling studies.

The resulting fits go beyond a bounded-performance analogy to RL scaling: they provide a higher-resolution measurement of how agents learn from interaction.

G Additional Benchmark and Experiment Details

G.1 Estimating the With- and Without-Experience Curves

This describes how the two curves in Section 5.2 are estimated. The with-experience curve averages three 12-hour best-so-far curves per task, then averages across tasks. For the without-experience curve, let $u_1, \ldots, u_n$ be the scores of the n independent attempts on a task; at elapsed time $t = k\tau$ we estimate the expected best of k attempts as

$$\hat{u}_{k\tau} = \mathbb{E}_S\big[\max_{i \in S} u_i\big], \tag{21}$$

where S is sampled uniformly from the $\binom{n}{k}$ size-k subsets of the n attempts. This is the score-valued, without-replacement extension of the pass@k estimator [11]; when each u_i is binary, it reduces to the usual pass@k estimate.

G.2 Gravitational-Wave Case Study Details

The selected milestones compress the run into interpretable phase transitions. Table 4 summarizes the main phases behind the selected milestones rather than every submission. The first selected milestone already passes the protocol and entrypoint checks, generates all five required CSV files on the canonical grids, and obtains a score of 42.8. A subsequent key update raises the score to 47.1, mainly by improving the source dynamics. Around hour 1, another milestone reaches roughly 50 as the agent improves the H1/L1 spectrograms and early H1 alignment in the time domain. These gains reflect standard signal processing: filtering, normalization, windowing in the time-frequency plane, interpolation to the canonical grid, and cropping around the event.

The largest score gain comes from the high-weight source-dynamics component. The largest jump occurs around hours 4–5: the overall score rises from about 52.3 to 59.7, driven almost entirely by the source dynamics subscore (64.2 to 89.0). Because this subtask carries the largest weight, a compact physical model plus calibration produces a large total-score gain. After that point, the agent shifts to H1 waveform tuning, where the H1 time-series subscore climbs from roughly 47 to 95 by the end.

The final solution is strong on H1 waveform and source dynamics, but it still leaves room to improve toward a human-standard LIGO-style analysis. Table 5 breaks down the final score. GPT-5.5 scores high on H1 waveform reconstruction and the source dynamics but remains weak on the spectrograms and L1 reconstruction. Empirical calibration and parameter search produce a substantial score, but they do not replace a coherent end-to-end pipeline for strain preprocessing, whitening, time-frequency analysis, and waveform alignment across detectors.

Stage	Time	Best score	Main improvement
Initial milestones	0h	42.8–47.1	Built a reconstruction pipeline that can be evaluated; replacing a noisy frequency estimate improved the inferred source motion from 51 to 64 (**+13 pp.**)
Early signal processing	1–4h	52.3	Recentered the time-frequency window on the merger; the Hanford detector waveform and both detector spectrograms improved together, raising the best score from 47.1 to 52.3 (**+5.2 pp.**)
Main breakthrough	4–5h	59.7	Calibrated the binary-source model; the orbital velocity and separation score jumped from 64 to 89 (**+25 pp.**)
Late waveform tuning	5–11.5h	66.9	Found the remaining mismatch in the Hanford detector waveform; time-shift alignment and error fitting raised Hanford from 47 to 95 (**+48 pp.**)
Final refinement	12h	67.0	Consolidated final corrections for the Hanford and Livingston detectors; a small Livingston waveform gain raised the aggregate score from 66.9 to 67.0 (**+0.1 pp.**)

Table 4 Main phases in the gravitational-wave evolving trajectory. The trajectory shows how the agent refines signal processing, source dynamics, and per-detector waveform reconstruction over the 12-hour run.

Component	Final subscore
H1 time series	95.0
L1 time series	57.1
H1 spectrogram	42.7
L1 spectrogram	44.7
Velocity/separation	89.0
Aggregate score	67.0

Table 5 Final subscore composition for the GPT-5.5 gravitational-wave 12h run.

Task	GPT-5.5			GPT-5.4		
	Base	w/ Goal	w/ Ralph	Base	w/ Goal	w/ Ralph
`portfolio_risk_calibration`	25.0	27.5	**34.3**	10.7	**19.8**	12.2
`storyboard_ad_copywriting`	77.0	88.3	**97.3**	65.7	78.0	**83.3**
`arc_compiler_runtime`	**72.4**	70.6	52.6	50.0	47.2	**51.2**
`battery_soh_rul_anomaly`	30.2	23.8	**48.8**	**14.7**	14.6	13.4
`borden_source_inversion`	38.5	32.8	**62.2**	8.0	17.3	**23.2**
`capecod_plume_reconstruction`	16.4	16.0	**17.3**	12.6	13.3	**13.7**
`combinatorial_games_formalization`	38.2	**45.3**	39.8	17.8	13.0	**20.2**
`ffmpeg_swscale_reimplementation`	15.3	16.4	**25.2**	13.9	**28.3**	21.8
`flt_regular_formalization`	**75.1**	66.7	58.6	**48.3**	43.7	43.7
`git_rewrite_in_zig`	18.4	20.5	**22.0**	15.4	**18.4**	15.8
`tryst_text_adventure`	55.7	**56.2**	40.2	**44.3**	32.9	42.4
`openttd_transport_ai`	28.1	**39.8**	30.6	**11.9**	1.2	8.1
`pocketbase_backend_architecture`	**62.5**	**62.5**	41.7	20.8	**54.2**	0.0
`symbolic_integration_engine`	**44.0**	36.6	37.7	30.9	**63.7**	37.2
Avg.	42.6	43.1	**43.4**	26.1	**31.8**	27.6

Table 6 Harness-level continuation ablation of /goal mode and the Ralph loop. GPT-5.5 and GPT-5.4 are evaluated with Base, Goal, and Ralph settings under the same 12-hour budget; each value is the mean score over valid runs, and the Avg. row averages the displayed task rows. Incomplete cells are detailed in Appendix G.3. Bold marks the best setting and underlining marks the second-best setting.

G.3 Harness-Level Continuation Ablations

Long-running agent evaluation depends not only on model capability, but also on how the harness keeps the run alive and carries useful state across many hours. In a 12-hour task, an agent may stop its running unexpectedly due to idleness and need to resume. These continuation mechanisms are therefore part of the practical measurement setup: a weak scaffold can make a capable model stop working on the task, while a stronger scaffold may help the same model keep active and improving. As a supplementary harness-level ablation, we evaluate two such mechanisms, /goal mode and the Ralph loop, under a fixed 12-hour budget with GPT-5.5 and GPT-5.4.

Base uses the standard harness: one continuing agent session, a stop hook to prevent voluntary early exit, and auto-resume for abnormal exits. In /goal mode [56], the harness prompts the agent to create a task-level goal at the beginning, keep it active during the run, and mark it complete only when the task is validated. The Ralph loop follows the file-backed fresh-context pattern [33]: each loop starts a new agent invocation on the same workspace, asks it to read and update progress.md, then appends judge feedback to that file before the next loop. It uses up to 100 loops, a 7200-second per-loop cap, and the same 12-hour overall budget. Each cell schedules three runs and reports the mean score over valid runs.

As shown in Table 6, the goal mode and the Ralph loop often outperform the base setting, suggesting that long-horizon agents benefit from harness support that preserves and updates task state across extended runs. In the displayed-task average, GPT-5.5 improves from 42.6 in Base to 43.1 with Goal and 43.4 with Ralph, while GPT-5.4 improves from 26.1 in Base to 31.8 with Goal and 27.6 with Ralph. The gains are not uniform across all tasks and models, so we treat this as an appendix-level harness diagnostic rather than a main result.

Most cells use all three scheduled runs. A few GPT-5.4 cells have fewer valid runs because of intermittent API or network instability: one-run cells are w/ Goal for `storyboard_ad_copywriting` and `flt_regular_formalization`, plus w/ Ralph for `symbolic_integration_engine`; two-run cells are `battery_soh_rul_anomaly` (w/ Goal and w/ Ralph), `capecod_plume_reconstruction` (w/ Goal), `combinatorial_games_formalization` (w/ Ralph), `ffmpeg_swscale_reimplementation` (w/ Goal), `flt_regular_formalization` (w/ Ralph), `git_rewrite_in_zig` (w/ Goal), and `symbolic_integration_engine` (w/ Goal). All reported GPT-5.5 cells use the full three runs.

G.4 Per-Task Design Notes

This appendix gives per-task design notes for the 134-task EdgeBench suite.

Task	Design notes
Systems & Software Engineering	
`ann_vector_search_qps`	Replace a brute-force NumPy nearest-neighbor baseline with a high-performance approximate nearest-neighbor implementation under a hard recall constraint. Scored by queries per second.
`arc_compiler_runtime`	Implement a complete TypeScript compiler pipeline (lexer, parser, type checker, code generator) for a novel programming language defined by specification documents.
`rust_multicrate_reconstruction`	Reconstruct missing Rust implementations across a multi-crate content-addressable storage workspace, given only type signatures and public API contracts.
`codeflash_repair_performance`	Diagnose and repair intentionally degraded modules in a Python code-optimization tool spanning CST manipulation, profiling infrastructure, and test harness integration. Scored on both functional correctness and runtime performance.
`copier_modular_refactor`	Implement six architectural targets in the Copier Python scaffolding library: structured exceptions, template management, version compatibility, rendering modes, a worker class, and a user-data layer.
`dependent_type_checker`	Build a complete dependent type checker in Rust for a subset of Martin-Löf Type Theory, supporting cumulative universes, Pi/Sigma types, general inductive types with positivity checking, and universe polymorphism. Scored on both correctness and normalization throughput.
`entt_graph_module`	Implement seven feature targets in the EnTT C++ entity-component-system framework, including a graph module with adjacency matrix, a task-graph builder with transitive reduction, DOT export, and several core utility additions.
`exchange_core_throughput`	Maximize peak throughput of a Java financial matching engine built on the LMAX Disruptor by tuning thread topology, wait strategies, ring-buffer sizing, order-book implementation, and JVM configuration.
`ffmpeg_swscale_reimplementation`	Reimplement FFmpeg's libswscale pixel-format conversion and scaling library in Rust, handling multiple pixel formats and scaling algorithms. A correctness-passing scaffold is provided; the agent must optimize for speed via SIMD.
`git_rewrite_in_zig`	Reimplement git as a drop-in Zig binary producing identical CLI output, exit codes, and repository state as the C reference implementation. The C source is available for reading but cannot be compiled.
`high_performance_object_mapper`	Implement a .NET object-to-object mapping library using expression-tree compilation and IL emission, handling flat/nested mapping, collections, custom type converters, nullables, and inheritance hierarchies.
`integer_compression_codec`	Improve a C++ integer compression codec for better compression ratio and decode throughput on uint32 datasets via techniques such as delta encoding, bit-packing, and SIMD vectorization. Exact round-trip correctness is mandatory.
`juliet_vulnerability_analyzer`	Implement a deterministic static analyzer that processes structured program facts to detect vulnerabilities across six CWE categories (stack/heap overflow, integer overflow, null dereference, use-after-free, command injection).
`libexpat_x86_assembly`	Reimplement the libexpat XML parser entirely in x86-64 assembly, producing an ABI-compatible shared library. No C compiler is available—only assembler, linker, and libc.
`odata_query_service`	Complete a .NET OData query processing library: query-string parsing, LINQ expression-tree generation for filtering, multi-field sorting, pagination, field projection, and ASP.NET Core middleware integration.
`litestar_infra_refactor`	Implement five async infrastructure subsystems in the Litestar Python web framework: key-value stores with expiry, an event bus, WebSocket listener abstractions, a DTO framework, and a channels pub/sub system.
`lua_native_compiler`	Build a Lua 5.4 ahead-of-time native compiler that reads Lua source and emits standalone ELF executables with real x86-64 machine code—not C API call sequences or bytecode dispatch loops. Output must match the reference interpreter byte-for-byte.
`mimesis_modular_refactor`	Implement seven refactoring targets in the Mimesis Python fake-data library: a declarative schema class, an enum system with metaclass-driven random selection, and five data providers spanning payments, cryptography, science, internet, and unit systems.
`nlohmann_json_modularization`	Implement five feature targets in the nlohmann/json C++ library: UBJSON binary serialization, JSON Merge Patch (RFC 7386), Grisu2 shortest-representation float formatting, a range-based iteration interface, and library metadata macros.
`notebook_lossless_compression`	Build a lossless compression pipeline for Jupyter notebooks with a training phase (dictionary learning from a visible corpus) and per-file compression/decompression. Scored by overall compression ratio; byte-exact reconstruction is mandatory.
`packer_plugin_datasources`	Implement six targets in HashiCorp Packer's Go plugin ecosystem: address parsing with DNS validation, filesystem-based plugin discovery with checksum verification, HCL2 data-block integration, sensitive value handling, and new template functions.
`pocketbase_backend_architecture`	Implement four architectural targets in PocketBase (Go): a chain-based hook/event system, an HTTP router with middleware stacking, a dynamic OAuth2 provider registry, and a regex-based random string generator.
`pocketbase_tools_extensions`	Implement four Go utility packages for PocketBase: a REST JSON serializer with nested field picking, a zip archive utility, a cron scheduler with timezone support, and a SQL index parser/builder.
`postgres_wire_on_sqlite`	Implement a server that speaks the PostgreSQL wire protocol (v3) using SQLite as the storage backend, handling authentication, simple and extended query protocols, type mapping, transactions, and system catalog queries. Scored by pass rate on PostgreSQL's own test suites.
`quic_transport_stack`	Implement a subset of the QUIC transport protocol (RFC 9000): connection establishment, TLS 1.3 key derivation, AEAD encryption, packet number encoding, header protection, and core frame types.

Continued on next page

Task	Design notes
regex_automata_repair	Repair broken implementations in Rust's regex-automata crate spanning NFA compilation, DFA transition construction, Unicode handling, capture groups, and the hybrid matching engine.
schemathesis_datagen_pipeline	Implement eight feature targets in the Schemathesis Python API testing framework, including structured HTTP header generation strategies, coverage-phase hooks, discriminator-aware validation and data generation, and schema-driven code generation fixes.
schemathesis_reporting_observability	Implement five targets in Schemathesis: a post-validation hook system, multi-format test report writers (VCR, HAR, JUnit, NDJSON), pytest plugin integration, and schema-branch-aware example generation.
schemathesis_config_modernization	Implement six modernization targets in Schemathesis: a TOML-based configuration system with auto-discovery, API namespace reorganization, a metrics framework, transport and response abstractions, and a redesigned check registration system.
stream_processing_engine	Implement a Rust-based stream processing engine supporting windowed aggregations, filtering, projection, and stateful operators over JSON event streams. Scored on correctness, robustness to malformed input, and throughput.
tls13_handshake_state_machine	Complete a Python TLS 1.3 protocol state machine that processes handshake message traces, implementing state transitions, key schedule computation, and message validation using a provided crypto API.
vault_sdk_resilience	Implement five targets in HashiCorp Vault's Go SDK: a string template engine, a fair-share job scheduler, persistent cache storage for agent tokens, certificate utility enhancements, and API client request/response hooks.
vliw_kernel_optimization	Optimize a VLIW/SIMD kernel generator for correctness and minimum cycle count on a custom architecture simulator. Hard-coded answers for specific inputs are forbidden.
cpu_full_flow	Work through a full RISC-V CPU design curriculum: ISA emulator implementation, hardware abstraction layer, Verilog processor design, and SoC integration with Verilator simulation.
zstd_api_modernization	Implement five API evolution targets in Zstandard's C library: stable struct alignment and query functions, a generic parameter-driven compression interface, static allocation support, dictionary enhancements, and memory estimation utilities.
cfzip_compression_engine	Implement a complete compression engine from scratch in C++17 without external libraries: custom archive format, CLI with streaming mode, dictionary training and use, and integrity verification. Scored on compression ratio, speed, memory usage, and correctness.
Scientific Problems & ML	
battery_soh_rul_anomaly	Predict battery state-of-health, remaining useful life, and anomaly type/severity per cycle for unseen cells, given multi-cell degradation training data. Evaluated under distributional shift with regime and calibration changes not present in training.
borden_pump_treat_dispatch	Build a physics-based groundwater flow model and solve a constrained multi-objective optimization for pump-and-treat remediation on the Borden aquifer, selecting wells, treatment types, and multi-phase schedules under budget and capacity constraints.
borden_source_inversion	Infer a finite-duration rectangular contaminant source from sparse, noisy monitoring-well observations in a 3D hydrogeological scene. The agent must implement its own forward model and inversion optimizer from scratch.
borden_sensor_fault_diagnosis	Classify sensor faults (spikes, drift, stuck-at-zero, unit errors, etc.) versus true plume arrivals in groundwater monitoring records using physics-informed checks such as travel-time ordering and neighbor-well consistency.
bridge_gnss_state_forecast	Process dirty bridge GNSS displacement time series (timestamp jitter, duplicates, spikes, drift) and produce cleaned reconstruction, state estimates, and short-term displacement forecasts.
capecod_plume_reconstruction	Reconstruct a multi-analyte groundwater plume from sparse monitoring wells: predict concentrations at withheld locations and times, compute plume metrics, and propose an optimal monitoring network under budget constraints.
vsg_stability_parameter_optimization	Integrate current-limited Virtual Synchronous Generators into the IEEE 39-bus power system, generate transient stability data, build a physics-informed neural network, and train a reinforcement learning agent to optimize VSG parameters.
cylinder_wake_prediction	Implement a CPU-only 2D cylinder wake solver for the incompressible Navier-Stokes equations. Evaluated on unseen Reynolds numbers and domain configurations; scored on velocity-field accuracy, pressure-field accuracy, and flow-regime prediction.
dabic_gravity_inversion	Implement the D-ABIC regularization method for 3D gravity inversion within the SimPEG framework, run on both synthetic and real Vinton salt dome data under L0 and L1 sparse norms, and compare against a Hamiltonian Monte Carlo baseline.
noisy_product_matching_pipeline	Determine whether pairs of product listings from different sources refer to the same real-world item, given noisy and incomplete attributes. Evaluated on hidden variants with different noise characteristics and data scales; no ground-truth labels are available during development.
neural_net_weight_recovery	Reconstruct the correct layer ordering of a neural network from 97 shuffled weight files and input-output historical data, using dimensional constraints and reconstruction-error analysis.
nanophotonic_simulation_reproduction	Reproduce published nanophotonic simulation results (multi-source electromagnetic field distributions) from a research paper, implementing the solver from scratch using only NumPy.
ftir_polymer_identification	Identify polyimide monomers from FTIR spectra by correlating IR absorption peaks with functional groups, optionally aided by quantum chemistry simulations. Submission attempts are rate-limited to prevent brute-force enumeration.
molecular_property_regression	Predict molecular properties from graph representations without graph neural network libraries. Evaluated on multi-domain data with perturbations that penalize solutions overfitting to the development distribution.
graph_node_classification	Implement graph neural networks from scratch using only base PyTorch for semi-supervised node classification on an unseen graph under CPU-only constraints.

Continued on next page

Task	Design notes
gravitational_wave_signal_detection	Reproduce the LIGO GW150914 gravitational-wave detection pipeline: whitening, bandpass filtering, matched filtering against a numerical-relativity template, time-frequency analysis, and residual computation.
herbal_depression_target_screening	Screen active components and protein targets for four traditional Chinese medicine herbs used in depression treatment, producing component lists, target mappings, disease-target intersections, and a PPI network visualization.
polyimide_homo_lumo_prediction	Compute HOMO/LUMO energies of polyimide repeating units using DFT calculations, analyze substituent and conjugation effects on electronic properties, and identify the monomer pair with the widest band gap.
industrial_anomaly_detection	Detect anomalies in multivariate industrial sensor time series using only classical ML libraries. Evaluated on hidden variants including time-series crops, reversals, and normal-only segments to prevent hard-coding.
bipedalwalker_locomotion_rl	Train a CPU-only locomotion policy for BipedalWalker and its Hardcore variant. The judge evaluates only the submitted policy checkpoint, not the training process. Pre-trained policies and external RL libraries are prohibited.
molecular_solubility_prediction	Predict aqueous log-solubility from SMILES strings and molecular descriptors, improving upon a provided random forest baseline. Scored by prediction error on hidden test molecules.
motor_clutch_model_reproduction	Implement a Gillespie/KMC stochastic simulation of the motor-clutch mechanotransduction model from sparse reference traces. Curve-fitting formulas and lookup tables are prohibited; the simulation must exhibit correct stochastic dynamics across unseen parameter combinations.
streaming_multilabel_classification	Implement streaming multi-label classification from scratch using only NumPy. Evaluated on subset accuracy, Hamming loss, and F1 metrics under CPU-only constraints.
barnes_hut_nbody_acceleration	Implement a Barnes-Hut N-body gravitational simulation in C++17. Scored on both force accuracy relative to direct summation and speedup over a naive baseline across varying particle counts.
blackbox_numerical_integration	Integrate hidden black-box functions over the 10-dimensional unit hypercube via an oracle interface with bounded query budgets. Scored on accuracy relative to true integral values.
pancreatic_radiotherapy_meta_analysis	Automate a two-stage systematic review pipeline: screen candidate PDFs and extract evidence, then perform inverse-variance meta-analysis with model selection and subgroup analysis. Evaluated on unseen publications.
monge_ampere_pde_solver	Implement a numerical solver for the fully nonlinear Monge-Ampère equation $\det(D^2u) = f(x, y)$. Evaluated on unseen right-hand sides and boundary conditions; scored on solution accuracy and computational efficiency.
pdf_structured_extraction	Extract structured page blocks (text, tables, formulas, figures) with bounding boxes, reading order, and content from PDFs using only classical computer vision tools—no deep learning models. Evaluated on diverse enterprise document layouts.
ocean_mt_lab_inversion	Implement 1D marine magnetotelluric forward modeling and Bayesian inversion with lateral coupling across ocean-bottom stations. Independent per-station or point-estimate-only solutions are penalized.
pv_power_forecasting	Forecast multi-site photovoltaic power generation from historical output and weather features. Evaluated on unseen data across multiple domain-specific metrics including ramp handling, peak accuracy, and multi-horizon performance.
quantum_architecture_search	Implement noise-adaptive quantum architecture search balancing classification accuracy and molecular ground-state energy estimation against hardware-realistic circuit depth and gate count constraints.
collaborative_filtering_recommender	Implement a top-K recommender system that jointly optimizes recommendation quality (NDCG) and runtime efficiency, including cold-start users with no training interactions.
touchstone_vna_diagnostics	Parse Touchstone S-parameter files in various representations (real-imaginary, magnitude-angle, dB-angle), compute derived RF metrics (return loss, VSWR, group delay, impedance), and produce structured diagnostic reports.
ecg_signal_processing_pipeline	Implement a three-stage ECG processing pipeline (denoising, QRS complex detection, beat classification) from scratch using only NumPy. Evaluated on unseen recordings with different noise profiles and arrhythmia distributions.
sketch_solve_least_squares	Solve large-scale overdetermined least-squares problems by choosing among direct solvers, iterative methods, and randomized sketching based on matrix properties. Scored on solution accuracy and speed across varied problem structures.
substrate_interface_simulation	Implement a coupled interface response simulation with correct parameter dependence. Hard-coded outputs are prohibited; evaluated on physical consistency, energy conservation, and statistical properties of stochastic trajectories.
thermo_fluid_field_prediction	Predict 2D velocity and temperature fields for thermo-fluid coupling problems under varying boundary conditions and dimensionless parameters, using only NumPy on CPU.
csi_time_series_forecasting	Forecast future wireless channel state information tensors from historical observations. Evaluated on unseen channel conditions and mobility scenarios under CPU-only constraints.
roof_damage_active_learning	Design an active learning pipeline for satellite-imagery roof damage detection: start from a small labeled seed set, strategically query an oracle for additional labels under a fixed budget, and train an object detector for evaluation on unlabeled images.
Combinatorial Optimization	
ad_placement_optimization	Partition a large integer grid into non-overlapping rectangles, each containing a designated anchor point, maximizing total satisfaction from how closely each rectangle's area matches its target.
treant_forest	Strategically place obstacles in a grid maze to maximize the shortest-path length between start and goal, or block the path entirely.
grid_turing_robot	Design transition rules and initial coloring for a Turing-like robot on a colored grid to maximize the number of distinct cells visited while minimizing the rule set size.

Continued on next page

Task	Design notes
molecular_self_assembly	Schedule bonding operations over discrete time steps to assemble atoms into a specified number of connected molecules, respecting spatial proximity and temporal ordering constraints.
apple_incremental_game	Decide each turn whether to invest in machines or collect output in an incremental production game, balancing short-term gains against long-horizon compounding.
first_order_theorem_prover	Build a first-order automated theorem prover from scratch (parsing, clausification, unification, saturation) that produces verified proof or model witnesses. External provers and benchmark fingerprinting are forbidden.
circuit_layout_optimization	Implement a VLSI standard-cell placement solver minimizing half-perimeter wire length on industry-standard benchmarks. Scored on wire-length quality and runtime.
order_addition_permutation_optimization	Find a permutation of 1,000 elements that minimizes a black-box cost function, using metaheuristic search (simulated annealing, genetic algorithms, local search) without access to the cost function's internals.
equivalence_class_divide_and_conquer	Solve six progressive competitive-programming problems centered on equivalence classes and divide-and-conquer, where techniques from earlier problems inform solutions to harder ones.
jagua_nesting_optimization	Improve a Rust-based 2D irregular bin packing optimizer for non-convex polygonal pieces. Solution geometry is independently verified; improvements below a minimum threshold receive no credit.
sat_solver	Build a SAT solver from scratch implementing conflict-driven clause learning, watched literals, and restart strategies. External solvers and benchmark-aware heuristics are forbidden; scoring is balanced across difficulty tiers.
smt_solver	Build an SMT solver from scratch for four quantifier-free theories (uninterpreted functions, linear real and integer arithmetic, and their combination). External SMT solvers are forbidden; model witnesses are independently validated.
symbolic_integration_engine	Extend a starter symbolic integration engine (with its own parser, simplifier, and differentiator) to handle a broader class of integrands. All computer algebra systems and numerical libraries are forbidden; correctness is verified by differentiating the returned antiderivative.
tree_block_partitioning	Solve six progressive problems on tree decomposition and block partitioning, where algorithmic ideas discovered in simpler variants transfer to harder ones.
triangulation_coloring_optimization	Minimize a cost function over a triangulation by jointly recoloring vertices and flipping edges, where the dominant term is a quadratic penalty on monochromatic ("ugly") triangles.
vibrating_path_graph_coloring	Color graph vertices and selectively remove edges to minimize a cost that penalizes both removed edges and monochromatic surviving edges.
vehicle_routing_time_windows	Implement a capacitated vehicle routing solver with time windows for Solomon-style benchmarks. Scored against best-known solutions on vehicle count and total travel distance.
warehouse_forklift_routing	Route a forklift in a grid warehouse to receive goods arriving in random order, store them, and dispatch them in sequential order, minimizing total movement.
wireless_electricity_layout	Position wire segments on a 2D plane to deliver wireless electricity from two fixed source plates to thousands of cities, minimizing a quadratic cost over city-to-wire distances and wire displacements while avoiding short circuits.
Formal Math & Theorem Proving	
lean_analysis_proofs	Complete proof obligations across a multi-file Lean 4 project formalizing results in real and functional analysis. Proofs are checked transitively: a theorem counts only if its entire dependency chain is fully proved.
carleson_formalization	Fill proof obligations in the Lean 4 formalization of Carleson's theorem on pointwise convergence of L^2 Fourier series. Transitive axiom checking ensures no dependence on unproved prerequisites.
combinatorial_games_formalization	Resolve proof obligations in a Lean 4 formalization of combinatorial game theory, covering surreal numbers, game arithmetic, and the Sprague-Grundy theorem.
new_foundations_consistency	Complete proof obligations in the ConNF Lean 4 project formalizing the consistency of Quine's New Foundations, involving permutation models and tangled type theory.
cup_product_formalization	Fill proof obligations in a Lean 4 formalization of the cup product in singular cohomology: cochain-level multiplication, the Leibniz rule, and the induced ring structure on cohomology groups.
erdos392_formalization	Complete proof obligations for Erdős Problem 392 (asymptotic prime distribution) within a Lean 4 analytic number theory project. Weighted scoring reflects relative difficulty of each proof.
flt_regular_formalization	Resolve proof obligations in a Lean 4 formalization of Fermat's Last Theorem for regular primes via Kummer's cyclotomic theory. Top-level results earn no credit unless foundational prerequisites are also fully proved.
godel_incompleteness_formalization	Complete proof obligations in a Lean 4 formalization of Gödel's First Incompleteness Theorem, spanning Gödel numbering, the fixed-point lemma, and the self-referential undecidable sentence.
medium_prime_number_theorem	Complete proof obligations for the Prime Number Theorem with an explicit error term, requiring complex-analytic techniques including contour integration and zero-free regions of the Riemann zeta function.
ordinal_notation_well_foundedness	Construct well-foundedness proofs for ordinal notation systems in Coq, involving Cantor Normal Form and ordinal arithmetic.
pfr_formalization	Resolve proof obligations in the Lean 4 formalization of the Polynomial Freiman–Ruzsa conjecture (Gowers–Green–Manners–Tao 2023), involving Shannon entropy, Ruzsa distance, and subgroup covering arguments.
sphere_eversion_formalization	Complete proof obligations in a Lean 4 formalization of sphere eversion, spanning smooth immersions, jet bundles, ample differential relations, and convex integration.
turing_machine_halting_proofs	Prove halting behavior for specific 6-state, 2-symbol Turing machines in Coq, contributing to the Busy Beaver frontier. Proofs must be mechanically verified with no admitted axioms.

Continued on next page

Task	Design notes
Professional Knowledge Work	
portfolio_risk_calibration	Implement a multi-module portfolio management system (risk calibration, constrained optimization, execution cost modeling, dynamic rebalancing) for a cross-asset ETF portfolio. Evaluated out-of-sample on risk-adjusted return metrics.
storyboard_ad_copywriting	Produce a promotional video script with dual variants and a shot-by-shot storyboard for a state-owned enterprise exhibition appearance, adhering to strict political communication standards and advertising compliance.
brand_annual_planning_ppt	Produce a comprehensive annual brand management plan as a presentation deck and companion data tables for a mid-size IPTV company, covering strategy, monthly action roadmaps, budgets with market rate benchmarks, and competitive analysis.
securities_protection_training	Produce a complete set of training deliverables for a securities investor protection seminar: legal framework overview, core systems analysis, case studies, comparative study, policy analysis, presentation deck, lecture script, and bibliography.
college_english_exam_bank	Produce five parallel examination papers with answer keys for a college English course, plus a blueprint table and an overlap self-check matrix ensuring cross-paper diversity meets pedagogical thresholds.
cross_border_commission_compliance	Produce a multi-module legal compliance report for a cross-border commission payment, covering multi-jurisdictional anti-corruption risk (FCPA, UK Bribery Act, Chinese Criminal Law), export control, transfer pricing, and evidence preservation strategy.
cross_border_investment_ppt	Produce a presentation covering 15 months of global cross-border investment policy developments with chronological policy inventory, original data charts, multi-sector industry analysis with deal case studies, and forward-looking risk assessment.
cta_risk_budget_optimization	Build a complete CTA multi-strategy futures trading system: multiple signal classes, dynamic risk budgeting, a multi-currency backtest engine with transaction costs, drawdown control, and performance attribution.
equity_objection_report	Produce a court-submission-ready legal research report analyzing whether beneficial owners under equity proxy-holding arrangements can defeat forced execution in Chinese civil enforcement law, citing specific judicial guidance and case precedents.
expo_visitor_conversion_model	Clean tens of thousands of messy visitor registration records (entity resolution, address parsing, company-name merging), tag professional visitors, and build a calibrated exhibitor conversion scoring model.
factor_stock_model_optimization	Implement an adaptive multi-factor stock selection model with IC-based factor selection, dynamic weighting, industry/market-cap neutralization, and constrained portfolio construction. Evaluated out-of-sample on information ratio, excess return, drawdown, and turnover.
global_terrorism_atlas_report	Transform a 200K+ record terrorism database into a single-PDF infographic atlas with maps, ranking charts, shaped word clouds, a sunburst diagram, and country-level fatality trends.
hebei_gaokao_strategy_report	Produce a multi-tier college admission preference plan with alternative strategies, given complex personal constraints (banned fields, health disqualifications, career priorities) and three years of historical admission data.
hk_connect_annual_metrics	Programmatically retrieve and populate financial metrics for 255 Hong Kong Stock Connect-eligible securities from annual reports and market data providers. Scored on per-cell accuracy.
k12_math_recommendation	Build a knowledge-tracing and question-recommendation system from hundreds of thousands of student interaction records, evaluated on prediction accuracy, mastery calibration, learning gain, and pedagogical constraint satisfaction.
property_actuarial_pricing	Build an actuarial pricing model for SME property insurance: loss modeling, risk grading, reinsurance cost allocation, and premium smoothing with renewal constraints. Evaluated on predictive accuracy and premium adequacy.
real_estate_bid_estimate	Conduct feasibility analysis for a supertall office land tender: market research, quarterly cash flow models under two bidding structures, sensitivity analysis, and a reasoned investment recommendation.
stock_momentum_backtest	Fetch live A-share market data, compute risk-adjusted momentum scores, apply multi-factor screening filters, and calculate a market-cap-weighted portfolio return for a specific holding period. Scored on numerical accuracy of each computation step.
storm_claim_ring_audit	Build a fraud detection pipeline for post-disaster insurance claims: produce hold/release/escalate decisions and fraud-ring cluster assignments from interconnected claim, payment, survey, and media datasets, distinguishing genuine disaster patterns from coordinated fraud.
Interactive Games & Simulators	
dcss_dungeon_ai	Write a Lua bot for Dungeon Crawl Stone Soup that autonomously explores, fights, and descends dungeon levels as a Minotaur Berserker under a wall-clock time budget. Scored by mean in-game score across multiple runs.
anchorhead_text_adventure	Play the Lovecraftian interactive fiction game *Anchorhead* via an HTTP API, sending text commands and receiving prose observations. Scored by peak in-game score, reflecting progression through the multi-day narrative and puzzle chain.
trinity_text_adventure	Play Infocom's *Trinity* via an HTTP game API. The game requires precise object manipulation and understanding of symbolic and temporal clues across interconnected zones.
tryst_text_adventure	Play *Tryst of Fate* via an HTTP game API. The branching narrative with irreversible choices requires strategic exploration to reach high-scoring endings.
nethack_dungeon_agent	Implement a decision policy for NetHack via the NLE harness, parsing ASCII map observations and stat vectors to navigate, fight, and survive across multiple procedurally generated runs.
openrct2_theme_park_ai	Write a JavaScript plugin for OpenRCT2 that autonomously builds rides, hires staff, sets pricing, and grows park company value across multiple scenarios of increasing complexity.

Continued on next page

Task	Design notes
`openttd_transport_ai`	Write an AI script for OpenTTD that builds road, rail, and air transport networks to connect towns and industries and grow company value across diverse procedurally generated maps.
`wesnoth_tactical_ai`	Write tactical AI logic for Battle for Wesnoth that defeats the built-in AI through custom recruitment, focus-fire targeting, terrain exploitation, and village-capture timing across multiple maps.

Table 7 Per-task design notes for all 134 EdgeBench tasks.

G.5 Per-Task Learning Curves

Figures 15–35 show learning curves for all tasks in the benchmark, grouped by capability family. Each plot shows raw task score versus elapsed time over the 12-hour budget; solid lines are the best-so-far envelope, faint dashed dots are individual submissions, and pale segments after the marked best indicate later submissions without improvement. On a few tasks the y-axis is zoomed to the main score band, with extreme outlier submissions off-axis (noted under each plot). Five agents are compared: Claude Opus 4.8, GPT-5.5, GPT-5.4, GLM-5.1, and DS-V4-Pro. The 18 representative tasks in the main text (Figure 4) are not repeated here.

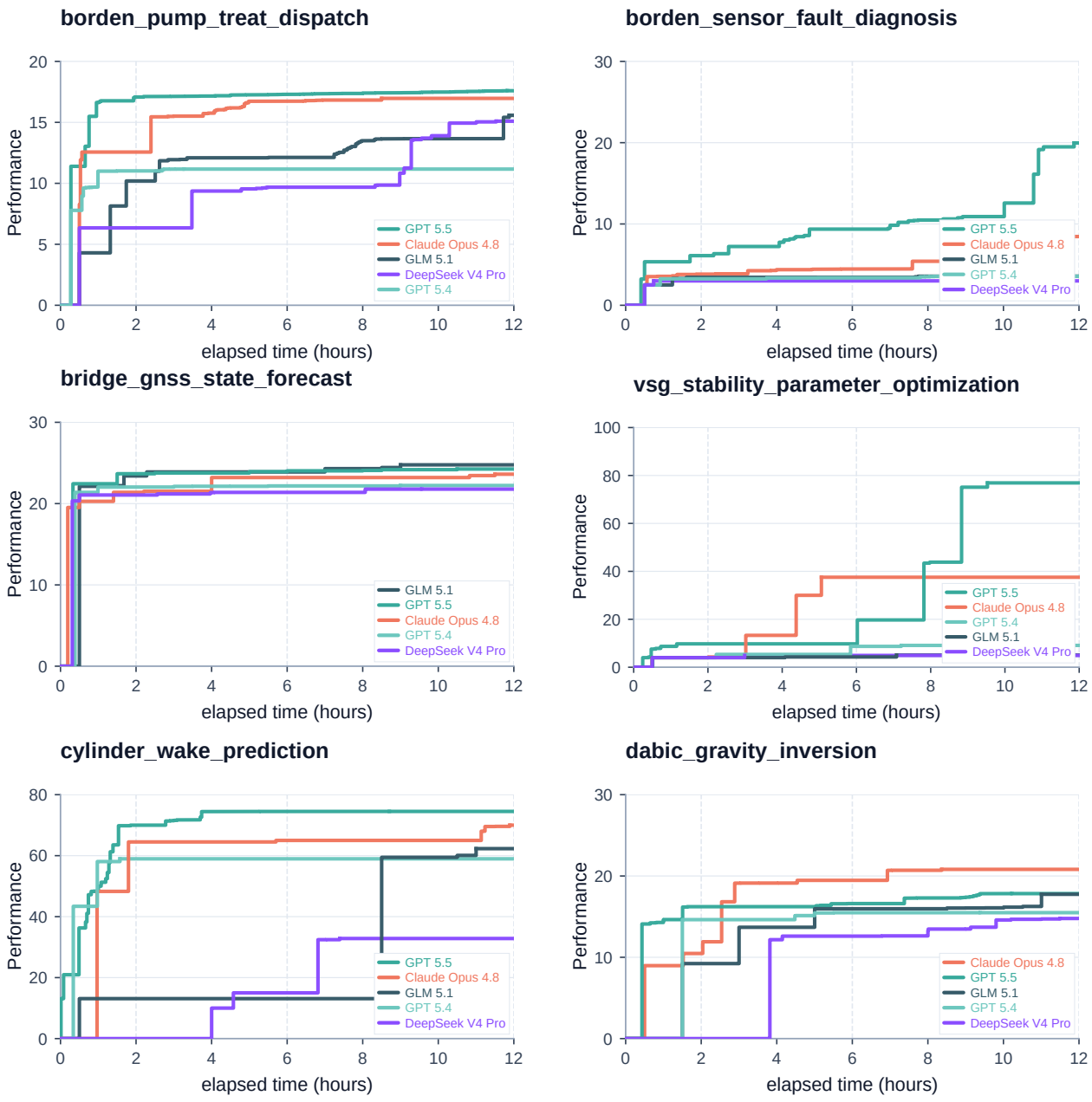


Figure 15 Per-task learning curves: Scientific Computing & ML (1/21).

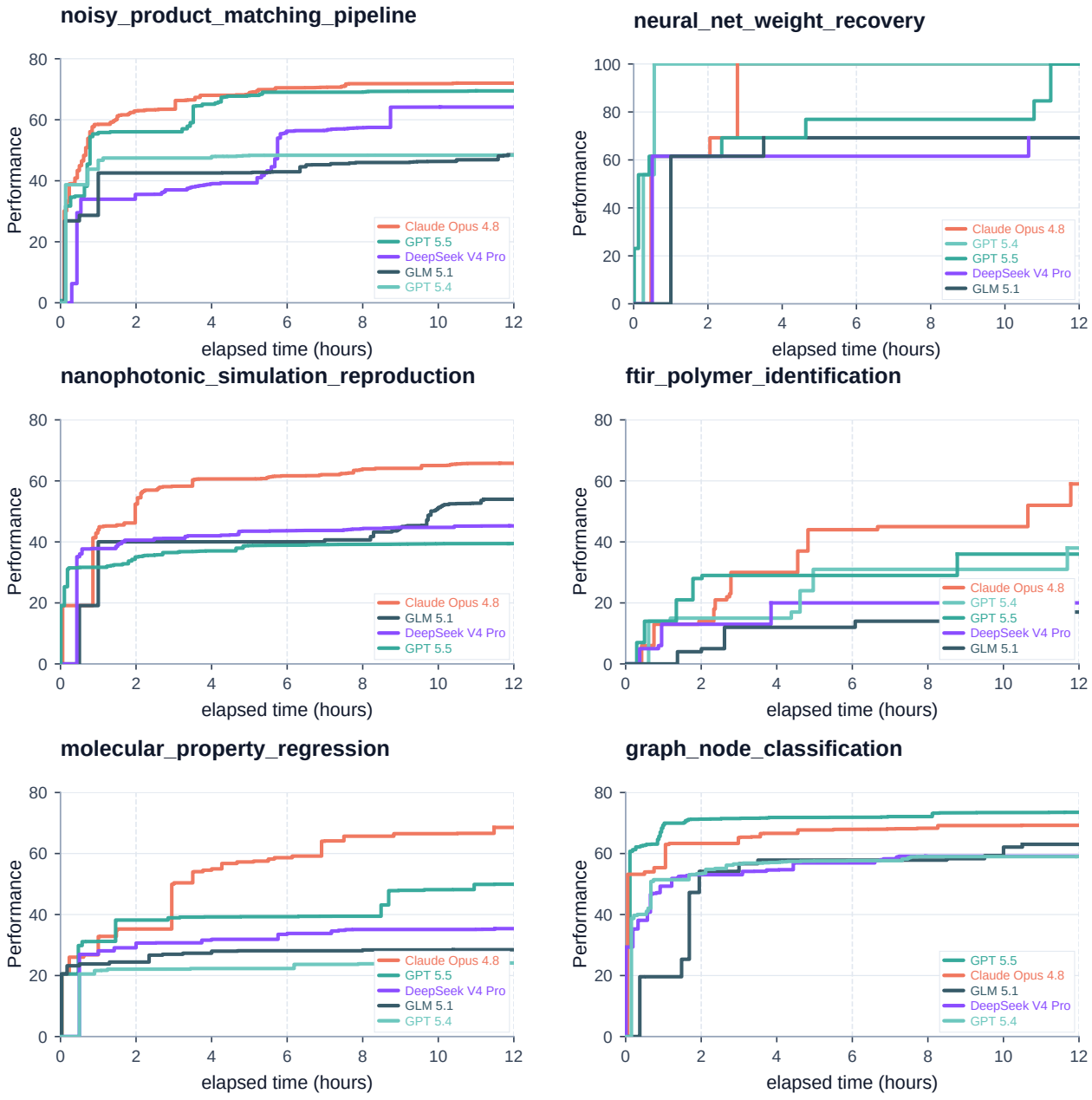


Figure 16 Per-task learning curves: Scientific Computing & ML cont. (2/21).

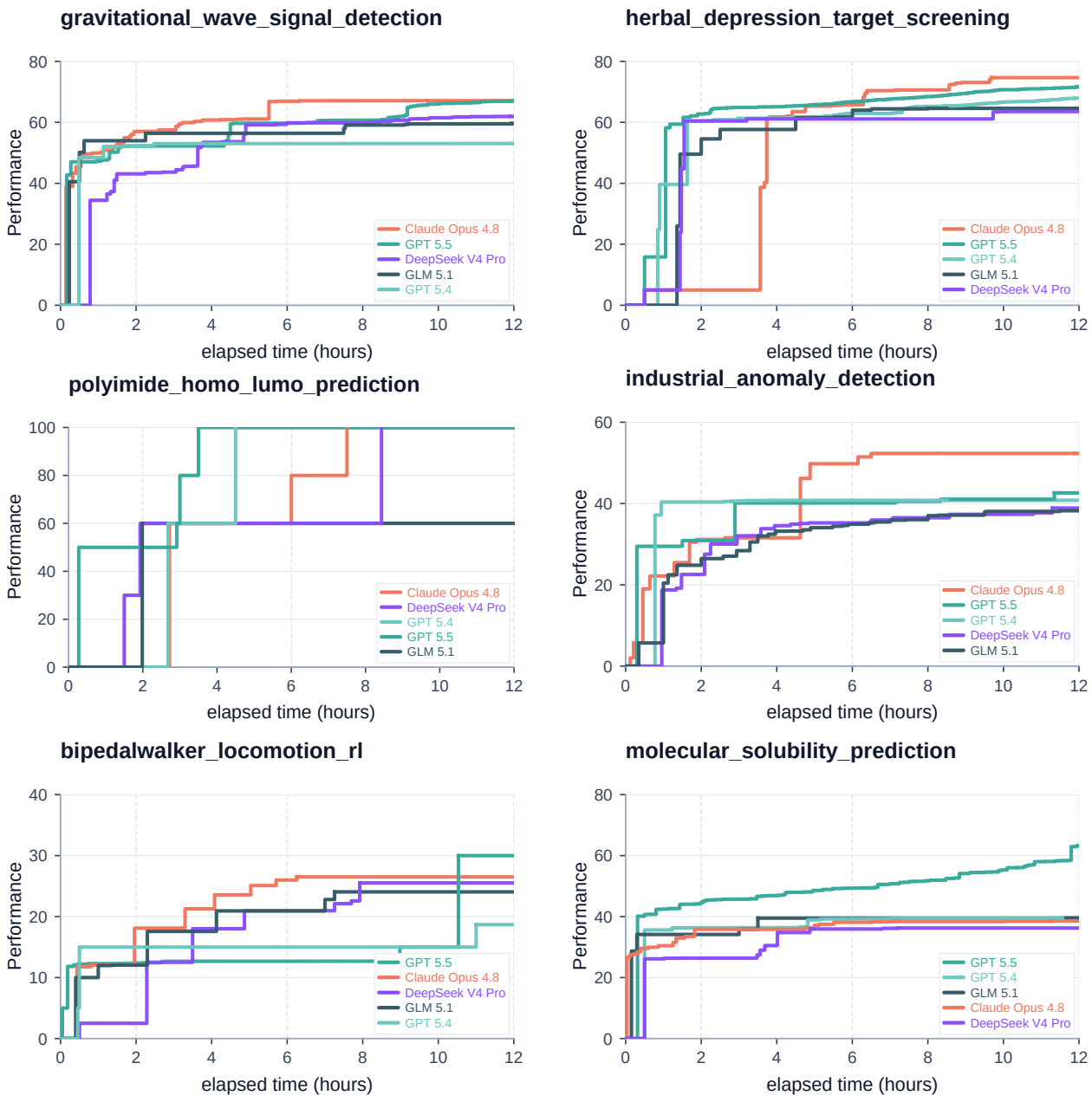


Figure 17 Per-task learning curves: Scientific Computing & ML cont. (3/21).

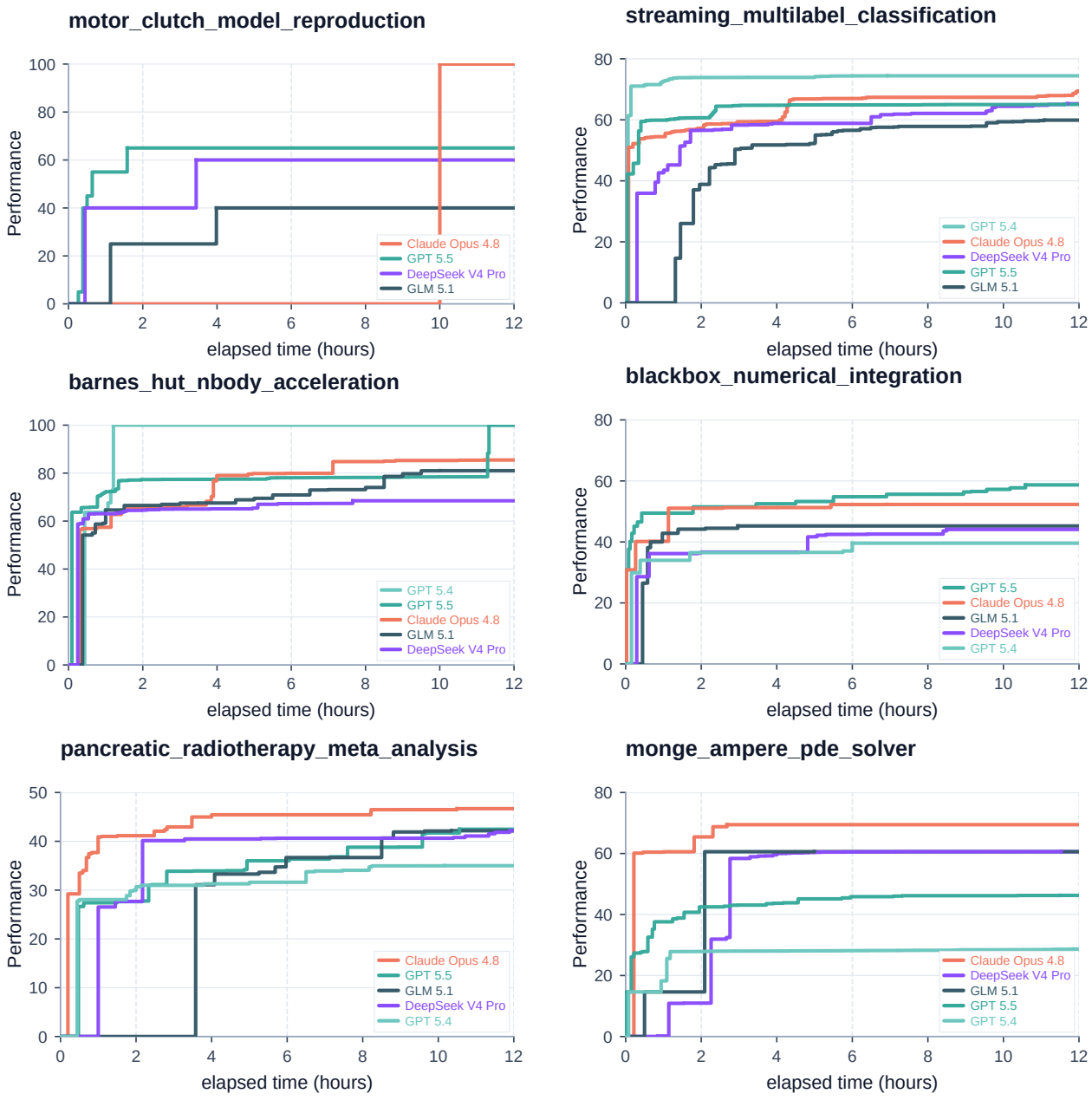


Figure 18 Per-task learning curves: Scientific Computing & ML cont. (4/21).

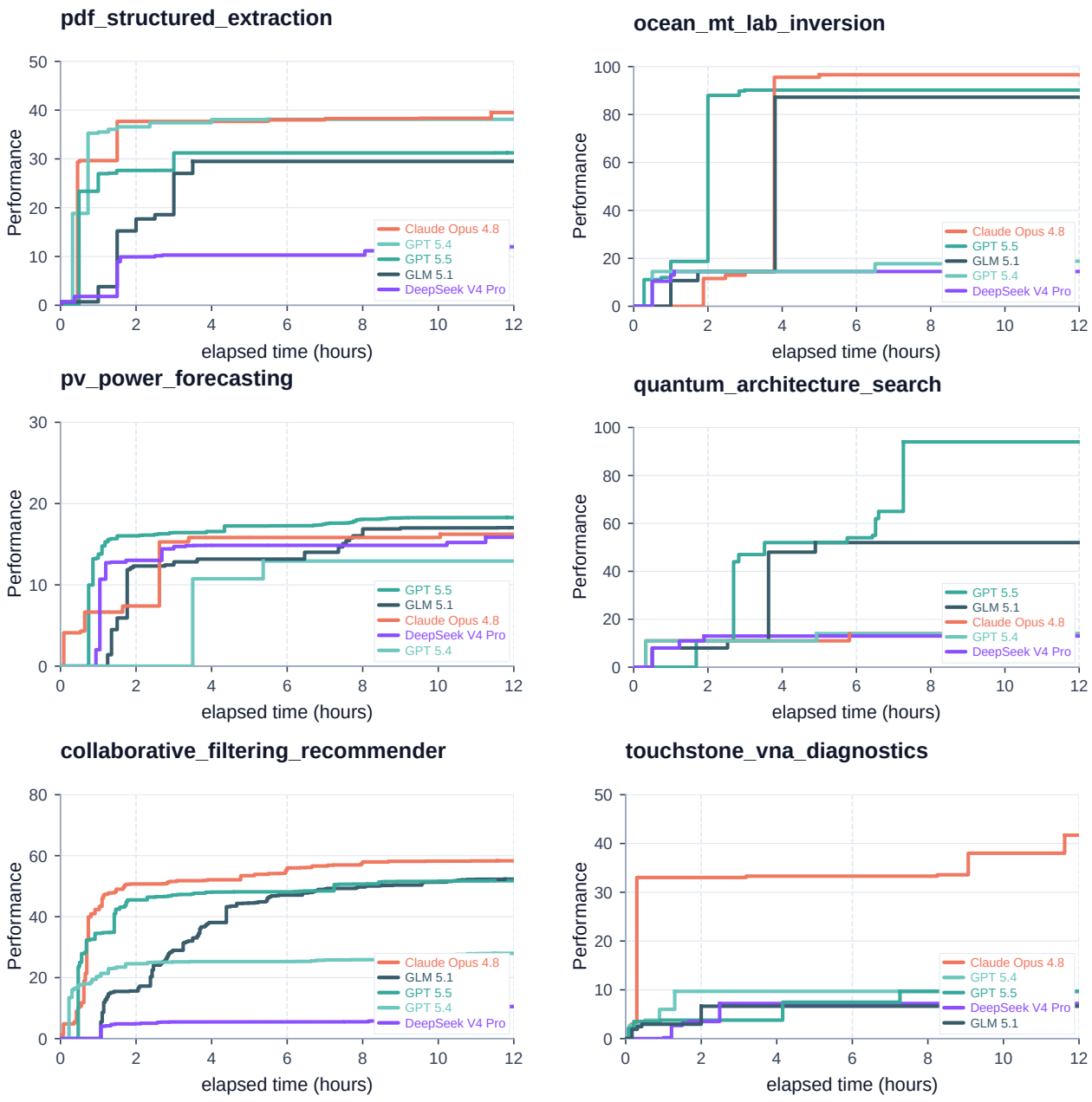


Figure 19 Per-task learning curves: Scientific Computing & ML cont. (5/21).

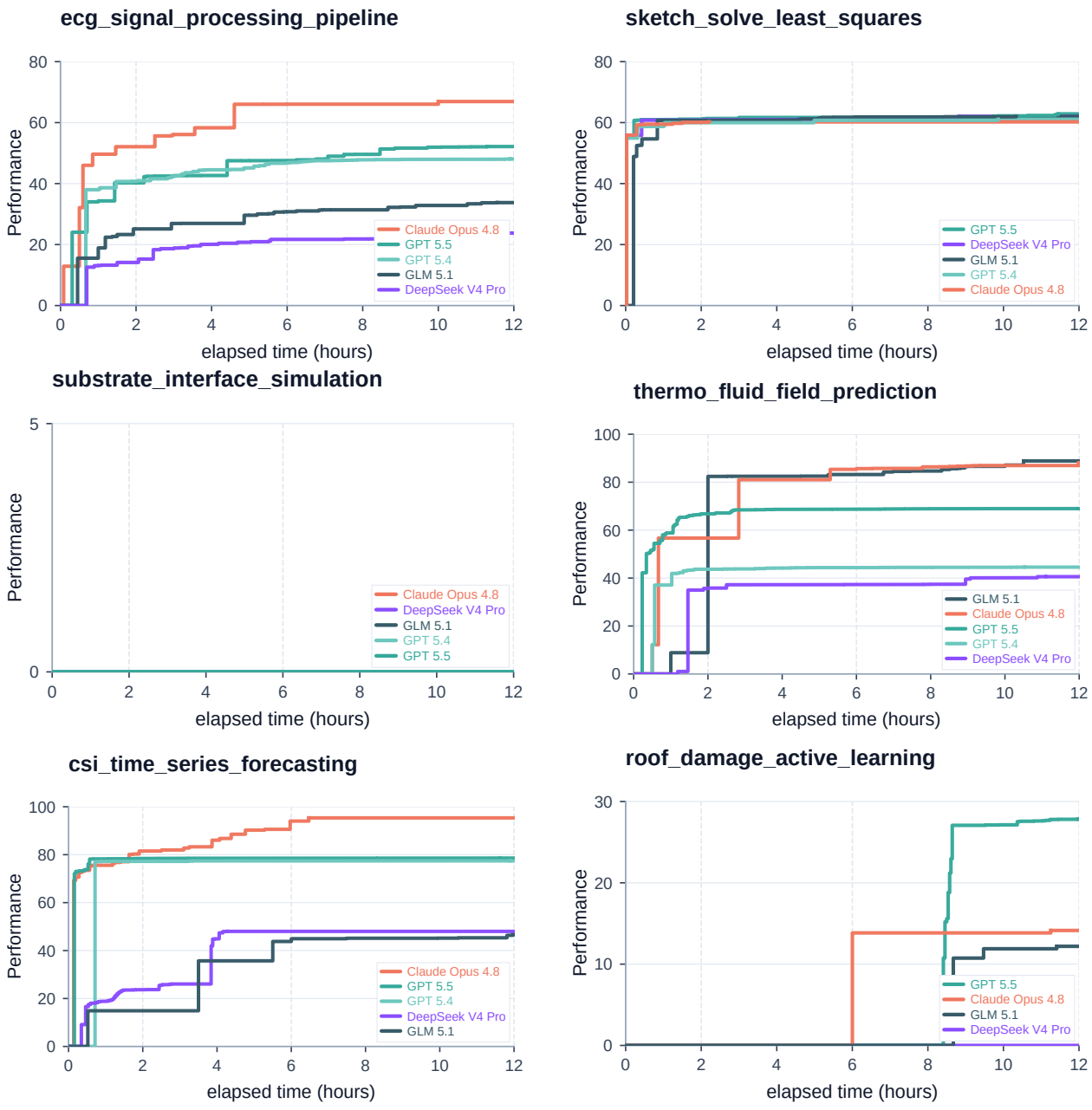


Figure 20 Per-task learning curves: Scientific Computing & ML cont. (6/21).

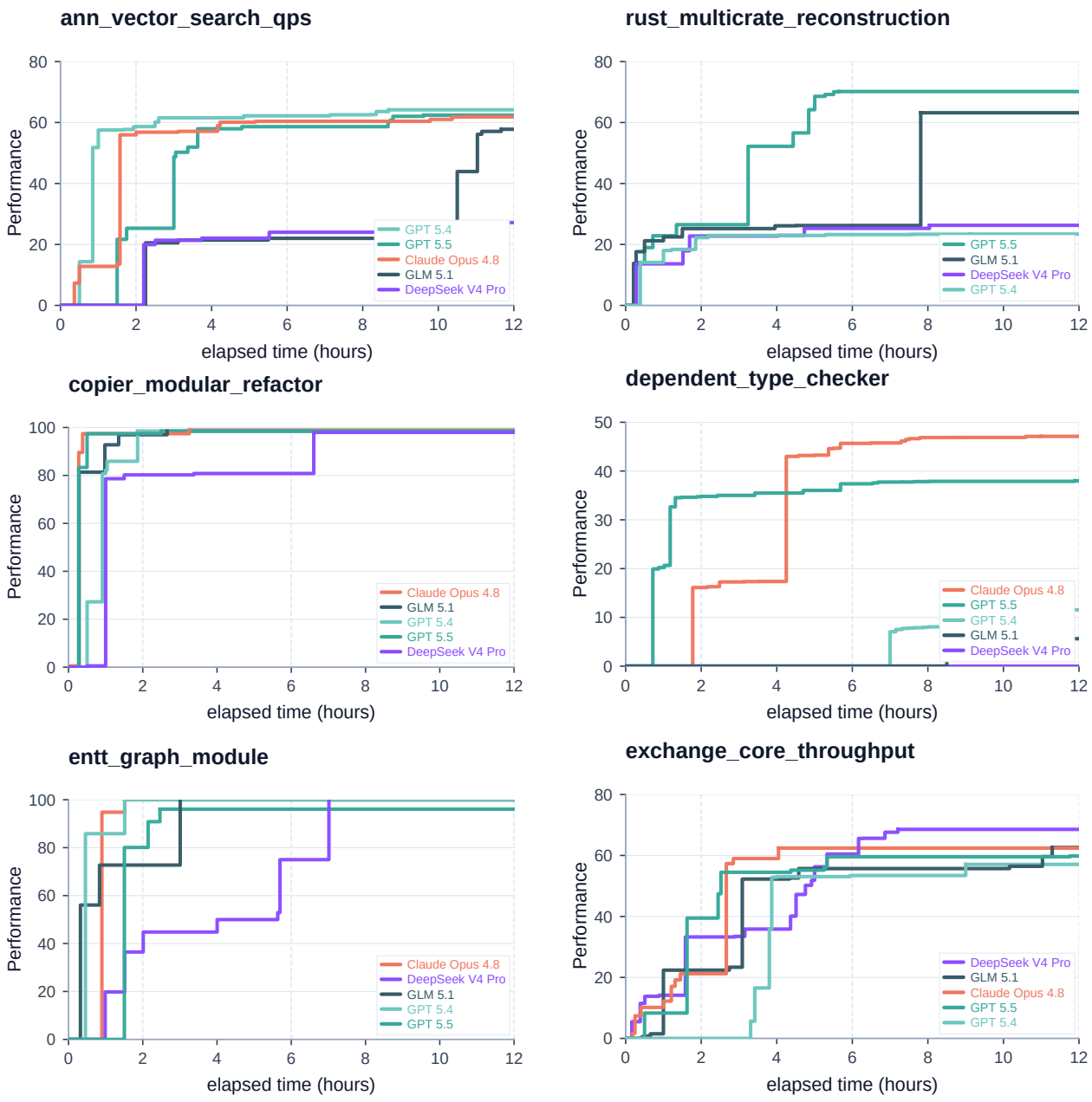


Figure 21 Per-task learning curves: Systems & Software Engineering (7/21).

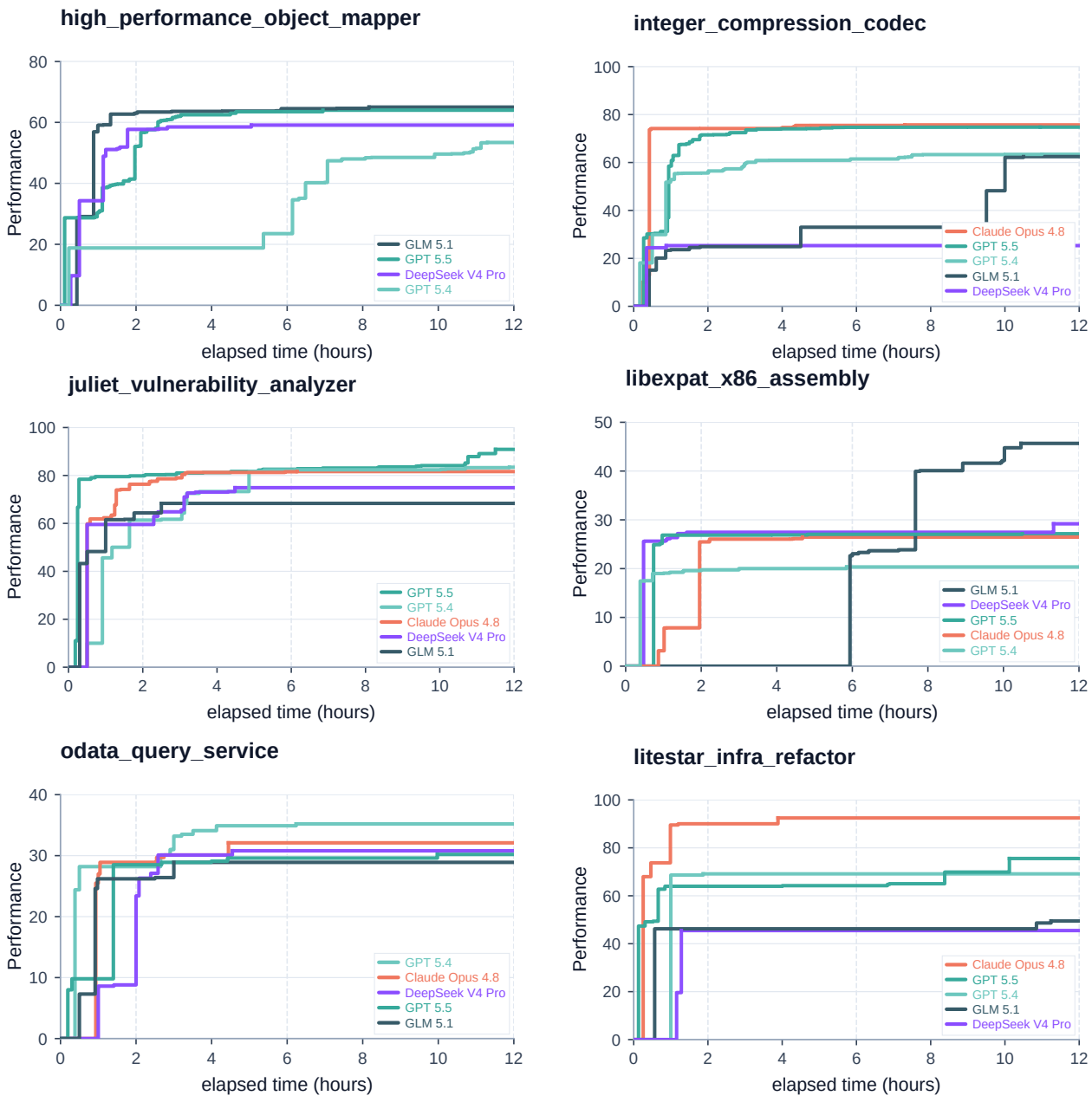


Figure 22 Per-task learning curves: Systems & Software Engineering cont. (8/21).

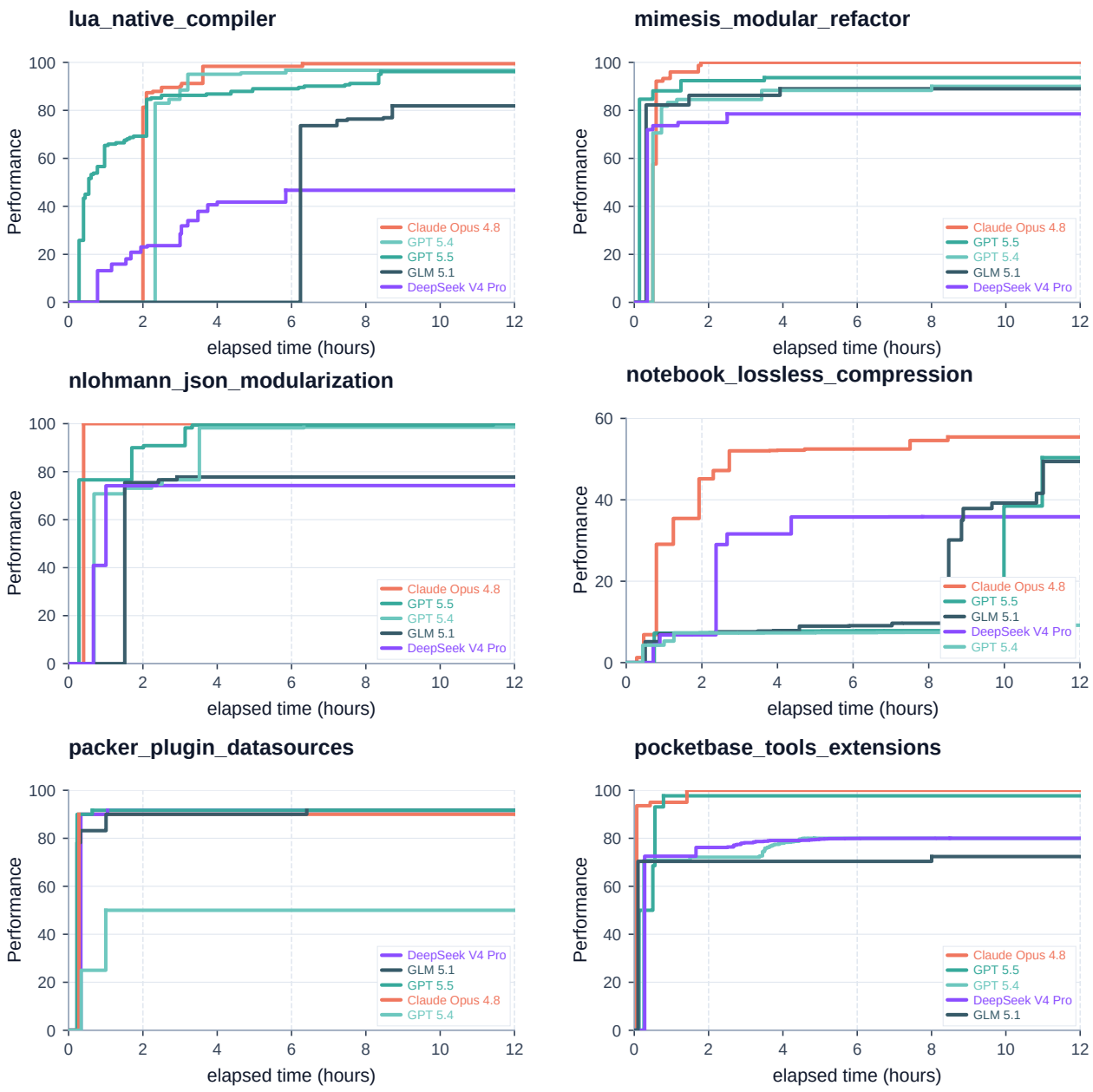


Figure 23 Per-task learning curves: Systems & Software Engineering cont. (9/21).

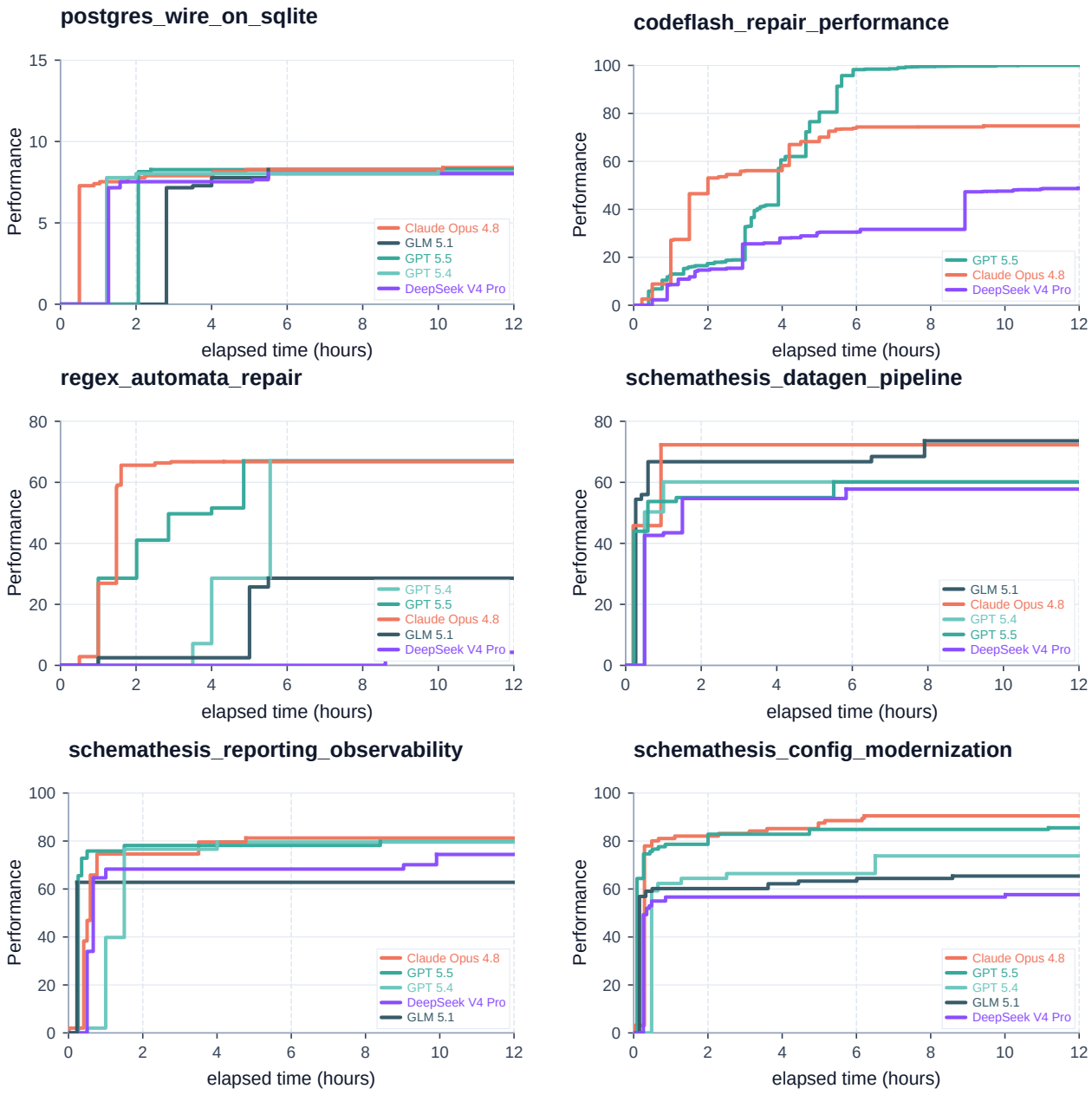


Figure 24 Per-task learning curves: Systems & Software Engineering cont. (10/21).

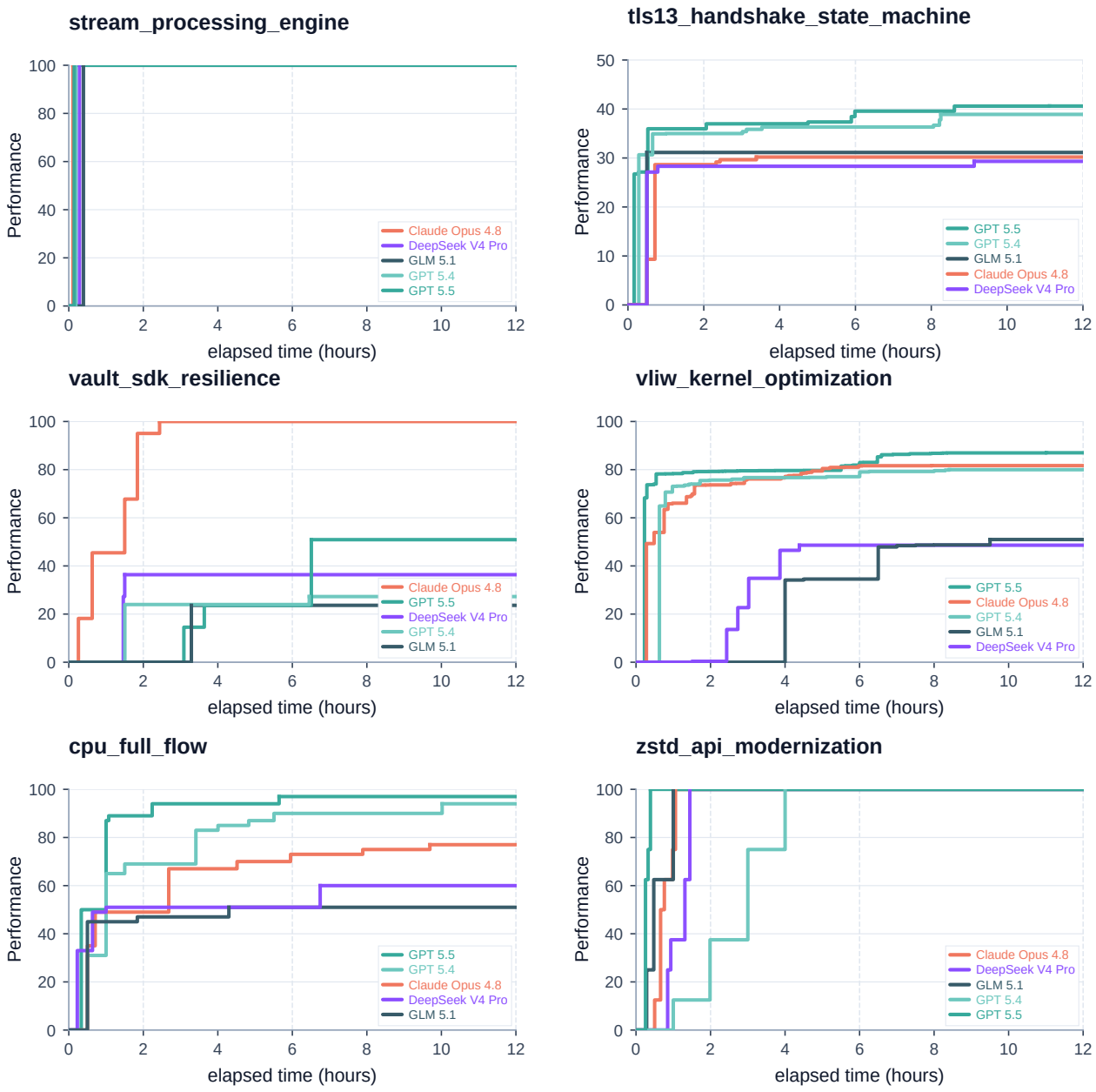


Figure 25 Per-task learning curves: Systems & Software Engineering cont. (11/21).

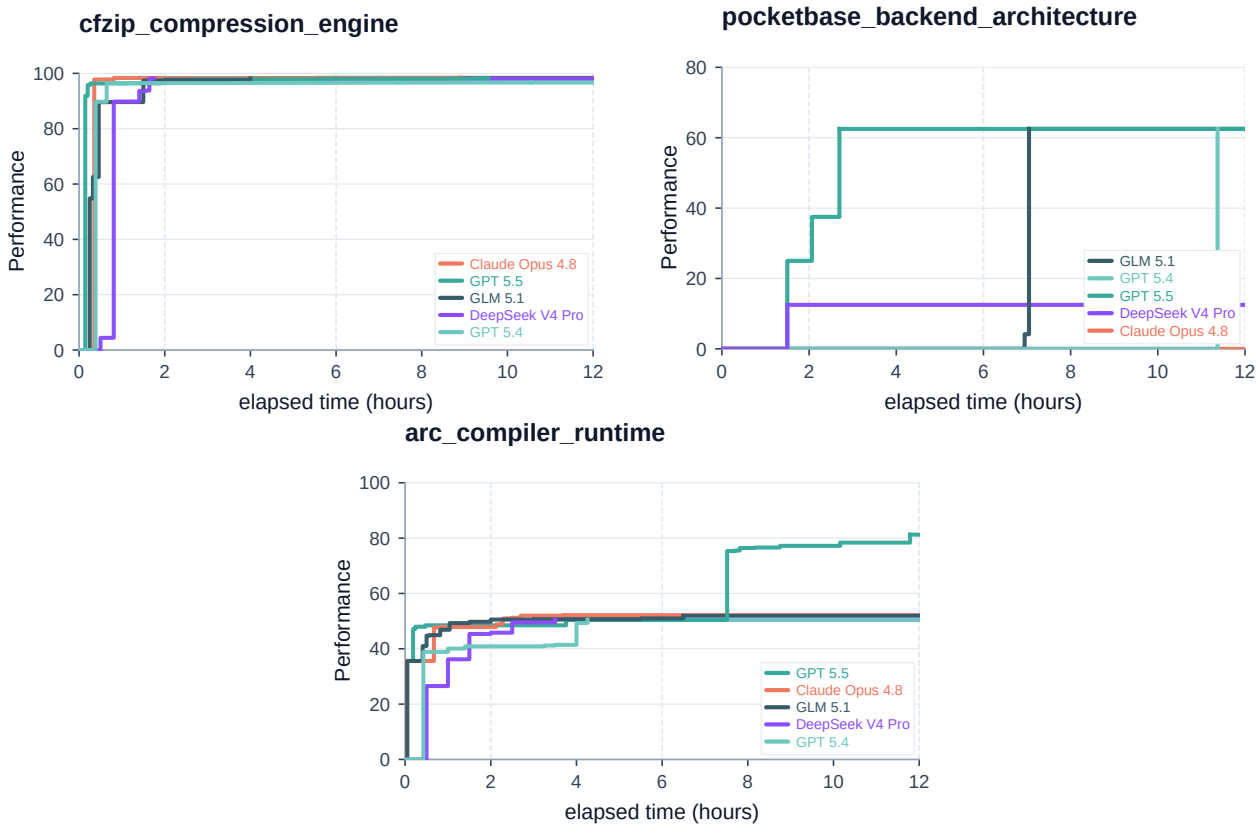


Figure 26 Per-task learning curves: Systems & Software Engineering cont. (12/21).

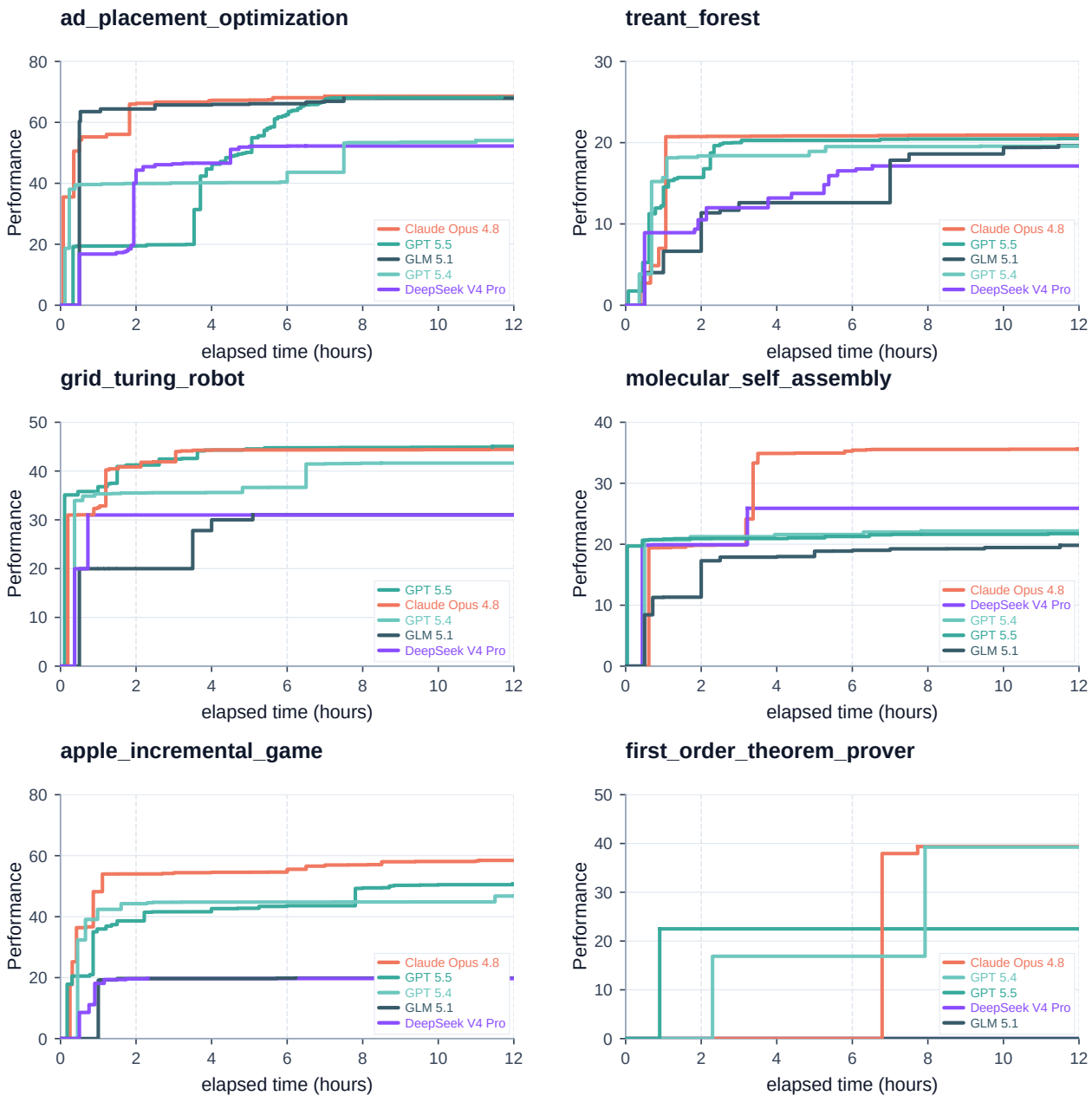


Figure 27 Per-task learning curves: Combinatorial Optimization & Planning (13/21).

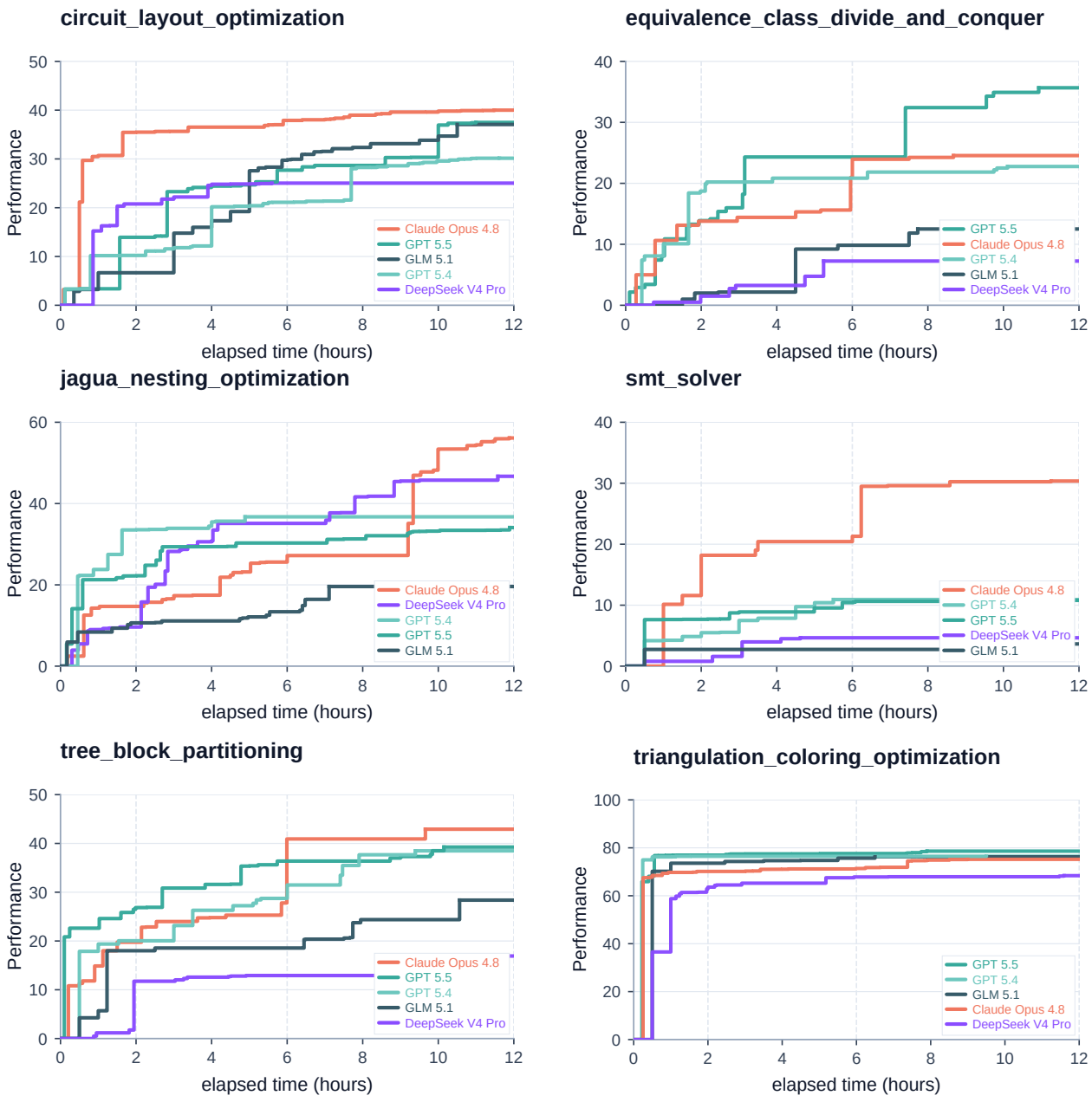


Figure 28 Per-task learning curves: Combinatorial Optimization & Planning cont. (14/21).

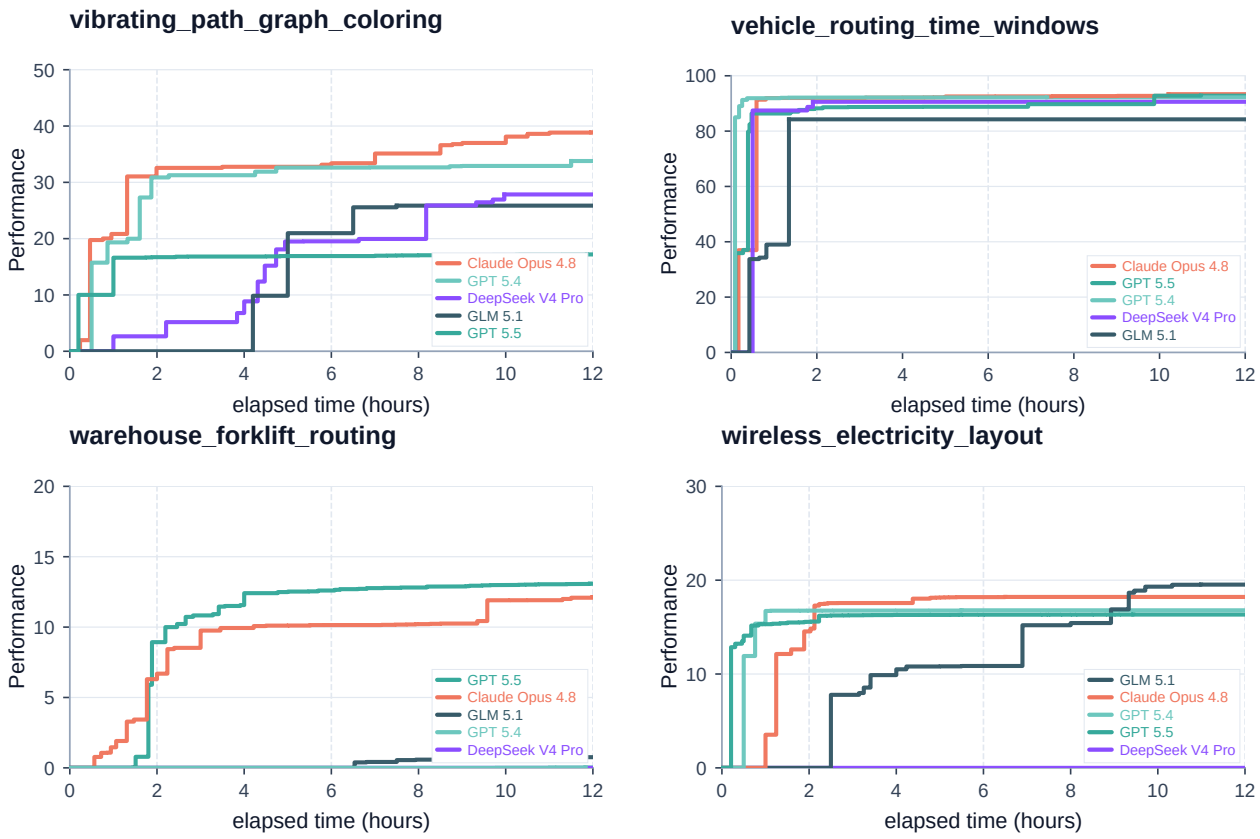


Figure 29 Per-task learning curves: Combinatorial Optimization & Planning cont. (15/21).

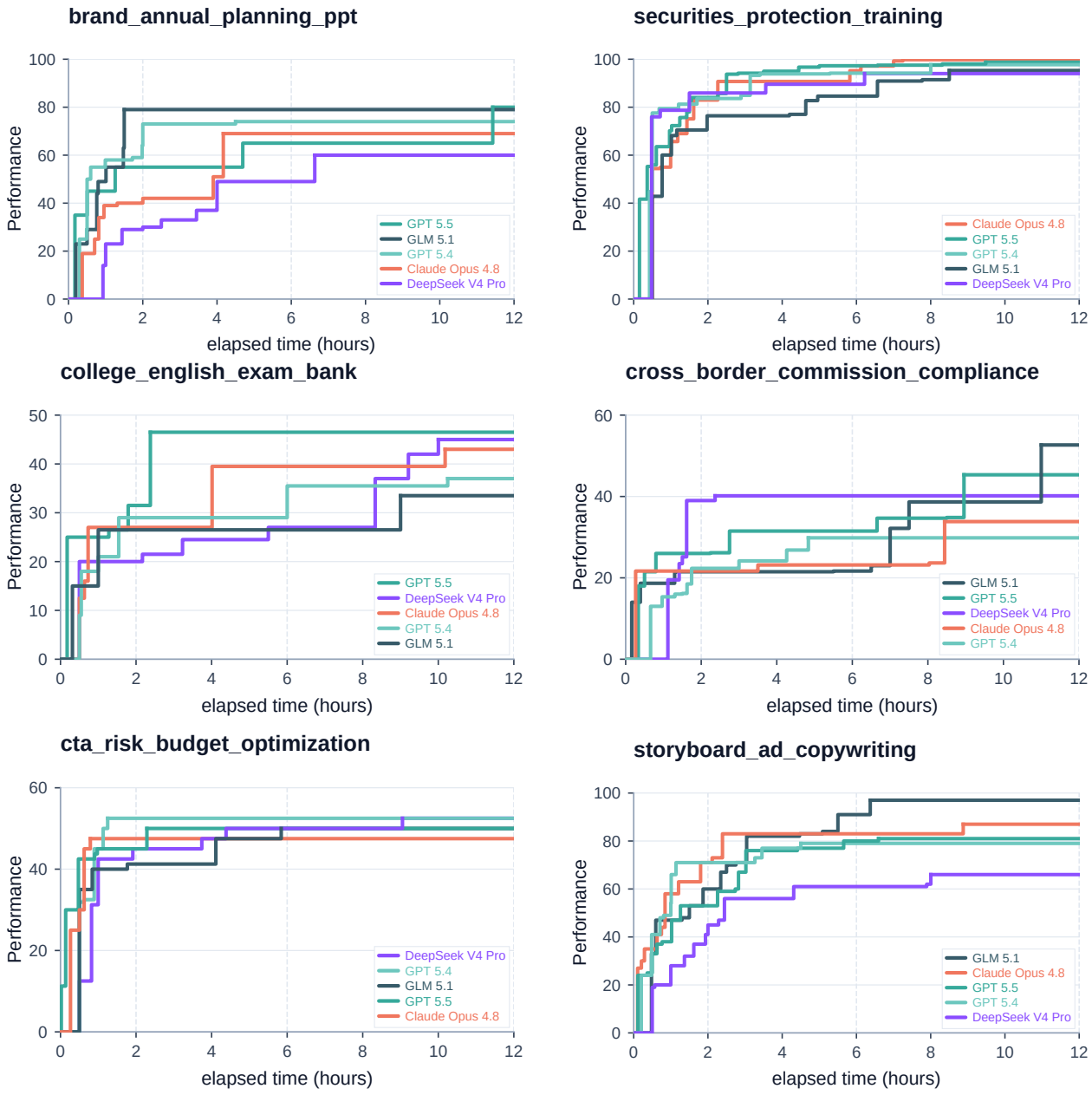


Figure 30 Per-task learning curves: Professional Knowledge Work (16/21).

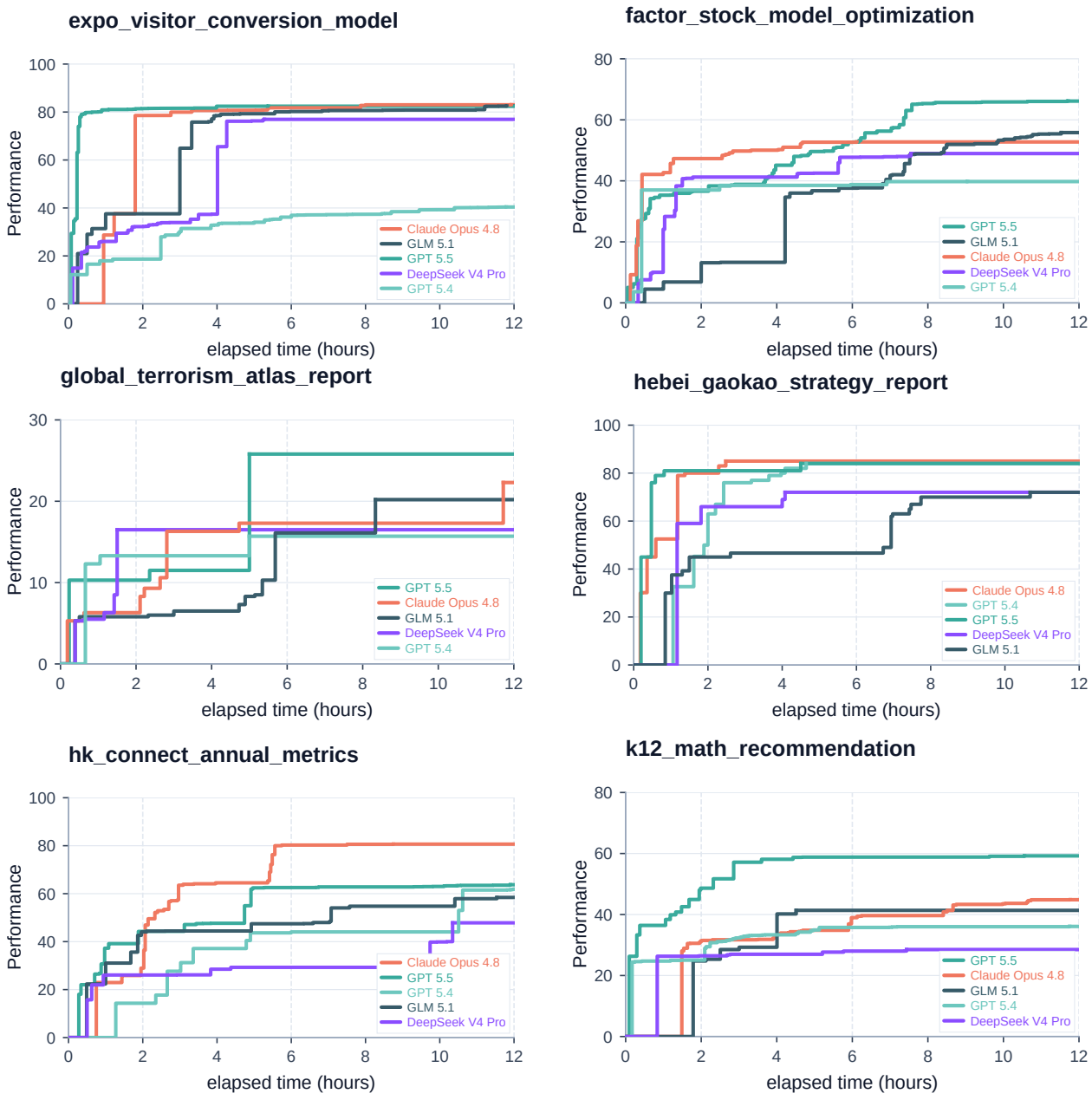


Figure 31 Per-task learning curves: Professional Knowledge Work cont. (17/21).

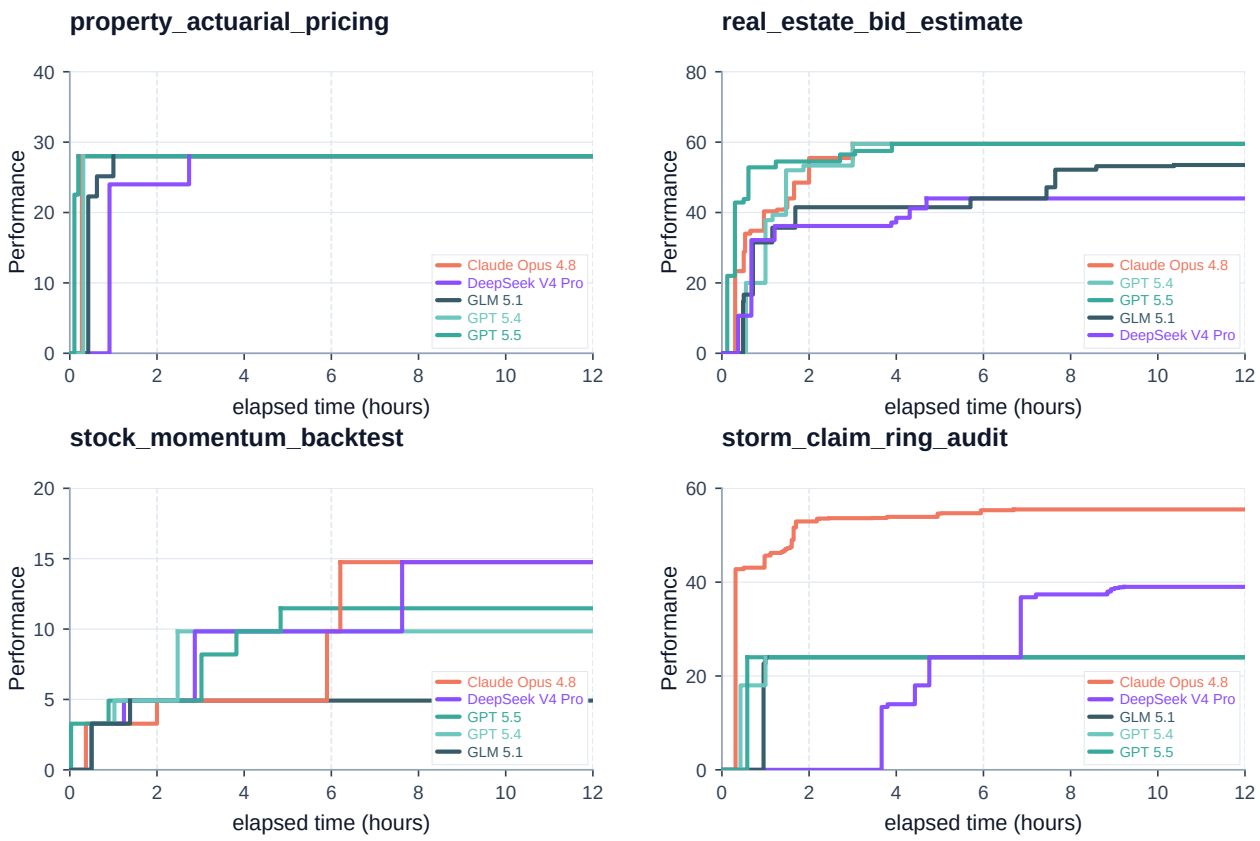


Figure 32 Per-task learning curves: Professional Knowledge Work cont. (18/21).

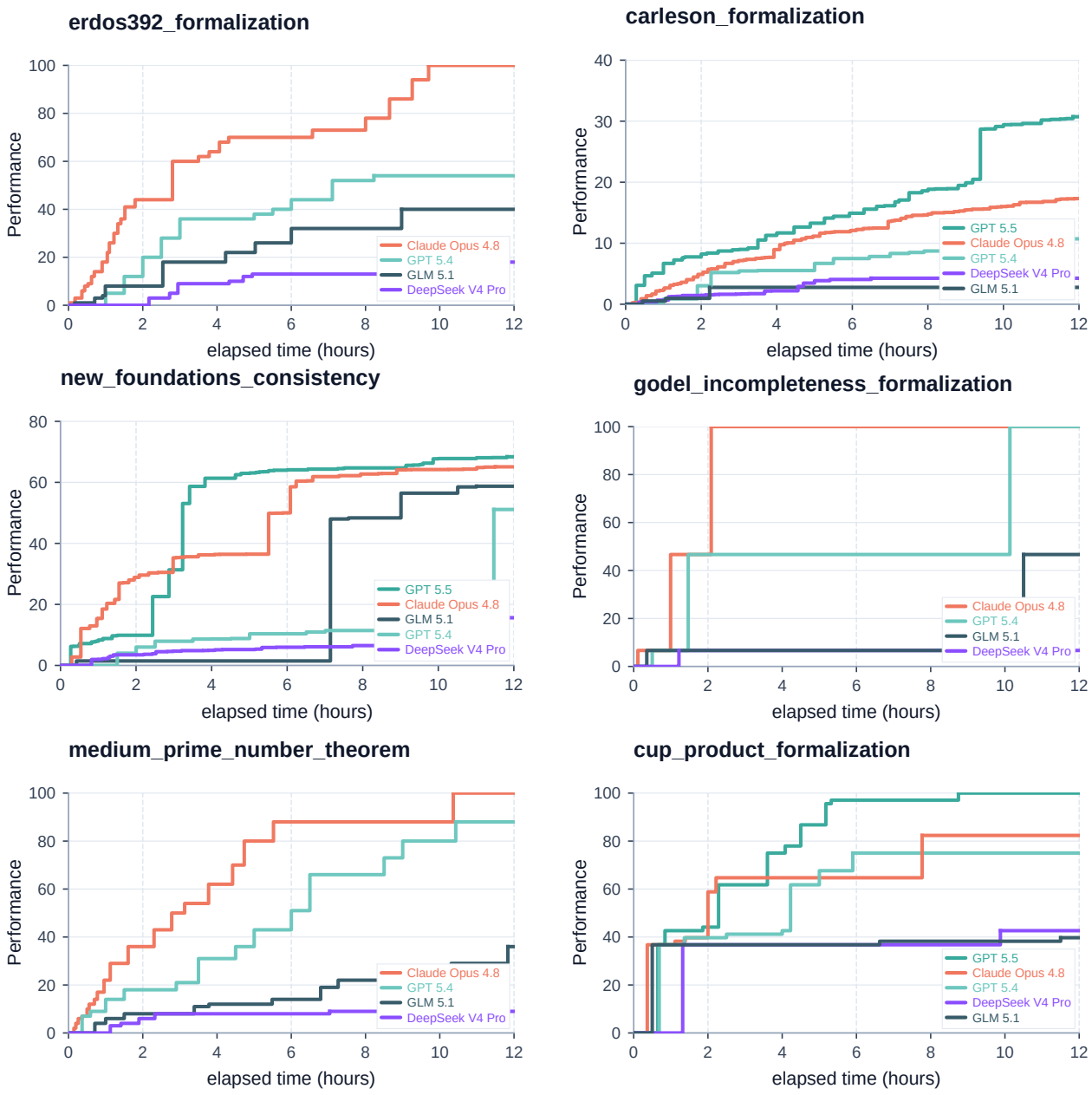


Figure 33 Per-task learning curves: Formal Math & Theorem Proving (19/21).

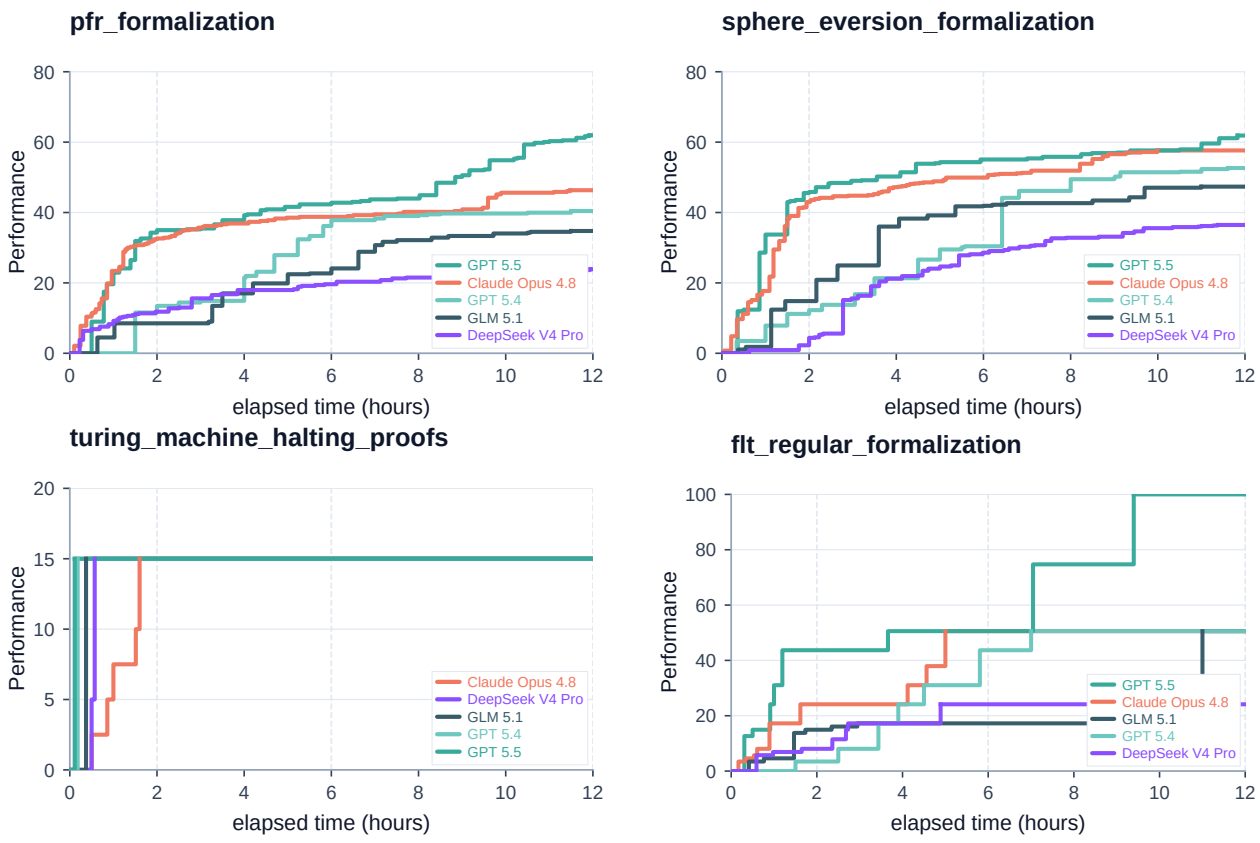


Figure 34 Per-task learning curves: Formal Math & Theorem Proving cont. (20/21).

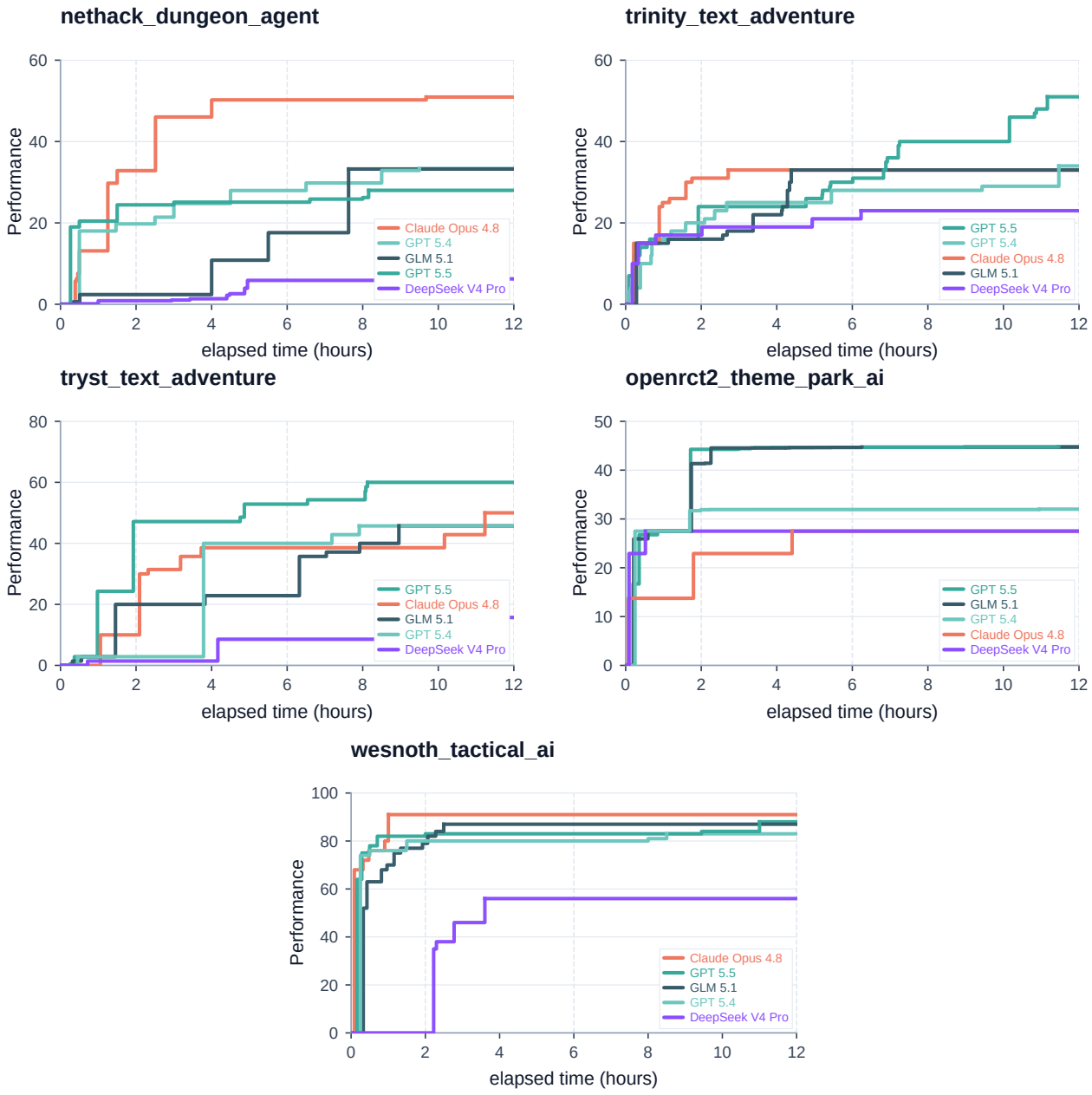


Figure 35 Per-task learning curves: Interactive Games & Simulators (21/21).

G.6 Per-Task Score Tables

For every task–model configuration, we scheduled three independent long-horizon runs. In practice, because these 12-hour trajectories are sensitive to network instability and serving-side reliability limits, we carried out multiple rolling evaluation rounds to recover failed or incomplete runs and produced the final score tables below. A small number of reported task–model cells nevertheless still have fewer than three valid runs; those cells are marked with *. Each run score is first rescaled to the 0–100 task scale; the adjacent $\pm s$ term reports the sample standard deviation across these rescaled run scores when at least two valid runs are available. This standard deviation is not clipped to the 0–100 range, so $\bar{x} \pm s$ should be read as run-to-run variation rather than as a bounded score interval.

Systems & Software Engineering Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
codeflash_repair_performance	57.2 ±21.9	**100.0* ±0.0**	—	—	30.5 ±17.9
ffmpeg_swscale_reimplementation	**21.1 ±7.9**	15.3 ±2.8	13.9 ±3.1	2.2 ±3.9	3.8 ±5.9
git_rewrite_in_zig	23.1 ±2.2	18.4 ±0.4	15.4 ±2.4	**23.5 ±1.9**	17.9 ±2.1
arc_compiler_runtime	52.0 ±0.1	**72.4 ±15.1**	50.0 ±0.7	48.7 ±3.5	44.2 ±4.7
pdf_structured_extraction	36.5 ±4.5	29.8 ±2.1	**36.9* ±1.7**	26.3 ±2.9	9.5 ±2.3
ann_vector_search_qps	**59.7 ±2.1**	40.7 ±18.8	50.2 ±17.4	38.3 ±17.5	23.8 ±3.0
rust_multicrate_reconstruction	—	**57.8 ±16.8**	21.4 ±3.0	38.5 ±21.4	23.6 ±2.4
copier_modular_refactor	**98.9 ±0.0**	97.8 ±1.2	98.1 ±0.6	98.0 ±0.9	87.2 ±9.4
dependent_type_checker	**44.7 ±3.7**	24.7 ±21.4	3.8 ±6.7	1.9 ±3.2	0.0 ±0.0
entt_graph_module	**100.0 ±0.0**	94.3 ±3.0	78.4 ±25.7	**100.0 ±0.0**	92.0 ±10.5
exchange_core_throughput	**59.7 ±2.7**	53.2 ±6.8	47.3 ±10.2	52.6 ±12.9	48.6 ±17.9
integer_compression_codec	**75.3 ±0.3**	74.4 ±0.5	42.3 ±18.4	28.9 ±4.3	16.2 ±8.1
juliet_vulnerability_analyzer	75.6 ±6.7	**89.8 ±1.9**	77.2 ±5.5	63.5 ±4.6	66.2 ±8.4
libexpat_x86_assembly	15.8 ±14.0	13.5 ±11.8	11.9 ±8.4	**18.1 ±24.3**	11.2 ±15.7
litestar_infra_refactor	**92.5***	66.4 ±15.9	60.8 ±7.7	46.0 ±3.7	43.6 ±3.3
lua_native_compiler	**98.9* ±0.8**	78.2 ±27.4	90.7 ±8.6	40.9* ±57.9	41.2* ±7.8
mimesis_modular_refactor	**100.0 ±0.0**	91.0 ±2.7	87.8 ±1.9	82.0 ±6.9	65.4 ±12.1
nlohmann_json_modularization	**100.0 ±0.0**	98.9 ±0.5	87.3 ±11.0	77.8 ±0.0	73.3 ±0.9
notebook_lossless_compression	**53.0 ±2.3**	19.3 ±27.2	7.3 ±2.0	35.1 ±24.3	16.0 ±17.2
packer_plugin_datasources	90.0 ±0.0	90.8* ±1.2	33.3 ±14.4	**91.1 ±1.0**	90.6 ±1.0
pocketbase_tools_extensions	**100.0 ±0.0**	94.8 ±2.6	66.5 ±15.2	57.4 ±13.0	60.0 ±17.3
postgres_wire_on_sqlite	**8.2 ±0.3**	7.7 ±0.6	7.8 ±0.3	8.1* ±0.3	7.9* ±0.2
quic_transport_stack	48.3 ±14.3	**63.6 ±8.9**	47.1 ±16.1	52.5* ±22.4	33.4 ±10.0
regex_automata_repair	66.7* ±0.1	**67.0 ±0.0**	61.0 ±10.3	28.6*	2.3 ±2.2
schemathesis_datagen_pipeline	**70.2 ±2.7**	56.7 ±3.0	56.6 ±3.3	67.0 ±7.0	52.3 ±5.3
schemathesis_reporting_observability	76.2 ±4.7	**77.1 ±3.5**	76.2 ±2.9	61.9 ±1.5	65.0 ±11.7
schemathesis_config_modernization	**87.7 ±2.6**	84.0 ±1.5	71.9 ±1.8	61.7 ±4.2	55.6 ±2.9
stream_processing_engine	**100.0 ±0.0**	**100.0 ±0.0**	**100.0 ±0.0**	**100.0 ±0.0**	**100.0 ±0.0**
tls13_handshake_state_machine	29.4 ±0.9	**39.3 ±1.1**	37.9 ±0.9	29.2 ±1.8	29.0 ±0.6
vault_sdk_resilience	**100.0 ±0.0**	29.1 ±21.0	18.2 ±15.7	13.9 ±8.4	15.2 ±18.9
vliw_kernel_optimization	80.9 ±0.9	**85.6 ±1.9**	79.1 ±1.5	35.9 ±25.1	34.1 ±19.1
cpu_full_flow	72.0* ±7.1	**88.5* ±12.0**	85.3 ±11.7	51.0*	56.3 ±4.7
zstd_api_modernization	**100.0 ±0.0**	95.8 ±7.2	**100.0 ±0.0**	**100.0 ±0.0**	91.7 ±14.4
cfzip_compression_engine	**98.4 ±0.0**	97.4 ±1.3	96.2* ±0.7	87.4 ±12.0	92.5 ±4.8
pocketbase_backend_architecture	0.0 ±0.0	**62.5 ±0.0**	20.8 ±36.1	61.1 ±2.4	4.2 ±7.2

Table 8 Model performance on Systems & Software Engineering tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

Scientific Problems & ML Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
capecod_plume_reconstruction	**19.9 ±11.1**	16.4 ±5.5	12.6 ±4.6	10.9 ±2.5	8.8 ±0.4
battery_soh_rul_anomaly	**37.2 ±10.7**	30.2 ±14.8	14.7 ±1.8	18.2 ±4.3	14.0 ±0.6
borden_source_inversion	**48.4 ±7.3**	38.5 ±14.3	8.0 ±1.4	15.1 ±13.7	38.2 ±3.6
borden_pump_treat_dispatch	14.9 ±2.5	**16.0 ±1.6**	10.7 ±0.6	14.5 ±1.7	13.6 ±2.1
borden_sensor_fault_diagnosis	5.4 ±2.6	**12.2 ±8.0**	3.4 ±0.2	3.4 ±0.3	3.0 ±0.0
bridge_gnss_state_forecast	22.0 ±1.4	21.8 ±2.3	21.0 ±1.1	**23.7 ±1.8**	21.3 ±0.5
vsg_stability_parameter_optimization	27.9 ±16.6	**47.0 ±34.0**	5.8 ±2.9	4.5* ±0.8	4.4 ±0.5
cylinder_wake_prediction	66.6 ±4.3	**69.9 ±4.6**	39.8 ±16.6	36.9* ±36.0	24.2 ±14.6
dabic_gravity_inversion	**17.5 ±3.0**	17.3 ±0.7	15.0* ±0.7	17.1 ±0.7	13.8* ±2.9
noisy_product_matching_pipeline	**68.1 ±6.5**	64.3 ±6.8	27.1 ±24.7	44.7 ±3.4	54.2 ±9.1
neural_net_weight_recovery	94.9 ±8.9	**100.0***	**100.0* ±0.0**	69.2 ±0.0	65.4* ±5.4
nanophotonic_simulation_reproduction	**64.8 ±1.0**	38.2* ±1.8	—	43.1 ±9.5	42.8 ±3.5
ftir_polymer_identification	**45.0 ±14.0**	32.0* ±5.7	19.3 ±16.4	12.3 ±4.0	12.3 ±7.1
molecular_property_regression	**49.5 ±16.9**	43.2 ±5.9	23.3 ±0.8	24.2 ±3.7	27.1 ±7.2
graph_node_classification	**66.6 ±3.0**	56.0 ±15.8	57.6 ±2.0	52.3 ±8.7	51.8 ±8.1
gravitational_wave_signal_detection	61.5 ±8.1	**64.5 ±2.2**	50.0 ±3.7	44.8 ±15.7	58.8 ±2.9
polyimide_homo_lumo_prediction	86.7 ±23.1	**100.0* ±0.0**	**100.0* ±0.0**	60.0* ±0.0	80.0* ±28.3
industrial_anomaly_detection	**49.3 ±5.2**	40.8 ±1.7	20.4 ±19.6	34.4 ±4.6	35.1 ±5.9
bipedalwalker_locomotion_rl	**23.3 ±3.8**	21.0 ±8.5	17.5 ±1.2	22.5 ±2.1	20.6 ±4.3
molecular_solubility_prediction	37.3 ±1.9	**51.7 ±9.9**	35.6 ±5.0	36.8 ±2.3	33.0 ±4.0
motor_clutch_model_reproduction	**100.0 ±0.0**	21.7 ±37.5	—	20.0* ±28.3	20.0 ±34.6
streaming_multilabel_classification	67.2 ±2.0	63.1 ±3.1	**68.5 ±5.2**	52.9 ±6.2	60.4 ±6.5
barnes_hut_nbody_acceleration	78.4 ±6.2	**91.6 ±14.5**	87.4 ±15.4	64.1 ±8.5	67.1 ±2.2
blackbox_numerical_integration	**45.1 ±9.1**	44.3 ±12.9	33.4 ±5.4	40.2 ±5.6	40.2 ±3.4
monge_ampere_pde_solver	**63.3 ±5.3**	37.1 ±8.8	9.6 ±16.5	48.1 ±11.2	32.9 ±30.2
ocean_mt_lab_inversion	41.9 ±47.4	39.7 ±43.7	15.2 ±3.3	**60.8 ±40.2**	14.5 ±0.0
pv_power_forecasting	15.3 ±1.0	**17.2 ±1.0**	12.6 ±0.5	16.6 ±0.5	14.1 ±2.4
collaborative_filtering_recommender	**55.0 ±2.9**	46.4 ±9.3	14.7 ±12.2	39.9 ±20.5	8.5 ±1.8
ecg_signal_processing_pipeline	**58.7 ±7.3**	44.7 ±8.7	39.2 ±7.8	31.8 ±1.9	14.3 ±8.6
sketch_solve_least_squares	59.4 ±0.8	61.1 ±2.3	60.2 ±1.3	61.1 ±0.8	**61.3 ±1.1**
substrate_interface_simulation	**0.0* ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**
thermo_fluid_field_prediction	69.6 ±20.8	57.8 ±15.1	39.1 ±5.3	**76.8 ±17.8**	22.5 ±15.7
csi_time_series_forecasting	**83.4 ±10.4**	76.3 ±2.5	77.3 ±0.2	44.6 ±3.1	40.4 ±12.3
roof_damage_active_learning	4.7 ±8.2	**22.8* ±7.1**	—	4.1 ±7.0	0.0 ±0.0

Table 9 Model performance on Scientific Problems & ML tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

Combinatorial Optimization Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
symbolic_integration_engine	**57.7 ±1.1**	44.0 ±4.2	30.9 ±1.7	26.7 ±8.9	18.4 ±2.9
order_addition_permutation_optimization	**36.4 ±5.7**	23.3 ±1.4	14.3 ±12.4	33.2 ±9.8	30.8 ±11.3
sat_solver	**14.4 ±7.6**	8.9 ±0.6	13.8 ±4.7	13.6 ±5.7	8.0 ±6.4
quantum_architecture_search	12.7 ±1.5	**68.3 ±22.2**	12.5* ±2.1	25.3 ±23.1	12.0 ±1.7
ad_placement_optimization	**67.7 ±1.0**	62.9 ±6.2	48.1 ±6.2	58.8 ±14.3	36.2 ±16.3
first_order_theorem_prover	**31.9 ±13.0**	11.2 ±11.2	13.1 ±22.7	0.0*	0.0 ±0.0
circuit_layout_optimization	**37.3 ±2.4**	33.0 ±3.9	26.0 ±3.9	31.3 ±5.6	22.5 ±3.5
equivalence_class_divide_and_conquer	21.3 ±4.9	**22.4 ±12.2**	20.3 ±2.4	10.6 ±2.0	3.4 ±3.5
jagua_nesting_optimization	**44.2 ±16.7**	21.6 ±10.9	24.1 ±11.0	12.4 ±8.2	28.4 ±16.7
smt_solver	**23.9 ±7.0**	8.6 ±3.1	9.2 ±2.8	3.6*	3.3 ±1.7
tree_block_partitioning	**37.7 ±6.6**	36.4 ±2.8	34.3 ±4.7	23.4 ±4.5	16.1 ±0.9
triangulation_coloring_optimization	73.4 ±2.4	**75.2 ±3.0**	74.3 ±3.3	73.0 ±1.5	59.3 ±10.3
vibrating_path_graph_coloring	**25.3 ±11.7**	11.4 ±5.1	24.1 ±8.5	22.9 ±3.2	22.1 ±5.1
vehicle_routing_time_windows	74.0 ±16.9	**90.8 ±2.3**	89.6 ±2.3	77.9 ±8.1	83.1 ±6.6
warehouse_forklift_routing	11.2 ±1.1	**12.6 ±0.6**	0.0 ±0.0	0.5* ±0.3	0.0* ±0.0
wireless_electricity_layout	**14.5 ±6.1**	7.2 ±7.9	11.1 ±9.6	9.5 ±9.8	0.0 ±0.0

Table 10 Model performance on Combinatorial Optimization tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

Formal Math & Theorem Proving Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
combinatorial_games_formalization	35.5 ±3.9	**38.2 ±10.1**	17.8 ±6.8	16.2* ±3.1	7.8 ±0.9
erdos392_formalization	**98.0 ±3.5**	—	48.0 ±5.3	32.7 ±11.0	10.7 ±6.7
cup_product_formalization	74.0 ±8.9	**93.1 ±6.1**	74.0 ±1.7	38.7 ±0.8	40.7 ±3.4
lean_analysis_proofs	33.0*	**42.5 ±5.1**	16.4 ±4.3	5.9*	9.5 ±2.7
carleson_formalization	16.8 ±0.5	**26.5 ±6.5**	7.1 ±3.2	2.2 ±0.7	2.5 ±1.5
new_foundations_consistency	65.1*	**66.5 ±2.0**	39.8 ±12.2	27.0 ±27.5	11.4 ±4.9
godel_incompleteness_formalization	**100.0 ±0.0**	—	64.4 ±30.8	46.7*	6.7 ±0.0
medium_prime_number_theorem	**100.0* ±0.0**	—	88.0 ±0.0	26.7 ±8.1	9.0 ±0.0
ordinal_notation_well_foundedness	**24.7 ±0.0**	**24.7 ±0.0**	21.6 ±5.4	5.9 ±0.0	4.7* ±1.7
pfr_formalization	46.3*	**60.0 ±2.7**	38.9 ±1.3	33.5 ±1.4	19.1 ±5.4
sphere_eversion_formalization	55.4 ±3.7	**58.5 ±2.9**	51.4 ±1.7	30.2 ±21.3	29.3 ±7.7
turing_machine_halting_proofs	**15.0 ±0.0**	**15.0 ±0.0**	**15.0 ±0.0**	**15.0 ±0.0**	**15.0 ±0.0**
flt_regular_formalization	50.6 ±0.0	**75.1 ±24.7**	48.3 ±4.0	38.7 ±13.4	17.6 ±11.3

Table 11 Model performance on Formal Math & Theorem Proving tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

Professional Knowledge Work Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
portfolio_risk_calibration	24.5 ±7.5	**25.0 ±6.5**	10.7 ±3.9	9.4 ±14.2	23.7 ±6.9
storyboard_ad_copywriting	79.7 ±8.1	77.0 ±5.3	65.7 ±14.6	**92.0 ±5.0**	56.5* ±7.8
cross_border_investment_ppt	38.3 ±4.7	34.0* ±2.2	35.7 ±6.5	**45.9 ±7.0**	32.3 ±5.9
herbal_depression_target_screening	**70.3 ±5.9**	68.1 ±3.1	58.9 ±7.9	57.0 ±12.8	60.1 ±3.3
pancreatic_radiotherapy_meta_analysis	**40.2 ±5.7**	39.8 ±3.2	30.6 ±4.1	36.5 ±5.2	34.5 ±7.2
touchstone_vna_diagnostics	**39.9 ±2.8**	11.5 ±4.9	8.4 ±1.1	4.7 ±1.8	4.5 ±2.4
high_performance_object_mapper	—	62.2 ±2.9	45.6 ±8.9	**64.0 ±1.3**	51.0 ±7.2
odata_query_service	30.9 ±1.1	27.8 ±3.3	**31.1 ±4.9**	26.1 ±3.2	20.8 ±11.0
brand_annual_planning_ppt	51.3 ±19.8	69.0 ±11.0	72.3 ±2.1	**73.7 ±9.2**	50.0 ±10.0
securities_protection_training	**97.2 ±4.3**	89.4 ±8.2	94.2 ±5.2	93.8 ±1.5	85.0 ±9.5
college_english_exam_bank	**39.8 ±4.3**	37.8 ±7.8	34.5 ±2.3	32.5 ±1.3	34.7 ±10.3
cross_border_commission_compliance	32.1 ±2.4	**43.8* ±2.1**	29.8*	39.2 ±12.3	35.2 ±5.4
cta_risk_budget_optimization	46.1 ±1.4	46.7 ±3.8	**49.8 ±2.8**	49.6 ±0.7	48.1 ±4.1
equity_objection_report	15.5 ±4.3	**22.5 ±2.3**	15.3 ±6.5	14.7 ±3.8	14.8 ±11.2
expo_visitor_conversion_model	**81.9 ±1.6**	54.9 ±23.9	29.1 ±10.0	79.0 ±3.8	46.2 ±26.7
factor_stock_model_optimization	51.0 ±1.5	**58.8 ±6.9**	36.3 ±3.2	54.3 ±1.5	35.6 ±11.9
global_terrorism_atlas_report	19.4 ±2.6	**21.5 ±5.5**	15.7*	17.3 ±4.4	15.2 ±1.7
hebei_gaokao_strategy_report	69.9 ±16.3	**82.0* ±2.8**	78.7 ±4.7	71.0 ±1.7	62.5* ±13.5
hk_connect_annual_metrics	**73.4 ±11.5**	61.8 ±3.2	45.2 ±14.4	44.8 ±18.4	41.5 ±10.9
k12_math_recommendation	**44.3 ±0.5**	44.0 ±13.3	31.4 ±4.2	32.7 ±8.2	26.3 ±2.2
property_actuarial_pricing	**28.0* ±0.0**	**28.0* ±0.0**	**28.0 ±0.0**	**28.0 ±0.0**	**28.0 ±0.0**
real_estate_bid_estimate	52.5 ±6.1	**57.3 ±1.9**	51.5 ±7.0	49.6 ±3.9	40.6 ±3.0
stock_momentum_backtest	9.8 ±4.9	10.7* ±1.2	8.2 ±2.8	4.1* ±1.2	**11.5 ±2.8**
storm_claim_ring_audit	**43.9 ±14.3**	24.0 ±0.0	24.0 ±0.0	24.0 ±0.0	30.3 ±7.8

Table 12 Model performance on Professional Knowledge Work tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

Interactive Games & Simulators Tasks	Opus 4.8	GPT-5.5	GPT-5.4	GLM-5.1	DS-V4-Pro
openttd_transport_ai	**52.0 ±8.5**	28.1 ±24.4	11.9 ±20.5	0.0 ±0.0	15.2 ±26.4
nethack_dungeon_agent	**41.9 ±9.1**	22.5 ±4.8	20.4 ±11.2	21.6 ±16.2	3.3 ±2.3
treant_forest	**18.0 ±4.9**	15.6 ±6.7	13.3 ±7.9	16.9 ±2.4	13.5 ±4.8
grid_turing_robot	40.3 ±3.6	**42.2 ±2.7**	28.9 ±11.3	25.7 ±5.9	24.2 ±6.0
molecular_self_assembly	**34.7 ±0.9**	20.7 ±1.0	21.6 ±0.6	13.2 ±5.9	21.9 ±3.5
apple_incremental_game	**50.6 ±8.1**	33.6 ±15.9	34.9 ±14.7	19.1 ±1.1	19.7 ±0.1
dcss_dungeon_ai	8.3 ±3.5	**13.4 ±0.5**	6.1 ±4.7	7.6 ±4.3	5.7 ±3.7
anchorhead_text_adventure	22.3 ±4.5	**36.3 ±6.4**	17.7 ±2.3	20.3 ±0.6	14.7 ±8.1
trinity_text_adventure	30.0 ±2.6	**40.0 ±10.5**	27.0 ±7.0	26.7 ±5.7	20.3 ±3.8
tryst_text_adventure	44.3 ±8.7	**55.7 ±3.8**	44.3 ±7.6	43.3 ±4.4	13.8 ±3.3
openrct2_theme_park_ai	27.5 ±0.0	**37.6 ±9.0**	23.1 ±11.7	36.2 ±9.5	26.0 ±2.6
wesnoth_tactical_ai	**88.0 ±2.6**	79.3 ±6.4	81.3 ±1.5	78.3 ±11.0	36.3 ±31.5

Table 13 Model performance on Interactive Games & Simulators tasks. Values are mean scores over up to three valid runs; adjacent entries show $\pm s$ when at least two valid runs are available. Bold marks the best model for each task, underlining marks the second-best model, — indicates no valid result, and * marks fewer than three valid runs.

H Acknowledgements

We thank UniPat for collaborating with us on the development of 13 tasks in the Systems & Software Engineering portion of EdgeBench. We also thank Xiaoxing Wu, Xiang Gao, Gao Liu, Yue Yang, Wen Heng, Weinan Zhao, Mailun Gao, Zongbao Zhang, and Yuchen Wu for helpful supporting contributions during the project. We thank Xinkai Zhou and Qi Zhao for meaningful discussions during the project. We thank Tenglong Ao, Ao Zhang, Shengjie Luo, Zeyi Zhang, Guhao Feng, Tianle Cai, Xinrong Zhang, Yizhong Wang, and Ruinian Chang for meaningful discussions and collaborations that took place prior to the EdgeBench project.