SWE-SERVE: BENCHMARKING AGENTIC ENGINEERING FOR PRODUCTION INFERENCE SERVING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce SWE-Serve, a benchmark for evaluating agents on production inference engineering tasks. Implementing an inference feature can require coordinating multiple changes across the serving stack, including model support, runtime execution, and public APIs. Existing benchmarks provide limited coverage of production inference engineering: repository-level software engineering benchmarks do not target inference, while general terminal-agent benchmarks include only a few inference tasks. Dedicated inference benchmarks, meanwhile, focus primarily on isolated kernel generation or performance optimization rather than repository-scale production feature implementation. SWE-Serve provides 53 repository-grounded tasks derived from recent production changes to SGLang, spanning six inference engineering families. Each task executes on either CPU or a single GPU (H100) and is evaluated with hidden functional and regression tests, including, where applicable, end-to-end (E2E) serving tests and calibrated performance gates. Executable no-op and oracle controls, adversarial verifier review, and closed-book execution support task validity and evaluation integrity. Across 11 models and 31 model–effort configurations, the best-performing configuration achieves 75% mean pass@1. SWE-Serve exposes a substantial gap between completing tasks locally and achieving production correctness. On 19 tasks with end-to-end coverage, model-serving E2E tests reject roughly one-third of patches that pass every other test (45.9% under the verifier versus 69.4% with E2E tests excluded from scoring), with pass rate increasing for each model's best-performing configuration. By making the production correctness gap directly measurable, SWE-Serve enables the field to track whether future agents move beyond completing tasks locally to achieving production correctness.

1 INTRODUCTION

Repository-scale software engineering has become a common setting for coding-agent evaluation (Jimenez et al., 2024; OpenAI, 2024; Deng et al., 2026; Huang et al., 2026). Recent benchmarks have begun to extend this setting toward feature-scale and longer-horizon work (Zhou et al., 2026; Feng et al., 2026; Luo et al., 2026; Yang et al., 2026; Le et al., 2026). However, repository-scale evaluation does not necessarily test whether a locally working implementation remains correct across the production path. We introduce SWE-Serve, which evaluates whether agents achieve production correctness for inference engineering tasks.

In production inference systems, a change can work in isolation yet fail when exercised through the complete system. On the Gemma 4 mixture-of-experts (MoE) core-serving task, for example, 48.5% of the agent-created patches evaluated for the primary leaderboard (16 of 33) passed every test other than the end-to-end (E2E) tests but failed to serve the specified model correctly through the live server's public interfaces (Appendix Table 10). Other SWE-Serve tasks test whether patches remain correct when requests share persistent state or execute concurrently. These failures affect users despite apparent local correctness, motivating us to define and measure *production correctness*: satisfying a task's behavioral requirements within the inference system, including preserving existing functionality and meeting any task-specific performance constraints.

SWE-Serve is a benchmark for production inference engineering built from changes merged since December 2025 into SGLang, an open-source inference-serving system (Zheng et al., 2024). There are

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

53 production-grounded tasks spanning six families of inference engineering, from model support and decoding to distributed execution and serving APIs. Tasks focus on concrete engineering objectives, derived either from individual pull requests (PRs) or from combinations of related PRs when the work spans PR boundaries. Each task consists of a containerized environment with the repository checked out at the task's designated base commit, a task instruction, hidden executable tests, and an oracle solution that establishes task feasibility. Each task executes in either a CPU-only environment or on a single H100 GPU. We qualify each task by checking that its hidden tests reject the unchanged repository (the no-op control) and accept the oracle solution, followed by agent-assisted adversarial probing and human adjudication. Additionally, evaluation integrity is protected with closed-book execution and trajectory auditing.

SWE-Serve provides an executable, production-grounded evaluation of whether agents carry locally working implementations through a complete production path. We release the benchmark, execution environments, and verifiers to support reproducible evaluation. Our contributions are:

- **A production inference engineering benchmark:** 53 production-grounded tasks across six engineering families, with CPU/H100 execution and hidden functional and regression tests.
- **A validity and integrity methodology** combining production-grounded construction, executable no-op and oracle controls, agent-assisted adversarial probing, human adjudication, closed-book execution, and trajectory auditing.
- **Evidence of a production correctness gap.** Removing model-serving end-to-end (E2E) tests from scoring while leaving the agent-created patches and every other test unchanged raises pass rate by 23.4 percentage points, with an increase in all 11 top-per-model configurations. Among the 12 tasks that support a matched comparison, removing E2E tests produces 2.0× as many newly passing patches as removing the same number and type of non-E2E tests.

2 Related Work

Existing benchmarks cover only narrow slices of production inference engineering. SWE-Serve builds upon related work spanning repository-level agentic benchmarks, inference-performance benchmarks, task construction across related repository changes, and closed-book evaluation (Table 1).

Agentic benchmarks. Repository-level software engineering benchmarks, such as SWE-bench Verified, SWE-bench Pro, and DeepSWE, have established executable evaluation of agents on changes to existing software repositories (Jimenez et al., 2024; OpenAI, 2024; Deng et al., 2026; Huang et al., 2026). However, they do not focus on inference engineering or GPU-dependent execution. Terminal-Bench 3 broadens agentic evaluation to interactive systems tasks and introduces a few inference tasks (3 of 74 tasks; 4.1%), but they remain a minor component of the benchmark (Marten et al., 2026). Together, these benchmarks provide limited coverage of the breadth of inference engineering.

Inference-performance benchmarks. Existing inference-performance benchmarks target several distinct layers of the inference stack. KernelBench and SOL-ExecBench evaluate GPU-kernel generation and optimization, while InferenceBench targets deployed-system inference optimization (Ouyang et al., 2025; Lin et al., 2026; Yeon et al., 2026). ISO-Bench is the only benchmark in this group that evaluates inference optimization at the repository level; its tasks are drawn exclusively from performance-optimization changes in vLLM and SGLang (Nangia et al., 2026). However, inference engineering encompasses a broader range of tasks than performance alone.

Construction across PRs. Pull requests reflect repository-specific development practices rather than a standardized unit of functionality: a coherent engineering objective may be contained in one PR or distributed across several. FeatureBench and SWE-EVO demonstrate task constructions that group related repository changes rather than requiring every task to follow a single-PR boundary (Zhou et al., 2026; Le et al., 2026). SWE-Serve brings this flexibility to production inference engineering, scoping one or more related upstream changes into bounded, executable tasks. This permits work that spans PR boundaries to be evaluated together.

Closed-book evaluation. Closed-book execution, where agents cannot retrieve upstream solutions from the public web, remains uncommon among agentic benchmarks. DeepSWE and SWE-EVO are notable exceptions (Huang et al., 2026; Datacurve, 2026; Le et al., 2026). SWE-Serve similarly evaluates agents closed book and further audits every evaluation trajectory for prohibited access.

Table 1: Comparison of SWE-Serve with related benchmarks. Checkmarks indicate documented coverage. Labels denote: "Beyond performance": inference-engineering tasks beyond optimization; "Closed-book": execution in which agents cannot retrieve upstream solutions; and "Multi-PR": tasks constructed from multiple PRs. Only 3 of 74 Terminal-Bench 3 tasks concern inference engineering.

	Inference coverage			Task construction		Execution setting		
Benchmark	Inference focused	Perf. metrics	Beyond perf.	Repo-grounded	Multi-PR	GPU task	CPU-only task	Closed-book
General SWE								
SWE-bench Verified				✓			✓	
SWE-bench Pro				✓			✓	
DeepSWE				✓			✓	✓
Terminal agents								
Terminal-Bench 3		✓	✓	✓		✓	✓	
Inference performance								
KernelBench	✓	✓				✓		
SOL-ExecBench	✓	✓				✓		
ISO-Bench	✓	✓		✓		✓		
InferenceBench	✓	✓				✓		
Production inference								
SWE-Serve	✓	✓	✓	✓	✓	✓	✓	✓

3 THE SWE-SERVE BENCHMARK

SWE-Serve turns recent engineering work on production inference systems into 53 repository-level software engineering tasks with executable behavioral verification. We build the benchmark from merged changes to SGLang (Zheng et al., 2024), an open-source inference-serving system. As Figure 1 illustrates, each task gives an agent a task instruction and the SGLang codebase at the task's designated base commit. The agent creates a patch; hidden executable tests then evaluate the patch for required behavior, regression protection, and applicable end-to-end or performance requirements. This section defines the tasks, characterizes the resulting benchmark, and presents the construction and qualification evidence supporting its validity.

3.1 TASK FORMULATION AND SCORING

Each SWE-Serve task includes an instruction, a sandboxed execution environment specification, an oracle solution, and a hidden executable verifier. We package and execute these tasks using Harbor, an open-source agent-evaluation framework from the creators of Terminal-Bench (Harbor Framework Team, 2026; Merrill et al., 2026). During execution, the agent may inspect, edit, build, and test the repository, including its existing tests. It does not receive the task's oracle solution, source-change identifiers, or the hidden verifier.

After the agent's work is completed, the hidden verifier evaluates the resulting agent-created patch. The agent-created patch is not directly compared with the oracle patch, which is used only to qualify that the task contract is achievable. Instead, following a standard established in SWE-bench, the verifier assesses behavior through executable tests: fail-to-pass (F2P) tests require behavior absent at the task's designated base commit, while pass-to-pass (P2P) tests protect relevant behavior already present there (Jimenez et al., 2024). When required by the task, the verifier additionally evaluates model-serving end-to-end behavior or enforces calibrated performance gates. The task's complete set of tests defines its *production correctness* requirements, and an agent passes only if every test passes.

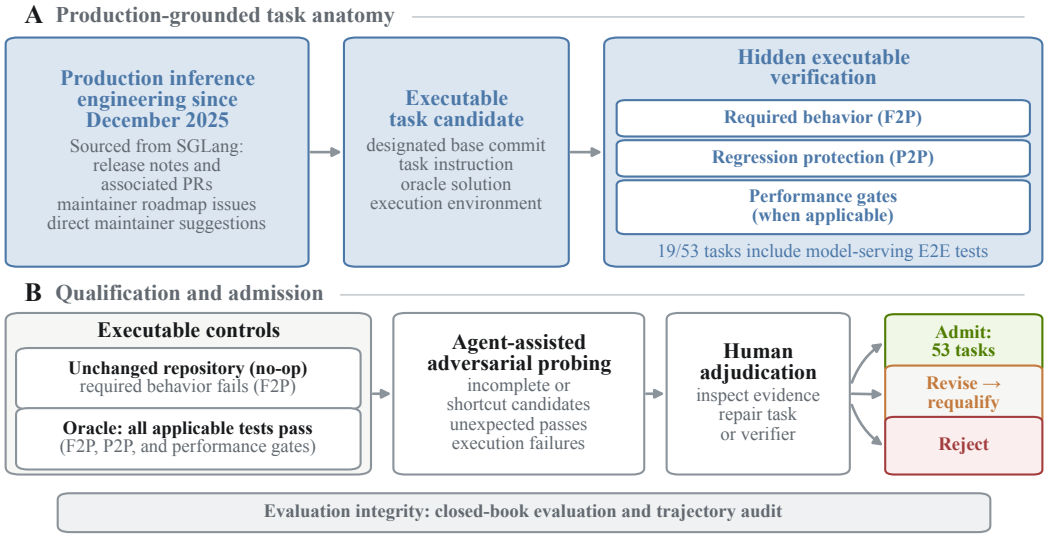


Figure 1: **SWE-Serve task construction and qualification.** Production grounding, executable verification, and qualification jointly support task validity. **(A)** Recent production inference engineering work from SGLang is mapped to executable task candidates with hidden behavioral verification. **(B)** Task admission is informed by no-op and oracle controls, agent-assisted adversarial probing, and human adjudication.

Any implementation that passes those tests is accepted. An implementation that fails any F2P or P2P test therefore does not satisfy the task's production correctness requirements.

3.2 Composition and Scope

Table 2 summarizes engineering breadth and task scale. To give a compact view of the engineering work represented, we organize the tasks into six mutually exclusive engineering families (Table 2A). The tasks also vary in construction scope: 37 derive from one upstream change, while 16 integrate two to six related changes. In total, the benchmark draws on 83 unique merged upstream pull requests. Execution spans 12 CPU and 41 GPU (H100) tasks, with strict model-serving E2E tests in 19 tasks and calibrated performance gates in 3. Together, the benchmark spans localized correctness fixes and integrations that cross API, scheduling, model-execution, and runtime-state boundaries.

Table 2: SWE-Serve benchmark composition and scale.

A. Engineering coverage

Engineering family	Tasks
Model and backend enablement	12
Speculative and advanced decoding	14
Kernels, quantization, and performance	8
Caching and runtime state	7
Distributed execution and scheduling	4
Serving APIs and runtime correctness	8
Total	53

B. Task and verification scale

Attribute	Median	Max.
Task-instruction words	200	915
Oracle lines changed	553	6,077
Oracle files changed	7	35
Fail-to-pass (F2P) tests	7	246
Pass-to-pass (P2P) tests	10	2,034

Table 2B reports oracle-solution size for the tasks: the median oracle solution changes 553 lines across seven files, with maxima of 6,077 lines and 35 files. The full distributions of oracle lines and files changed reveal smaller patches not apparent from the summary statistics (Appendix Figure 4).

SWE-Serve's oracle solutions change more lines and files on average than those of SWE-bench Verified, SWE-bench Pro, and DeepSWE, while its mean prompt is shorter than those of DeepSWE and SWE-bench Pro (Appendix Figure 5; Huang et al., 2026). The oracle comparison suggests broader repository-level implementation scope, while the shorter prompts leave agents to determine how to implement that work rather than follow prescriptive plans.

Before examining any model outcomes, we used only the task instruction and tests in the executable verifier to assign each task three labels: runtime-domain breadth, persistent state, and concurrent coordination. Runtime-domain breadth indicates how many of four domains the verifier tests: request I/O, scheduling and request lifecycle, model execution, and KV-cache and resource management. These domains were derived from recurring runtime responsibilities in four prominent open-source inference systems: vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), TensorRT-LLM (NVIDIA, 2026a), and Triton (NVIDIA, 2026b), as documented in Appendix Table 4. Persistent state indicates that the verifier tests state maintained across operations or lifecycle phases. Concurrent coordination indicates that the verifier tests coordination between overlapping operations. Appendix Table 5 reports these labels alongside each task's family, hardware, and other verifier attributes.

3.3 Construction and Qualification

Construction had three stages: first, five discovery paths identified and screened 786 task-source records, retaining 203 source candidates; next, task construction produced 156 task candidates; finally, qualification admitted 53 tasks (Appendix Figure 6). The discovery paths used four sources: two release-note scans, a manual review of release materials and associated pull requests, maintainer-roadmap issues, and direct SGLang-team suggestions. Next, task construction deduplicated the 203 source candidates and created task candidates. Each task candidate comprised a task instruction, a designated base commit, a sandboxed execution environment specification, an oracle solution, and hidden tests. For multi-PR candidates, the designated base commit precedes the first source change.

At the final stage, 53 of the 156 task candidates were admitted to SWE-Serve, a 34.0% admission rate. Admission required each candidate to run in its specified hardware environment and satisfy the following executable controls: the no-op had to fail every F2P and pass every P2P test, while the oracle had to pass all tests; for performance tasks, these controls were repeated across H100 nodes to account for timing variability. Alongside these executable controls, every admitted task underwent agent-assisted adversarial probing using agent-created patches and trajectories to identify vulnerabilities in its task instruction and verifier (see Appendix Table 6 for validation coverage). This probing looked for valid alternatives rejected by the verifier, incomplete or shortcut solutions that it accepted, unexercised requirements, and environment-induced execution failures, paralleling Terminal-Bench 3's cheating-agent runs and instruction–verifier alignment review (Marten et al., 2026). Executable controls, agent-assisted adversarial probing, and human adjudication jointly informed task revision and final-roster selection. Across this process, 17 broad task candidates were replaced by narrower tasks, and 27 task candidates required verifier changes (Appendix Section A.1.5). Upon admission, a benchmark canary (a fixed string that supports future detection of exposure to released SWE-Serve artifacts) was added to each released task configuration, following BIG-bench and Terminal-Bench (Srivastava et al., 2023; Merrill et al., 2026); Appendix A.1.6 describes the canary's scope and limitations. Together, these controls establish that each task is nontrivial at its base commit, oracle-solvable, reproducible on declared hardware, and scored against its production correctness requirements.

4 Experimental Setup

All experiments use Harbor 0.13.1 (Harbor Framework Team, 2026) to run the full 53-task benchmark in sandboxes with CPU or GPU (H100) resources. The agent develops a solution until it finishes or reaches 350 steps or 210 minutes, then the hidden verifier scores its patch; individual commands time out after 120 seconds without ending the attempt. The 210-minute limit accommodates SWE-Serve's long-running tasks: across nine effort-matched configurations, mean task wall-clock time is 43.3 vs. 34.3 minutes on DeepSWE, and 24.5% vs. 3.5% of task means fall in the one-to-four-hour bucket (Appendix Figure 7). Furthermore, only 2.4% (42/1,749) of attempts across the 11 top-per-model configurations ended as execution-limit failures: 15 at 210 minutes and 27 at 350 steps (Appendix Figure 8). We rerun a task only when the evaluation fails to produce a valid score because of an

issue with the model-serving route, runner, hardware, or verifier infrastructure. We do not rerun tasks because of agent errors, budget exhaustion, test failures, or crashes caused by the agent-created patch.

We compare models using mini-SWE-agent v2.4.3, a model-agnostic harness developed by the team behind SWE-bench and SWE-agent (Jimenez et al., 2024; Yang et al., 2024). Within this harness, models use only Bash to inspect, edit, and test the repository. We then evaluate 11 models, including frontier models: Claude Opus 5 and Sonnet 5 (Anthropic, 2026b;c); GPT-5.6 Sol, Luna, and Terra (OpenAI, 2026b); Kimi K3 (Kimi Team et al., 2026); DeepSeek V4 Flash 0731 (DeepSeek-AI et al., 2026; DeepSeek-AI, 2026); GLM-5.2 (GLM-5-Team et al., 2026; Z.ai, 2026); Gemini 3.6 Flash (Google DeepMind, 2026); Laguna S 2.1 (Poolside, 2026); and Inkling S (Thinking Machines Lab, 2026). Appendix Table 7 lists serving, token-limit, and harness configurations. To assess reasoning-effort sensitivity, we evaluate two Claude and three GPT-5.6 models at `low`, `medium`, `high`, `xhigh`, and `max`: 31 model–effort configurations, each evaluated three times ($K = 3$).

The primary leaderboard highlights each model's configuration with the highest mean pass@1, the average of three run-level pass rates (Section 3.1). Appendix A.2.4 details pass@1 aggregation. We also measure pass@3, the task fraction solved at least once, and passˆ3, the fraction solved in all three attempts (Yao et al., 2025). Furthermore, to assess harness sensitivity, we compare two leading model–effort configurations under mini-SWE-agent and their native harnesses (Section 5.3).

We enforce a closed-book evaluation because an open-book pilot confirmed task-specific upstream retrieval by all three models evaluated (Appendix Table 8). This setting blocks public web and upstream repositories. Only Hugging Face remains allowlisted because serving tasks require model artifacts; task source repositories stay blocked, and model communication remains separate. To assess closed-book integrity, we audit every trajectory for prohibited retrieval. Every prohibited retrieval attempt was blocked, so no trial was invalidated on this basis.

5 Results

5.1 SWE-Serve Reveals a 40-Point Pass@1 Spread Across Evaluated Models

Across 11 models, mean pass@1 under each model's best-performing configuration spans a 40-percentage-point range, from 75% for Claude Opus 5 and GPT-5.6 Sol to 35% for Inkling S. Pass@1, however, captures only one dimension of performance: among four configurations tied at 64% pass@1, mean per-task cost varies by 7.6× ($0.95–$7.24) and wall-clock time by 3.9× (25.5–99.9 minutes), while output tokens range from 61k to 98k and agent steps from 82 to 148. Thus, even among agents with identical pass@1 under a shared harness, selection depends on how cost, token use, steps, and latency are prioritized (Table 3).

To assess reasoning-effort trade-offs, we evaluated each of five models spanning the Claude and GPT-5.6 families at five effort settings, bringing the full evaluation to 31 model–effort configurations (Appendix Table 9). These effort sweeps show that resource trade-offs also arise within models as effort changes. Across the five models, higher effort consistently increases cost, while performance gains vary substantially across models and effort settings (Appendix Figure 9). For Claude Opus 5, moving from `high` to `max` effort raises pass@1 only modestly, from 74% to 75%, while increasing mean cost by 76% ($9.89 to $17.40 per task). This example shows that optimizing for pass@1 alone can incur substantially higher resource costs for only a marginal performance gain. Appendix Figures 10–12 plot pass@1 against cost, output-token use, and wall-clock time and show the observed Pareto frontier for each resource.

Pass@1 can also obscure differences in repeated-run reliability. For example, DeepSeek V4 Flash at `max` effort and GPT-5.6 Luna at `xhigh` effort both achieve 55% pass@1, yet DeepSeek has higher coverage across retries (74% versus 64% pass@3), while Luna has higher consistency (45% versus 42% passˆ3). Table 3 reports consistency, while Appendix Table 9 reports both metrics for all 31 model–effort configurations.

Family-level results provide a complementary view of performance across the benchmark: pooled pass@1 across the 11 best-performing model configurations ranges from 42.9% for model and backend enablement to 74.6% for kernels, quantization, and performance (Appendix Figure 13). Taken together, SWE-Serve separates models across a 40-point pass@1 range while revealing differences in resource use and repeated-run reliability among configurations with similar pass@1.

Table 3: Primary leaderboard. For 11 models, we report the effort configuration with the highest observed mean pass@1 with mini-SWE-agent; rows are ordered by mean pass@1. The pass@1 entry reports the mean and 95% confidence interval across three runs. The pass^3 entry is the fraction of tasks solved in all three runs. The final four columns report means over task attempts.

				Mean per task			
Model	Reasoning effort	pass@1 95% CI	pass^3	Cost	Output tokens	Agent steps	Wall-clock time (min)
Claude Opus 5	max	$75\% \pm 4\%$	70%	$17.40	122k	134	57.5
GPT-5.6 Sol	max	$75\% \pm 6\%$	68%	$12.26	52k	63	29.5
Claude Sonnet 5	xhigh	$64\% \pm 4\%$	51%	$6.61	98k	148	40.6
Kimi K3	max	$64\% \pm 6\%$	51%	$7.24	79k	117	99.9
GPT-5.6 Luna	max	$64\% \pm 4\%$	51%	$0.95	65k	110	28.9
GPT-5.6 Terra	max	$64\% \pm 4\%$	55%	$5.06	61k	82	25.5
DeepSeek V4 Flash (0731)	max	$55\% \pm 5\%$	42%	$0.69	99k	173	36.4
GLM-5.2	max	$48\% \pm 2\%$	34%	$2.10	53k	101	34.0
Gemini 3.6 Flash	high	$48\% \pm 7\%$	30%	$4.84	99k	92	37.3
Laguna S 2.1	max	$46\% \pm 5\%$	26%	$0.33	147k	184	56.9
Inkling S	xhigh	$35\% \pm 3\%$	25%	$0.44	36k	96	17.6

5.2 Local Correctness Does Not Ensure Production Correctness

A patch may satisfy component and integration tests yet fail when exercised end to end through a live model-serving path. SWE-Serve measures this distinction with 276 model-serving end-to-end (E2E) tests across 19 tasks. These tests launch a standalone serving process with the required model and assess its behavior through a public request or evaluation interface. The model-serving E2E tests are production-grounded: 242 (87.7%) are sourced or adapted from the SGLang repository; the remaining 34 were added by SWE-Serve to cover accepted SGLang behavior not otherwise tested.

Model-serving E2E gap. To isolate model-serving E2E requirements, we remove these tests from scoring while leaving the agent-created patches and all other tests unchanged. Across 19 tasks (627 patches), pass rate increases from 45.9% under the full verifier to 69.4% when the E2E tests are removed from scoring, a 23.4-percentage-point increase (Figure 2A). Pass rate increases for all 11 top-per-model configurations (Appendix Figure 14A).

Matched test-removal control. We test whether removing the same number of non-E2E tests produces a comparable effect. Across 12 eligible tasks, we evaluate 10,000 randomized test-removal pairs. Each pair excludes from scoring 38 E2E tests and 38 non-E2E tests matched within task and F2P/P2P role, while holding all 396 agent-created patches fixed. E2E test removal produces 16.1 fail-to-pass transitions on average, compared with 8.0 for matched non-E2E test removal, a $2.0\times$ difference. E2E test removal produces more fail-to-pass transitions in 92.2% of the randomized pairs (Appendix A.4.1). The corresponding E2E-minus-control pass-rate gap averages 2.1 points, with a 95% mask-randomization interval of -0.5 to 4.3 points.

SWE-Serve task example. The Gemma 4 MoE core-serving task illustrates this pattern: 48.5% of its agent-created patches (16 of 33) passed every non-E2E test but failed at least one E2E test (Appendix Table 10). These tests exercise loading the official checkpoint, routing each token through the top eight of the model's 128 experts, text and image serving through public interfaces, and ordered native batch generation with log probabilities (Gemma Team et al., 2026).

Sensitivity analyses. Next, we assess robustness of the pass-rate increase to task composition and tasks containing only E2E tests. First, resampling the 19 tasks 10,000 times yields a 95% bootstrap confidence interval of 13.4–34.4 points for the pass-rate increase (Appendix A.4.1). Second, after excluding the four tasks with no non-E2E tests, the pass-rate increase remains 22.8 points (Appendix Table 10).

Production inference task properties. Beyond the model-serving E2E analysis, we examine whether runtime-domain breadth, persistent state, and concurrent coordination are associated with

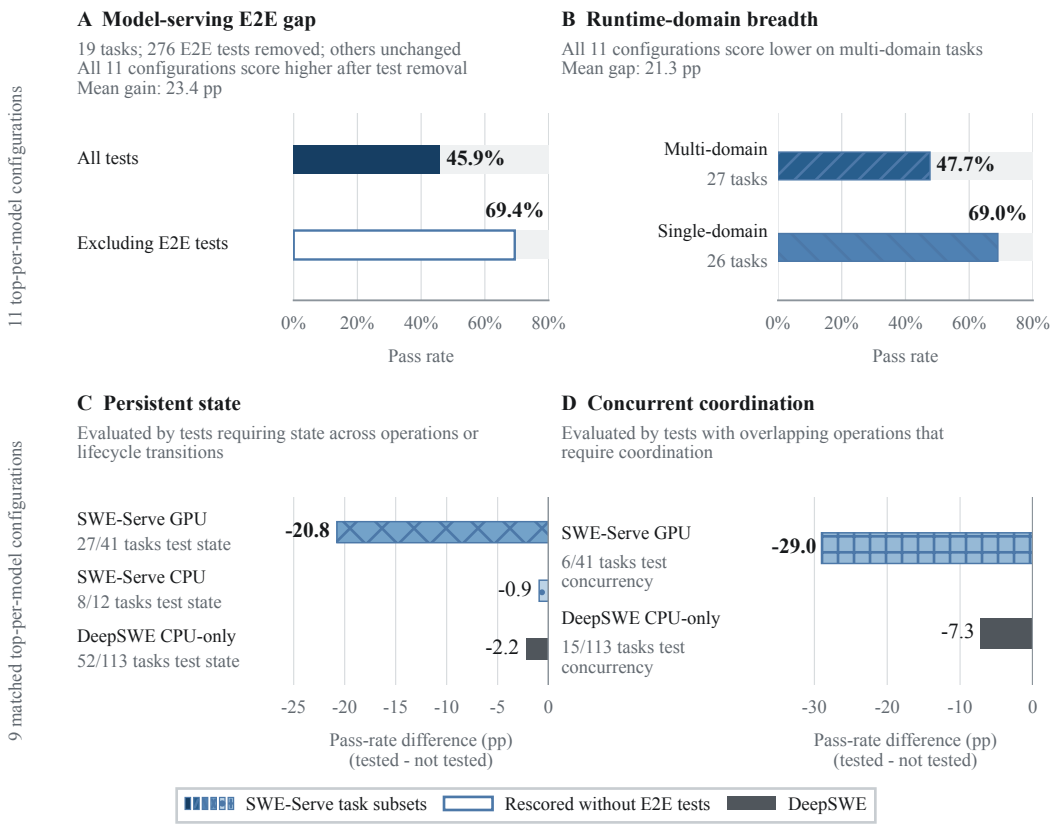


Figure 2: **Agentic implementations often fall short of production correctness. (A)** Rescoring the same patches without model-serving E2E tests raises pass rate from 45.9% to 69.4%, a 23.4-percentage-point increase across 19 tasks. **(B)** Multi-runtime-domain tasks have pass rates 21.3 percentage points lower than single-runtime-domain tasks across 11 top-per-model configurations. **(C–D)** Across nine matched configurations, SWE-Serve GPU tasks whose tests require persistent state or concurrent coordination have pass rates 20.8 and 29.0 percentage points lower, respectively, than tasks not tested for these properties; gaps are smaller on DeepSWE (Huang et al., 2026).

lower observed pass rates across the 11 top-per-model configurations. Runtime-domain breadth measures coverage across four domains in the request-to-output path (Appendix Table 4); it does not cover infrastructure, deployment, tooling, observability, or the full inference engineering stack. The 26 single-runtime-domain tasks have a mean pass rate of 69.0%, compared with 47.7% for the 27 multi-runtime-domain tasks, an average difference of 21.3 percentage points (Figure 2B). Multi-runtime-domain tasks have lower pass rates in every configuration (Appendix A.4.1). Persistent state and concurrent coordination also show large performance differences among SWE-Serve GPU tasks. Mean pass rate is 20.8 points lower for tasks requiring persistent state and 29.0 points lower for the six tasks that explicitly test concurrent coordination (Figure 2C–D). The corresponding gaps are small on SWE-Serve CPU and DeepSWE tasks (Huang et al., 2026).

Reasoning-effort robustness. Across the GPT-5.6 model family, increasing reasoning effort improves aggregate pass rates but does not consistently close the production correctness gaps, which remain substantial even at maximum effort (Appendix Figure 15).

5.3 Sensitivity to Agent Harness

The primary leaderboard uses mini-SWE-agent, a model-agnostic harness (Yang et al., 2024), to hold the agent scaffold constant across models. However, this may not reflect the performance achievable

with model-specific harnesses. We therefore re-evaluate the two best-performing configurations, GPT-5.6 Sol and Claude Opus 5, both at maximum effort, using their model-specific harnesses, Codex and Claude Code, respectively (OpenAI, 2026a; Anthropic, 2026a).

Switching to model-specific harnesses does not improve mean pass@1 in either evaluated configuration. mini-SWE-agent achieves 75.5% for both configurations, compared with 73.6% for GPT-5.6 Sol with Codex and 69.8% for Opus 5 with Claude Code (Figure 3). These results support using mini-SWE-agent for the primary leaderboard, while showing that harness choice can affect measured performance. The absence of an advantage for model-specific harnesses is consistent with the finding reported for the DeepSWE benchmark (Huang et al., 2026).

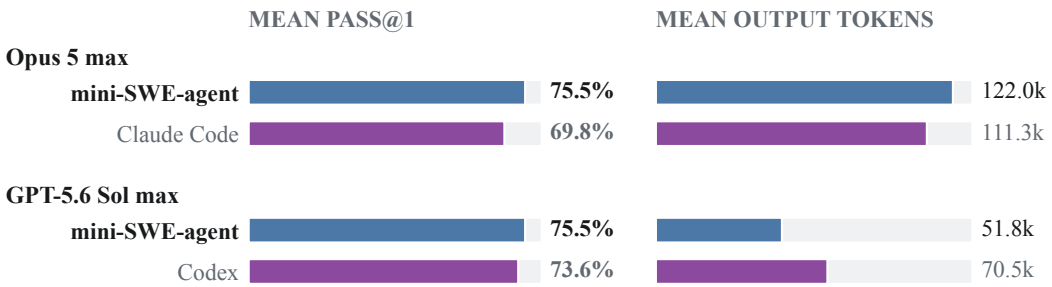


Figure 3: Model-specific harness sensitivity on 53 SWE-Serve tasks, with three runs per harness. mini-SWE-agent matches Codex for GPT-5.6 Sol and exceeds Claude Code for Opus 5.

Harness choice also affects resource trade-offs. For GPT-5.6 Sol, mini-SWE-agent achieves similar pass@1 to Codex with fewer mean output tokens. For Opus 5, mini-SWE-agent achieves higher pass@1 than Claude Code with more mean output tokens. Appendix Figures 16 and 17 compare mean cost, wall-clock time, and token counts across harnesses.

6 Limitations

SWE-Serve provides an in-depth evaluation of production inference engineering. Extending executable coverage beyond SGLang to other inference systems will help establish how broadly the findings generalize. We restrict execution to CPU or a single H100 to reduce evaluation costs and broaden access. This trade-off omits multi-GPU and multi-node tasks involving parallelism, disaggregation, and distributed coordination. Additionally, executable scoring measures functional, regression, integration, end-to-end, and targeted performance requirements, but does not assess every dimension of upstream readiness. Patch size and change dispersion complement these functional measurements as coarse indicators of implementation complexity, but cannot establish maintainability, architectural fit, or replace maintainer review (Appendix Figure 18 and Table 11). Finally, closed-book execution and trajectory auditing prevent agents from retrieving upstream solutions during evaluation, but cannot rule out prior exposure to public SGLang code. To support future training-corpus decontamination after release, each task includes a benchmark canary string (Appendix A.1.6). A private held-out task set would provide a complementary safeguard and remains a direction for future work.

7 Conclusion

We introduced SWE-Serve, a benchmark for evaluating agentic LLMs on challenging, repository-scale production inference engineering tasks. We qualify its tasks with executable no-op and oracle controls, agent-assisted adversarial probing, and targeted human review. Across SWE-Serve, agents often satisfy local requirements without achieving production correctness. SWE-Serve quantifies this gap directly: removing model-serving E2E tests from scoring while leaving the agent-created patches and all other tests unchanged raises mean pass rate by 23.4 percentage points; all 11 top-per-model configurations improve. SWE-Serve complements existing agentic benchmarks by enabling the field to track whether future agents close the gap between completing tasks locally and achieving production correctness.

AI USE STATEMENT

We used generative AI tools for benchmark task construction, verifier review and adversarial probing, analysis code, literature review, and manuscript editing. The authors checked citations against their sources, tested generated code, and reviewed task specifications and manuscript claims. We take responsibility for the final text, claims, and artifacts, including those produced with generative AI assistance.

REPRODUCIBILITY STATEMENT

Section 3 defines the 53-task benchmark release, task contract, construction, and qualification evidence. Section 4 specifies the common scaffold, evaluation budgets, replication, scoring, rerun policy, and closed-book audit, while Appendix Table 7 records the API interfaces, resolved reasoning controls, serving configurations, inference hardware where applicable, token limits, and native-harness versions. We plan to release the benchmark tasks, verifiers, runbook, and preflight instructions in a public GitHub repository.

ACKNOWLEDGMENTS

We thank the SGLang maintainers and contributors for developing and maintaining the open-source codebase on which SWE-Serve's benchmark tasks are based. We especially thank the maintainers for their task suggestions, helpful pointers, and feedback throughout the development of this work.

REFERENCES

Anthropic. Claude Code overview. Claude Code documentation, 2026a. URL https://code.claude.com/docs/en/overview.

Anthropic. Claude Opus 5 system card. Anthropic system card, July 2026b. URL https://www.anthropic.com/claude-opus-5-system-card.

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

Datacurve. DeepSWE benchmark repository. GitHub repository, 2026. URL https://github.com/datacurve-ai/deep-swe.

DeepSeek-AI. DeepSeek-V4-Flash-0731 model card. Hugging Face model card, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4-Flash-0731.

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

Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa R Kundurthy, Sean M. Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson, Bing Liu, and Brad Kenstler. SWE-bench pro: Can AI agents solve long-horizon software engineering tasks? In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=uEVTdoAbnK.

Yukang Feng, Jianwen Sun, Zelai Yang, Jiaxin Ai, Chuanhao Li, Zizhen Li, Fanrui Zhang, Kang He, Rui Ma, Jifan Lin, Jie Sun, Yang Xiao, Sizhuo Zhou, Wenxiao Wu, Yiming Liu, Pengfei Liu, Shenglin Zhang, and Kaipeng Zhang. LongCLI-bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 29952–29963, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1497. URL https://aclanthology.org/2026.findings-acl.1497/.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adamek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat

Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bražinskas, Ryan Burnell, Robert Busa-Fekete, Mu Cai, Daniele Calandriello, Glenn Cameron, Charlotte Caucheteux, Rahma Chaabouni, Garima Chadha, Jetha Chan, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Cheng, Tzu hsiang Chien, Nikolai Chinaev, Yi Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Li Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vlad Feinberg, Milen Ferev, Sebastian Flennerhag, Takumi Fujimoto, João Gabriel Oliveira, Isaac Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Tamara von Glehn, Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Mayank Gupta, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra Harlalka, Demis Hassabis, Anja Hauth, Joe Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiaopeng Huang, Yangsibo Huang, Kevin Hui, Adrian Hutter, Te I, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Ivan Korotkov, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Vivek Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungji Lee, Dmitry Lepikhin, Alon Levkovitch, Dongdong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Ivan Lobov, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Maciej Mikuła, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Brendan O'Donoghue, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Nicolas Perez-Nieves, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Plomecka, Siim Poder, Octavio Ponce, Arijit Pramanik, David Racz, Anish Rajan, Michelle Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikołaj Rybiński, Noveen Sachdeva, Michaël E. Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, George Scrivener, Benoit Seguin, Andrew Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Bobak Shahriari, Yuan Shangguan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Tafti, Jean Tarbouriech, Chetan Tekur, Shantanu Thakoor, Rahul Thapa, Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Çağlar Ünlü, Petar Veličković, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Cindy Wu, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, Arun Yadav, Jun Yan, Antoine Yang, Lin Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Morteza Zadimoghaddam, Sajjad Zafar, Fred Zhang, Jiageng Zhang, Jianyi Zhang, Xiaofan Zhang, Chao Zhao, David Zhou, and Chen Zou. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

GLM-5-Team, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu, Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye

Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. GLM-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Google DeepMind. Gemini 3.6 Flash model card. Google DeepMind model card, July 2026. URL https://deepmind.google/models/model-cards/gemini-3-6-flash/.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments, June 2026. URL https://github.com/harbor-framework/harbor/tree/v0.13.1.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Wenqi Huang, Charley Lee, Leonard Tng, and Serena Ge. DeepSWE: Measuring frontier coding agents on original, long-horizon engineering tasks, 2026. URL https://arxiv.org/abs/2607.07946.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, pp. 54107–54157, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang,

Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi K3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Tue Le, Minh V. T. Thai, Dung Nguyen Manh, Huy Phan Nhat, and Nghi D. Q. Bui. SWE-EVO: Benchmarking coding agents in long-horizon software evolution scenarios, 2026. URL https://arxiv.org/abs/2512.18470.

Edward Lin, Sahil Modi, Siva Kumar Sastry Hari, Qijing Huang, Zhifan Ye, Nestor Qin, Fengzhe Zhou, Yuan Zhang, Jingquan Wang, Sana Damani, Dheeraj Peri, Ouye Xie, Aditya Kane, Moshe Maor, Michael Behar, Triston Cao, Rishabh Mehta, Vartika Singh, Vikram Sharma Mailthody, Terry Chen, Zihao Ye, Hanfeng Chen, Tianqi Chen, Vinod Grover, Wei Chen, Wei Liu, Eric Chung, Luis Ceze, Roger Bringmann, Cyril Zeller, Michael Lightstone, Christos Kozyrakis, and Humphrey Shi. SOL-ExecBench: Speed-of-light benchmarking for real-world GPU kernels against hardware limits, 2026. URL https://arxiv.org/abs/2603.19173.

Jane Luo, Xin Zhang, Steven Liu, Jie Wu, Jianfeng Liu, Yiming Huang, Yangyu Huang, Chengyu Yin, Ying Xin, Yuefeng Zhan, Hao Sun, Qi Chen, Scarlett Li, and Mao Yang. RPG: A repository planning graph for unified and scalable codebase generation. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 91932–91985, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/9482f45fdd89aba9130bb04c44f788a9-Paper-Conference.pdf.

Ryan Marten, Alex Shaw, Ivan Bercovich, Benedikt Droste, Tommaso Cerruti, Steven Dillmann, Ruiyang Wang, Dariush Wahdany, Allen Hart, Karl Krauth, ScaleAI, Snorkel AI, Turing, gNucleus AI, Boolean AI, Nicholas Carlini, Shengrui Lyu, Anjiang Wei, Arpandeep Khatua, Björn Plüster, Chaitanya Dwivedi, Christine Sutcliffe, Yuming, David Tivris, Di Wang, Hui Wen Goh, Hanwen

Xing, Haowei Lin, Irakli Salia, Jaejung Seol, Jiajun Bao, Jialin Ouyang, Junha Park, Liam Walsh, Luyang Kong, Maksim Ivanov, Malte Ubl, Mikhail Liamets, Orfeas Menis, Piotr Migdal, Qingquan Bao, Raj Movva, Roey Ben Chaim, Namburi Srinath, Sergey Bogdanik, Shubham Yadav, Stephen Benjamin, Tony Kung, Walker Hughes, Xin Lan, Himanshu Gupta, Swaroop Mishra, Chenguang Wang, Hao He, Jianhong Tu, Kyle Montgomery, Zengji Tu, Atharva Naik, David Mortensen, Ivan Zhang, Yash Mathur, Emmy Liu, Karanpartap Singh, Michael Yu, Steven Feng, Varun Gangal, Zhuofu Tao, Sherry Ruan, Jonas Mueller, Joan Cabezas, Justin Bauer, Kevin Xiang Li, Robert Zhang, Aaron Feller, Alec Madayan, Leon Chen, Ben Feuer, Xiangyi Li, Boxuan Li, Harsh Raj, Samuel Galler, Lin Shi, Ivgeni Segal, Kelly Buchanan, Shreyas P., Rishi Desai, Aaron Schneider, Chris Settles, Xiangning Lin, Marianna Nezhurina, Andrew Wang, Michał Kowalczyk, Jin-Xiang Zhao, Sanyam Satia, Jessie Hu, Sherif Atef, Kobe Chen, Sam Vance, Gian Segato, Jenia Jitsev, Alex Dimakis, Mike Merrill, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench, July 2026. URL https://github.com/harbor-framework/terminal-bench.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E. Kelly Buchanan, Junhong Shen, Guanghao Ye, Haowei Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Di Lu, Orfeas Menis Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Guha, Gabriel Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xiangning Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong (Ryan) Wang, Changzhi Zhou, David Heineman, Hange Liu, Harsh Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wuwei Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Christopher Rytting, Ryan Marten, Yixin Wang, Jenia Jitsev, Alex Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/444a3737adaee10d86ad2ef5f74468e6-Paper-Conference.pdf.

Ayush Nangia, Shikhar Mishra, Aman Gokrani, and Paras Chopra. ISO-Bench: Can coding agents optimize real-world inference workloads?, 2026. URL https://arxiv.org/abs/2602.19594.

NVIDIA. Architecture overview. TensorRT-LLM documentation, 2026a. URL https://nvidia.github.io/TensorRT-LLM/latest/torch/arch_overview.html.

NVIDIA. Triton architecture. NVIDIA Triton Inference Server documentation, 2026b. URL https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/user_guide/architecture.html.

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

OpenAI. Codex CLI. OpenAI documentation, 2026a. URL https://learn.chatgpt.com/docs/codex/cli.

OpenAI. GPT-5.6 system card. OpenAI system card, July 2026b. URL https://deploymentsafety.openai.com/gpt-5-6.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Re, and Azalia Mirhoseini. KernelBench: Can LLMs write efficient GPU kernels? In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47356–47415. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/ouyang25a.html.

Poolside. Laguna S 2.1 model card. Hugging Face model card, 2026. URL https://huggingface.co/poolside/Laguna-S-2.1.

SGLang. SGLang source repository. GitHub repository, revision f825d7293631, 2026. URL https://github.com/sgl-project/sglang/tree/f825d729363136a2d4a4b330fa694d0b37a878fa.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, Alice Xiang, Alicia Parrish, Allen Nie, Aman Hussain, Amanda Askell, Amanda Dsouza, Ambrose Slone, Ameet Rahane, Anantharaman S. Iyer, Anders Johan Andreassen, Andrea Madotto, Andrea Santilli, Andreas Stuhlmüller, Andrew M. Dai, Andrew La, Andrew Kyle Lampinen, Andy Zou, Angela Jiang, Angelica Chen, Anh Vuong, Animesh Gupta, Anna Gottardi, Antonio Norelli, Anu Venkatesh, Arash Gholamidavoodi, Arfa Tabassum, Arul Menezes, Arun Kirubarajan, Asher Mullokandov, Ashish Sabharwal, Austin Herrick, Avia Efrat, Aykut Erdem, Ayla Karakaş, B. Ryan Roberts, Bao Sheng Loe, Barret Zoph, Bartłomiej Bojanowski, Batuhan Özyurt, Behnam Hedayatnia, Behnam Neyshabur, Benjamin Inden, Benno Stein, Berk Ekmekci, Bill Yuchen Lin, Blake Howald, Bryan Orinion, Cameron Diao, Cameron Dour, Catherine Stinson, Cedrick Argueta, Cesar Ferri, Chandan Singh, Charles Rathkopf, Chenlin Meng, Chitta Baral, Chiyu Wu, Chris Callison-Burch, Christopher Waites, Christian Voigt, Christopher D Manning, Christopher Potts, Cindy Ramirez, Clara E. Rivera, Clemencia Siro, Colin Raffel, Courtney Ashcraft, Cristina Garbacea, Damien Sileo, Dan Garrette, Dan Hendrycks, Dan Kilman, Dan Roth, C. Daniel Freeman, Daniel Khashabi, Daniel Levy, Daniel Moseguí González, Danielle Perszyk, Danny Hernandez, Danqi Chen, Daphne Ippolito, Dar Gilboa, David Dohan, David Drakard, David Jurgens, Debajyoti Datta, Deep Ganguli, Denis Emelin, Denis Kleyko, Deniz Yuret, Derek Chen, Derek Tam, Dieuwke Hupkes, Diganta Misra, Dilyar Buzan, Dimitri Coelho Mollo, Diyi Yang, Dong-Ho Lee, Dylan Schrader, Ekaterina Shutova, Ekin Dogus Cubuk, Elad Segal, Eleanor Hagerman, Elizabeth Barnes, Elizabeth Donoway, Ellie Pavlick, Emanuele Rodolà, Emma Lam, Eric Chu, Eric Tang, Erkut Erdem, Ernie Chang, Ethan A Chi, Ethan Dyer, Ethan Jerzak, Ethan Kim, Eunice Engefu Manyasi, Evgenii Zheltonozhskii, Fanyue Xia, Fatemeh Siar, Fernando Martínez-Plumed, Francesca Happé, Francois Chollet, Frieda Rong, Gaurav Mishra, Genta Indra Winata, Gerard de Melo, Germàn Kruszewski, Giambattista Parascandolo, Giorgio Mariani, Gloria Xinyue Wang, Gonzalo Jaimovitch-Lopez, Gregor Betz, Guy Gur-Ari, Hana Galijasevic, Hannah Kim, Hannah Rashkin, Hannaneh Hajishirzi, Harsh Mehta, Hayden Bogar, Henry Francis Anthony Shevlin, Hinrich Schuetze, Hiromu Yakura, Hongming Zhang, Hugh Mee Wong, Ian Ng, Isaac Noble, Jaap Jumelet, Jack Geissinger, Jackson Kernion, Jacob Hilton, Jaehoon Lee, Jaime Fernández Fisac, James B Simon, James Koppel, James Zheng, James Zou, Jan Kocon, Jana Thompson, Janelle Wingfield, Jared Kaplan, Jarema Radom, Jascha Sohl-Dickstein, Jason Phang, Jason Wei, Jason Yosinski, Jekaterina Novikova, Jelle Bosscher, Jennifer Marsh, Jeremy Kim, Jeroen Taal, Jesse Engel, Jesujoba Alabi, Jiacheng Xu, Jiaming Song, Jillian Tang, Joan Waweru, John Burden, John Miller, John U. Balis, Jonathan Batchelder, Jonathan Berant, Jörg Frohberg, Jos Rozen, Jose Hernandez-Orallo, Joseph Boudeman, Joseph Guerr, Joseph Jones, Joshua B. Tenenbaum, Joshua S. Rule, Joyce Chua, Kamil Kanclerz, Karen Livescu, Karl Krauth, Karthik Gopalakrishnan, Katerina Ignatyeva, Katja Markert, Kaustubh Dhole, Kevin Gimpel, Kevin Omondi, Kory Wallace Mathewson, Kristen Chiafullo, Ksenia Shkaruta, Kumar Shridhar, Kyle McDonell, Kyle Richardson, Laria Reynolds, Leo Gao, Li Zhang, Liam Dugan, Lianhui Qin, Lidia Contreras-Ochando, Louis-Philippe Morency, Luca Moschella, Lucas Lam, Lucy Noble, Ludwig Schmidt, Luheng He, Luis Oliveros-Colón, Luke Metz, Lütfi Kerem Senel, Maarten Bosma, Maarten Sap, Maartje Ter Hoeve, Maheen Farooqi, Manaal Faruqui, Mantas Mazeika, Marco Baturan, Marco Marelli, Marco Maru, Maria Jose Ramirez-Quintana, Marie Tolkiehn, Mario Giulianelli, Martha Lewis, Martin Potthast, Matthew L Leavitt, Matthias Hagen, Mátyás Schubert, Medina Orduna Baitemirova, Melody Arnaud, Melvin McElrath, Michael Andrew Yee, Michael Cohen, Michael Gu, Michael Ivanitskiy, Michael Starritt, Michael Strube, Michał Swędrowski, Michele Bevilacqua, Michihiro Yasunaga, Mihir Kale, Mike Cain, Mimee Xu, Mirac Suzgun, Mitch Walker, Mo Tiwari, Mohit Bansal, Moin Aminnaseri, Mor Geva, Mozhdeh Gheini, Mukund Varma T, Nanyun Peng, Nathan Andrew Chi, Nayeon Lee, Neta Gur-Ari Krakover, Nicholas Cameron, Nicholas Roberts, Nick Doiron, Nicole Martinez, Nikita Nangia, Niklas Deckers, Niklas Muennighoff, Nitish Shirish Keskar, Niveditha S. Iyer, Noah Constant, Noah Fiedel, Nuan Wen, Oliver Zhang, Omar Agha, Omar Elbaghdadi, Omer Levy, Owain Evans, Pablo Antonio Moreno Casares, Parth Doshi, Pascale Fung, Paul Pu Liang, Paul Vicol, Pegah Alipoormolabashi, Peiyuan Liao, Percy Liang, Peter W Chang, Peter Eckersley, Phu Mon Htut,

Pinyu Hwang, Piotr Miłkowski, Piyush Patil, Pouya Pezeshkpour, Priti Oli, Qiaozhu Mei, Qing Lyu, Qinlang Chen, Rabin Banjade, Rachel Etta Rudolph, Raefer Gabriel, Rahel Habacker, Ramon Risco, Raphaël Millière, Rhythm Garg, Richard Barnes, Rif A. Saurous, Riku Arakawa, Robbe Raymaekers, Robert Frank, Rohan Sikand, Roman Novak, Roman Sitelew, Ronan Le Bras, Rosanne Liu, Rowan Jacobs, Rui Zhang, Russ Salakhutdinov, Ryan Andrew Chi, Seungjae Ryan Lee, Ryan Stovall, Ryan Teehan, Rylan Yang, Sahib Singh, Saif M. Mohammad, Sajant Anand, Sam Dillavou, Sam Shleifer, Sam Wiseman, Samuel Gruetter, Samuel R. Bowman, Samuel Stern Schoenholz, Sanghyun Han, Sanjeev Kwatra, Sarah A. Rous, Sarik Ghazarian, Sayan Ghosh, Sean Casey, Sebastian Bischoff, Sebastian Gehrmann, Sebastian Schuster, Sepideh Sadeghi, Shadi Hamdan, Sharon Zhou, Shashank Srivastava, Sherry Shi, Shikhar Singh, Shima Asaadi, Shixiang Shane Gu, Shubh Pachchigar, Shubham Toshniwal, Shyam Upadhyay, Shyamolima Shammie Debnath, Siamak Shakeri, Simon Thormeyer, Simone Melzi, Siva Reddy, Sneha Priscilla Makini, Soo-Hwan Lee, Spencer Torene, Sriharsha Hatwar, Stanislas Dehaene, Stefan Divic, Stefano Ermon, Stella Biderman, Stephanie Lin, Stephen Prasad, Steven Piantadosi, Stuart Shieber, Summer Misherghi, Svetlana Kiritchenko, Swaroop Mishra, Tal Linzen, Tal Schuster, Tao Li, Tao Yu, Tariq Ali, Tatsunori Hashimoto, Te-Lin Wu, Théo Desbordes, Theodore Rothschild, Thomas Phan, Tianle Wang, Tiberius Nkinyili, Timo Schick, Timofei Kornev, Titus Tunduny, Tobias Gerstenberg, Trenton Chang, Trishala Neeraj, Tushar Khot, Tyler Shultz, Uri Shaham, Vedant Misra, Vera Demberg, Victoria Nyamai, Vikas Raunak, Vinay Venkatesh Ramasesh, vinay uday prabhu, Vishakh Padmakumar, Vivek Srikumar, William Fedus, William Saunders, William Zhang, Wout Vossen, Xiang Ren, Xiaoyu Tong, Xinran Zhao, Xinyi Wu, Xudong Shen, Yadollah Yaghoobzadeh, Yair Lakretz, Yangqiu Song, Yasaman Bahri, Yejin Choi, Yichi Yang, Sophie Hao, Yifu Chen, Yonatan Belinkov, Yu Hou, Yufang Hou, Yuntao Bai, Zachary Seid, Zhuoye Zhao, Zijian Wang, Zijie J. Wang, Zirui Wang, and Ziyi Wu. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj. Featured Certification.

Thinking Machines Lab. Inkling-Small model card. Thinking Machines Lab model card, July 2026. URL https://thinkingmachines.ai/model-card/inkling-small/.

vLLM. Architecture overview. vLLM documentation, 2026. URL https://docs.vllm.ai/en/latest/design/arch_overview/.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 50528–50652. Curran Associates, Inc., 2024. doi: 10.52202/079017-1601. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf.

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?, 2026. URL https://arxiv.org/abs/2605.03546.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 9965–10017, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/1b126cc38b8638e07bef37e7b2bb72bf-Paper-Conference.pdf.

Jehyeok Yeon, Ben Rank, and Maksym Andriushchenko. InferenceBench: A benchmark for open-ended LLM inference optimization by AI agents, 2026. URL https://arxiv.org/abs/2607.20468.

Z.ai. GLM-5.2: Built for long-horizon tasks. Model release, June 2026. URL https://huggingface.co/blog/zai-org/glm-52-blog.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang:

efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Qixing Zhou, JiaCheng Zhang, Haiyang Wang, Rui Hao, Jiahe Wang, Minghao Han, Yuxue Yang, Shuzhe Wu, Feiyang Pan, Lue Fan, Dandan Tu, and Zhaoxiang Zhang. FeatureBench: Benchmarking agentic coding for complex feature development. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 22019–22055, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/25203d1cc8c58381eab578f4fcf9c4f8-Paper-Conference.pdf.

A APPENDIX

APPENDIX CONTENTS

Section		Page
A.1	**Benchmark composition, construction, and qualification**	20
A.1.1	Composition and scope	20
A.1.2	Development of the runtime-domain taxonomy	20
A.1.3	Model-serving end-to-end (E2E) test classification	21
A.1.4	Task characterization	22
A.1.5	Source discovery, task construction, and qualification	25
A.1.6	Release canary	29
A.2	**Experimental setup**	30
A.2.1	Cross-benchmark evaluation wall-clock time	30
A.2.2	Attempt outcomes and execution limits	31
A.2.3	Model, serving, and harness configurations	32
A.2.4	Leaderboard aggregation	33
A.2.5	Closed-book evaluation and retrieval calibration	33
A.3	**Agentic performance and resource trade-offs**	34
A.3.1	Performance across 31 model–effort configurations	34
A.3.2	Performance and resource trade-offs	35
A.3.3	Performance by inference engineering family	39
A.4	**Production correctness**	40
A.4.1	Local correctness does not ensure production correctness	40
A.4.2	Reasoning-effort robustness of production correctness gaps	43
A.5	**Sensitivity to agent harness**	45
A.6	**Limitations of executable scoring**	47

A.1 Benchmark composition, construction, and qualification

This section expands the composition, scope, construction, and qualification evidence in Section 3.

A.1.1 Composition and scope

Figure 4 complements the summaries in Table 2B with the full distribution of oracle scope, including the number of lines and files changed. The full distribution highlights that the median oracle solution changes 553 lines across seven files, while individual tasks range from 11 to 6,077 lines and from one to 35 files, showing that SWE-Serve includes both localized changes and broader implementations.

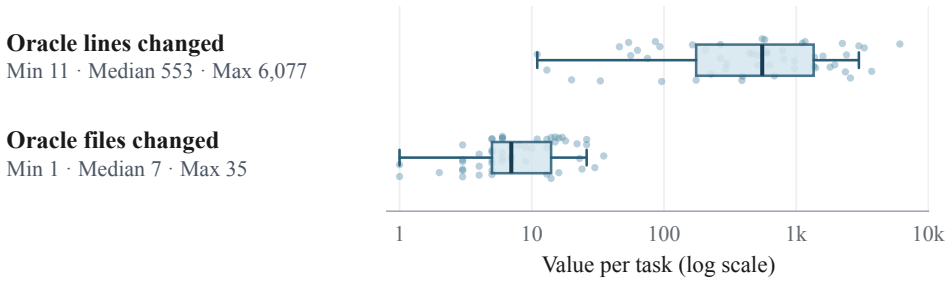


Figure 4: Oracle patch size and file count across SWE-Serve tasks. Dots show individual tasks; boxes show medians and interquartile ranges (IQRs). Whiskers reach the most extreme observations within $1.5\times$ IQR of the box edges, computed in original units. The value axis is logarithmic. Oracle lines changed counts additions plus deletions.

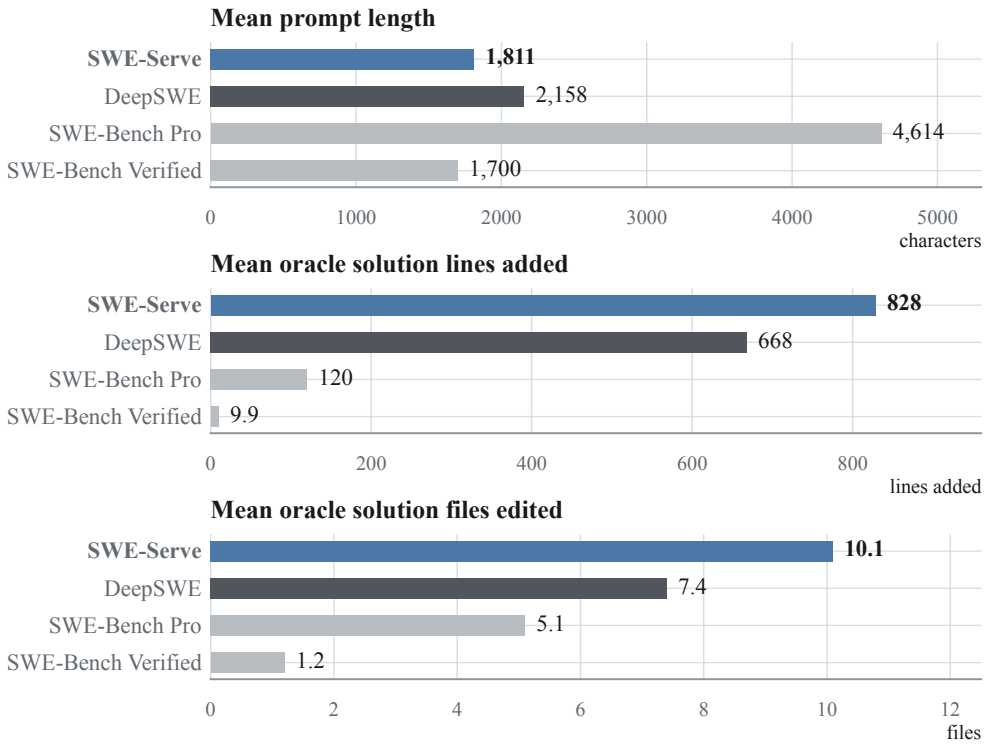
Figure 5 shows that SWE-Serve combines concise task instructions with substantial oracle implementation scope. Its prompts are shorter on average than those of DeepSWE and SWE-bench Pro, while its oracle solutions add more lines and modify more files on average than those of DeepSWE, SWE-bench Pro, and SWE-bench Verified. Instructions describe the desired behavior, and a separate executable verifier checks whether the agent's solution satisfies the task's requirements. This allows prompts to remain concise while leaving agents responsible for exploring the repository, planning the implementation, and producing a patch that passes the tests.

A.1.2 Development of the runtime-domain taxonomy

To characterize the diversity of runtime responsibilities represented in SWE-Serve tasks, we developed a runtime-domain taxonomy from documented components in four prominent open-source inference systems: vLLM (Kwon et al., 2023), SGLang (Zheng et al., 2024), TensorRT-LLM (NVIDIA, 2026a), and Triton (NVIDIA, 2026b). We selected these systems for their popularity, as each official repository exceeded 10,000 GitHub stars on September 1, 2026[1], and for their differing scopes: vLLM, SGLang, and TensorRT-LLM are LLM-serving runtime stacks, while Triton is a broader inference server.

We examined the responsibilities of documented components along the path from receiving a request to producing its output, then grouped components with corresponding responsibilities across systems into four domains: **request handling and I/O**, covering input and output representation; **scheduling and request lifecycle**, covering admission, queuing, batching, and request progression; **model execution**, covering forward computation, decoding, attention, kernels, and backend invocation; and **KV cache and runtime-resource management**, covering the allocation and lifecycle of persistent inference-time state. These categories describe shared responsibilities across systems whose component names and implementations differ. For example, Triton's resource-management components handle sequence state and backend-managed resources, whereas the LLM-serving systems include explicit KV-cache management.

[1]GitHub repository snapshots: vLLM (90.7K), SGLang (33.0K), TensorRT-LLM (14.5K), and Triton (11.0K).



DeepSWE, SWE-bench Pro, SWE-bench Verified: reproduced from DeepSWE Figure 2.

Figure 5: SWE-Serve's oracle solutions add more lines and modify more files on average than those of DeepSWE, SWE-bench Pro, and SWE-bench Verified, while its prompts are shorter on average than those of DeepSWE and SWE-bench Pro. Bars show mean prompt length in characters, lines added by the oracle solution, and files changed. SWE-Serve values cover the full benchmark; comparator values are reproduced from DeepSWE Figure 2 (Huang et al., 2026).

Table 4 maps these responsibilities to documented components, with links to the supporting passages in pinned repository versions (vLLM, 2026; SGLang, 2026; NVIDIA, 2026a;b). We finalized the taxonomy and mapping before examining model outcomes, then used the taxonomy to characterize the responsibilities exercised by each task's executable tests.

A.1.3 Model-serving end-to-end (E2E) test classification

Motivated by SGLang maintainer feedback, we classified a test as model-serving E2E only if it met all six requirements below:

1. It uses a real pretrained checkpoint.
2. It executes the actual production SGLang runtime without mocked or synthetic private substitutes.
3. It drives the behavior under test through that runtime.
4. It makes a caller-visible behavioral assertion.
5. It launches a standalone serving process.
6. It exercises that process through a public request or evaluation boundary, such as HTTP/OpenAI-compatible requests or an evaluation runner targeting the server.

A task has model-serving E2E coverage when its executable verifier includes at least one such test that contributes to the task's score.

Table 4: Evidence for constructing the runtime-domain taxonomy from four prominent open-source inference systems. Documented components illustrate four recurring responsibilities: request handling and I/O, scheduling and request lifecycle, model execution, and KV cache and runtime-resource management. Bracketed keys link to the corresponding evidence in pinned repository versions.

System	Request / I/O	Scheduling / lifecycle	Model execution	KV / runtime resources
vLLM	API server and input/output processors [V-IO1, V-IO2]	EngineCore scheduler [V-SL]	Executor, GPU worker, ModelRunner [V-ME1, V-ME2]	KV cache manager [V-RM]
SGLang	HTTP server, TokenizerManager, Detokenizer Manager [S-IO]	Scheduler, request queues, batch construction [S-SL1, S-SL2]	TP worker, ModelRunner, attention and kernels [S-ME1, S-ME2, S-ME3, S-ME4]	KV pools, allocators, RadixCache [S-RM1, S-RM2]
TensorRT-LLM	LLM interface and tokenization/ detokenization [T-IO]	PyExecutor scheduler [T-SL]	Model engine and decoder [T-ME]	ResourceManager and KVCacheManager [T-RM]
Triton	HTTP/REST, gRPC, C API, returning outputs [R-IO]	Per-model schedulers and batchers [R-SL]	Model backends and instances [R-ME1, R-ME2]	Sequence state and backend-managed resources [R-RM1, R-RM2]

A.1.4 Task characterization

Before examining any model outcomes, we defined four task characteristics: runtime-domain breadth using the taxonomy in Appendix A.1.2, persistent state and concurrent coordination as defined in Section 3.2, and model-serving E2E coverage as defined in Appendix A.1.3. Using these definitions, we generated annotations for each characteristic across all SWE-Serve tasks. The annotation-creation process used only each task's instruction and the tests in its executable verifier. We assigned a characteristic to a task only when it was tested by the verifier, making it part of the task's executable correctness requirements. An unassigned characteristic therefore means that the verifier does not establish that requirement; it does not imply that an implementation that solves the task cannot choose to use persistent state, coordinate concurrent operations, span additional runtime domains, or execute an E2E serving path.

Beyond these four task characteristics, additional properties associated with each task include its engineering family (summarized in Table 2A), required execution hardware, and the presence of performance gates. Table 5 reports the full characterization of each SWE-Serve task.

Table 5: Task characterization of all SWE-Serve tasks. Engineering family identifies the type of engineering work. HW denotes the required execution hardware; all GPU tasks use an H100. E2E test denotes a model-serving end-to-end test, and Perf. gate abbreviates performance gate. In the model-serving E2E test, persistent state, concurrent behavior, and performance gate columns, a checkmark indicates a characteristic tested by the verifier as a correctness requirement. A dash means that the verifier does not establish this characteristic as a correctness requirement. The runtime-domains column indicates whether the verifier tests one (Single) or multiple (Multi) runtime domains. Runtime-domain codes are I=request handling and I/O, S=scheduling and request lifecycle, M=model execution, and K=key-value (KV) cache and runtime-resource management.

Task	Engineering family	HW	E2E test	Runtime domains	Persistent state	Concurrent behavior	Perf. gate
Adaptive EAGLE step control + config guards	Speculative / decoding	CPU	–	Single (M)	✓	–	–
NVFP4 and DeepSeek MoE inference optimizations	Kernels / quant. / perf.	CPU	–	Multi (M+K)	✓	–	–
Gemma 4 fused Triton operations	Kernels / quant. / perf.	GPU	–	Single (M)	–	–	–
Masked MoE activation + GDN QKV-split kernels	Kernels / quant. / perf.	GPU	–	Single (M)	–	–	–
HiCache eviction and tombstone correctness	Caching / runtime state	GPU	–	Multi (S+K)	✓	–	–
HiCache framework + sliding-window attention	Caching / runtime state	GPU	–	Multi (S+K)	✓	–	–
Custom spec-algorithm registry + N-gram metric fix	Speculative / decoding	GPU	–	Single (M)	✓	–	–
Anthropic Messages request/response conversion	Serving API / runtime	CPU	–	Single (I)	–	–	–
Spec: forward-timeout ordering fix (EAGLE v1)	Speculative / decoding	GPU	✓	Multi (S+M)	✓	–	–
Mamba state offloading + HybridCacheController	Caching / runtime state	GPU	–	Single (K)	✓	–	–
DeepSeek-V4 DSML tool-call parser	Serving API / runtime	GPU	–	Single (I)	✓	–	–
O(1) EAGLE bigram RadixKey view	Speculative / decoding	CPU	–	Single (K)	✓	–	–
torch.mm for DeepSeek-V3.2 indexer GEMM	Kernels / quant. / perf.	GPU	–	Single (M)	–	–	✓
Spec: split accept_length counters	Speculative / decoding	CPU	–	Single (M)	✓	–	–
Kimi-K2.5 EAGLE3-MLA draft path	Speculative / decoding	CPU	–	Single (M)	–	–	–
Spec: split draft-extend dataclass	Speculative / decoding	CPU	–	Single (M)	✓	–	–
PD hybrid state-transfer refactor	Distributed / scheduling	CPU	–	Single (K)	–	–	–
DeepSeek-V4 w4a4 MegaMoE	Model / backend	GPU	–	Single (M)	✓	–	–
PD disaggregation priority-scheduling fix	Distributed / scheduling	GPU	–	Single (S)	✓	–	–

Continued on next page.

Table 5, continued.

Task	Engineering family	HW	E2E test	Runtime domains	Persistent state	Concurrent behavior	Perf. gate
Kimi tokenizer TTFT fast path	Kernels / quant. / perf.	GPU	–	Single (I)	–	–	✓
UnifiedRadixCache device-match semantics fix	Caching / runtime state	GPU	–	Single (K)	✓	–	–
TMA bulk-store MLA KV-buffer kernel	Kernels / quant. / perf.	GPU	–	Multi (M+K)	✓	–	✓
Spec-V2 paged tree drafting	Speculative / decoding	GPU	–	Single (K)	–	–	–
Ideogram4 NVFP4 config from safetensors	Model / backend	CPU	–	Single (M)	–	–	–
DFLASH v1 non-overlap serving	Speculative / decoding	GPU	✓	Multi (S+M+K)	✓	–	–
EAGLE v2 page-one tree runtime	Speculative / decoding	GPU	✓	Multi (S+M+K)	✓	✓	–
Diffusion-LLM radix-graph serving	Model / backend	GPU	✓	Multi (S+M+K)	✓	✓	–
DeepSeek-V3.2 index-cache store + paged gather	Caching / runtime state	GPU	–	Multi (M+K)	✓	–	–
SDAR dense + MoE day-0 serving	Model / backend	GPU	✓	Single (M)	–	–	–
EPD registration cleanup	Distributed / scheduling	CPU	–	Multi (I+S)	✓	–	–
Context-parallel strategy abstractions	Distributed / scheduling	CPU	–	Single (M)	✓	–	–
Chunked SGMV CUDA-graph replay fix	Serving API / runtime	GPU	–	Multi (M+K)	✓	–	–
Qwen3.5 dense + MoE core serving	Model / backend	GPU	✓	Multi (I+M)	–	–	–
Breakable prefill CUDA-graph engine	Kernels / quant. / perf.	GPU	–	Multi (M+K)	✓	–	–
HiSparse page-aware decode lifecycle + FP8 routing	Caching / runtime state	GPU	–	Multi (S+M+K)	✓	–	–
MoE LoRA route alignment + numerical kernel	Kernels / quant. / perf.	GPU	–	Single (M)	–	–	–
Mixed-chunk piecewise CUDA-graph replay	Serving API / runtime	GPU	✓	Multi (S+M+K)	✓	✓	–
Adaptive speculative allocation safety	Speculative / decoding	CPU	–	Single (K)	✓	–	–
Transformers-backend compute runtime	Model / backend	GPU	✓	Single (M)	–	–	–
Transformers 5 MoE core serving	Model / backend	GPU	✓	Single (M)	✓	–	–
Gemma 4 piecewise CUDA-graph execution	Model / backend	GPU	✓	Multi (I+M+K)	✓	–	–

Continued on next page.

Table 5, continued.

Task	Engineering family	HW	E2E test	Runtime domains	Persistent state	Concurrent behavior	Perf. gate
Gemma 4 speculative execution	Model / backend	GPU	✓	Multi (S+M+K)	✓	–	–
Gemma 4 26B-A4B core serving	Model / backend	GPU	✓	Multi (I+M)	–	–	–
Multi-item Score API packing	Serving API / runtime	GPU	–	Multi (I+S+M)	–	✓	–
N-gram trie refactor + stateful matching	Speculative / decoding	GPU	✓	Multi (S+M+K)	✓	–	–
UnifiedRadixCache streaming-session KV	Caching / runtime state	GPU	–	Multi (S+K)	✓	–	–
GLM-V repetition-penalty sampling	Serving API / runtime	GPU	✓	Single (M)	✓	–	–
N-gram external SAM corpus	Speculative / decoding	GPU	✓	Multi (I+M+K)	✓	–	–
Embedding-model LoRA serving	Model / backend	GPU	–	Multi (I+M)	–	–	–
Score API hybrid token + embedding inputs	Serving API / runtime	GPU	✓	Multi (I+M)	–	–	–
SequenceClassification Score API	Model / backend	GPU	✓	Multi (I+S+M)	–	–	–
Spec-V2 streaming-session KV commit	Speculative / decoding	GPU	✓	Multi (S+M+K)	✓	✓	–
EAGLE3 overlap metadata lifecycle	Serving API / runtime	GPU	✓	Multi (S+M)	✓	✓	–

A.1.5 SOURCE DISCOVERY, TASK CONSTRUCTION, AND QUALIFICATION

The pipeline used to generate SWE-Serve comprised three stages: source discovery and screening, task construction, and qualification and admission, as illustrated in Figure 6. Source discovery and screening used four source types to identify and screen production inference engineering contributions made to the SGLang repository since December 2025, yielding source candidates. Task construction deduplicated these source candidates and turned them into executable task candidates. Qualification and admission used executable controls, agent-assisted review and adversarial probing, and human adjudication to determine whether each task candidate should be revised and requalified, excluded, or admitted. This process produced the SWE-Serve benchmark.

Source discovery and screening.

SOURCES. We used four source types through five discovery and screening paths.

Release notes. We reviewed SGLang release-note bullets to extract individual PRs or groups of related PRs. Bullets citing multiple PRs were treated as a multi-PR source. We conducted two rounds of review, with the second incorporating newer release notes and removing candidates already retained during the first round. Together, the two rounds covered release notes published from January through June 25, 2026.

Release notes and associated PRs. We manually reviewed SGLang release notes from January through June 24, 2026, and their associated PRs to identify broader features whose scope was not fully captured by a single release-note bullet. This review connected related changes across bullets and PRs to synthesize additional feature-level source candidates.

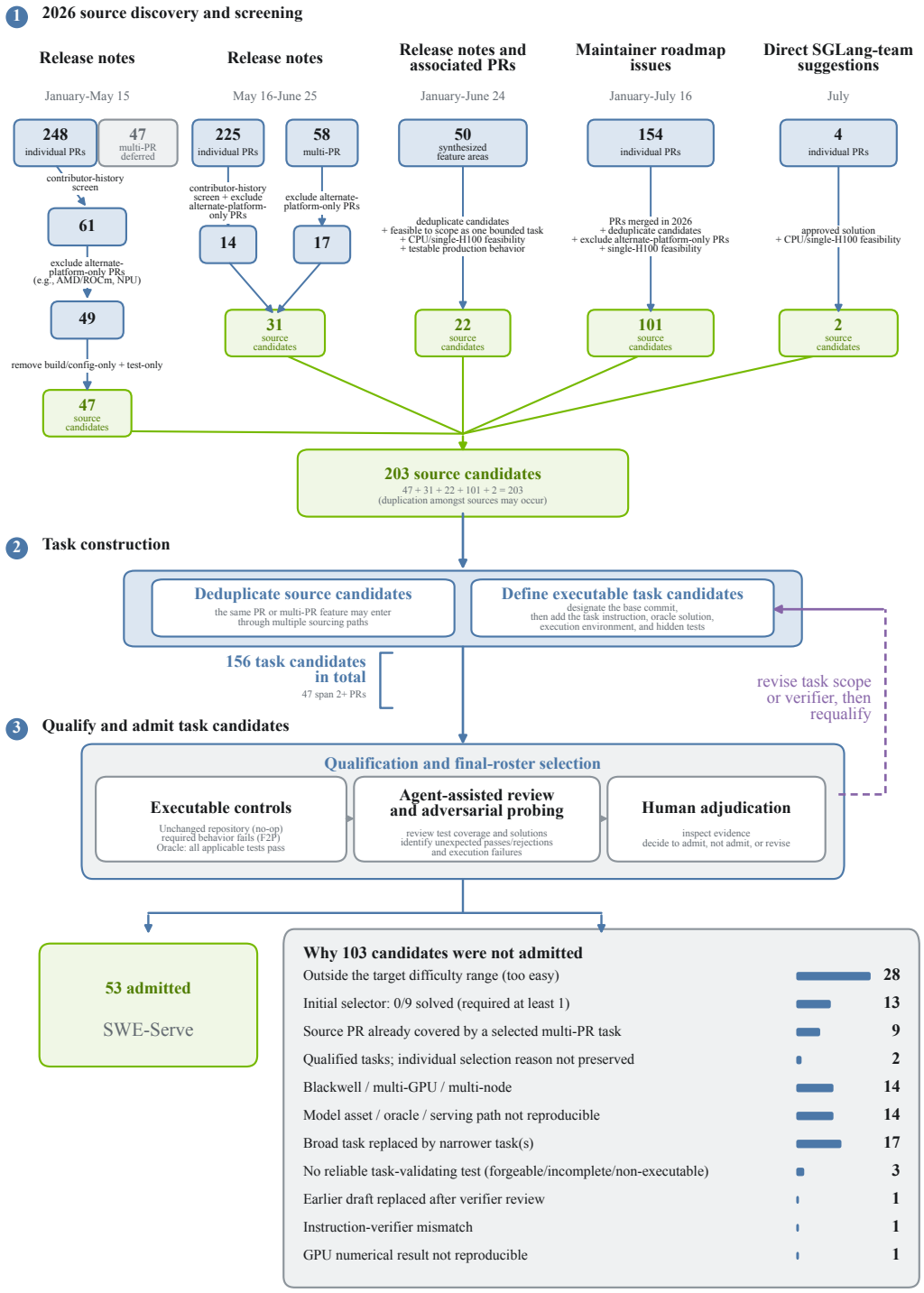


Figure 6: The 156 task candidates include every constructed or revised candidate version. Qualification may return candidates to construction for scope or verifier revision; human adjudication admits 53 task candidates and does not admit 103 candidates.

Maintainer roadmap issues. We reviewed maintainer-authored roadmap issues and extracted PRs they linked to as individual PR source candidates. PRs that were part of the same feature were subsequently combined during task construction.

Direct SGLang-team suggestions. Additionally, SGLang maintainers nominated four recent PRs for review as potential benchmark contributions.

SCREENING CRITERIA. Each discovery path applied the screening criteria specified for that path in Figure 6.

The **contributor-history screen** selected individual PRs whose authors had both authored and reviewed at least three PRs within a snapshot of the 500 most recently created SGLang PRs that had been merged at the time. This served as a recent-activity heuristic rather than a measure of contributors' entire contribution history. Additionally, authors needed to have submitted a review within that snapshot with GitHub association `OWNER`, `MEMBER`, or `COLLABORATOR`. This criterion applied to individual PRs in both rounds of release note review.

Exclude alternate-platform-only PRs. To support benchmark execution within a standardized hardware configuration with CPUs and a single H100 GPU, we excluded contributions focused only on hardware or software backends outside that configuration, such as AMD/ROCm and Ascend/NPU. Release note review identified these contributions using PR titles and labels, while roadmap review examined their implementation scope.

The **remove build/config-only + test-only screen** excluded candidates whose code changes were limited to build files, configuration, or tests. This criterion applied during the first round of release note review.

During manual feature review, we inspected the associated PR diffs to apply the deduplication, bounded-task scope, hardware-feasibility, and testable-production-behavior criteria. **Deduplication** was performed by assessing overlap with existing source candidates. Next, **feasible to scope as one bounded task** required a feature to support a clearly defined behavioral objective. Then **testable production behavior** required a way to deterministically test that behavior, although the tests could be authored during task construction.

The **CPU or single H100 GPU feasibility criterion** assessed whether the intended behavior could be exercised on CPU or a single H100 GPU, keeping benchmark execution within a standardized hardware configuration. This criterion applied to manual feature review and direct suggestions. Roadmap screening specifically required feasibility on a single H100 GPU.

For roadmap candidates, **PRs merged in 2026** restricted eligibility to changes merged from January 1 through the July 16 screening cutoff. **Deduplication** also applied during roadmap screening.

For direct suggestions, **approved solution** required each PR to have an accepted implementation that could serve as the task's reference solution.

SCREENING OUTCOME. Across the five discovery and screening paths, we retained 47 source candidates from the first round of release note review, 31 from the second round, 22 from manual review of release notes and associated PRs, 101 from maintainer roadmap issues, and two from direct SGLang-team suggestions. This yielded 203 source candidates for task construction.

Task construction.

Task construction converted 203 source candidates into 156 task-candidates, including revised versions. This process first deduplicated overlapping source candidates and then, where feasible, developed the remaining candidates into executable tasks. Each executable task-candidate required a designated base commit, task instruction, oracle solution, reproducible execution environment, and hidden tests. Only source candidates that met these requirements became task-candidates, and each could include one or several related PRs.

During qualification, some task-candidates were returned to task construction to revise their scope or verifier before being requalified.

Qualify and admit task candidates.

The final admission stage applied three qualification steps and final-roster selection to all 156 task-candidates. The qualification steps included executable controls, agent-assisted review and adversarial probing, and human adjudication. Qualification and final-roster selection resulted in 53 tasks being admitted to the SWE-Serve benchmark, while 103 task-candidates were not admitted.

EXECUTABLE CONTROLS. The unchanged base commit served as the no-op control. It was required to pass all pass-to-pass tests and fail all fail-to-pass tests. The oracle solution was required to pass all applicable tests, including both pass-to-pass and fail-to-pass tests.

AGENT-ASSISTED REVIEW AND ADVERSARIAL PROBING. Review assessed whether the tests covered the behavioral requirements stated in the task instructions. Agent-generated solutions and execution traces were also used to identify incomplete or shortcut solutions that unexpectedly passed the verifier, valid solutions that were incorrectly rejected, and failures caused by the execution environment. These findings provided evidence for human adjudication of whether candidates required revision, exclusion, or admission.

HUMAN ADJUDICATION. Human reviewers examined evidence from the executable controls and the agent-assisted review and adversarial probing to determine whether each task-candidate should be admitted, excluded, or revised and requalified.

TASK-CANDIDATE REVISION AND REQUALIFICATION. Task scope and verifier revisions occurred throughout construction and qualification, with revised candidates undergoing requalification. For example, one qualification review of 54 task-candidates identified 27 requiring verifier changes. Of these, 22 involved changes to the tests used for scoring, while the remaining five involved verifier execution or how the verifier determined which source files to trust. Twenty-three of the final 53 tasks incorporated verifier revisions from this review.

VALIDATION COVERAGE. Table 6 summarizes validation coverage for the admitted SWE-Serve tasks. All received no-op and oracle controls, test coverage and scoring review (F2P/P2P), and task-specific agent-assisted adversarial review of candidate solutions. Nine tasks also triggered additional review because no agent attempts had passed. Reviewers examined the instructions, tests, proposed solutions, verifier outputs, and execution traces before interpreting these results as evidence of difficulty. Four tasks that triggered this review later recorded successful attempts. The remaining five had no successful attempts across the evaluated runs and received task-specific analyses of the causes of failure.

Table 6: Validation coverage for all SWE-Serve tasks. "Applicable" in the last row refers to the nine tasks that triggered zero-pass review based on results available at the time of assessment. Four of these tasks subsequently recorded successful attempts.

Validation component	Coverage
No-op rejection	53/53
Oracle acceptance	53/53
Test coverage and scoring review (F2P/P2P)	53/53
Task-specific agent-assisted adversarial review of candidate solutions	53/53
Additional causal review for tasks triggering a zero-pass screen	9/9 (all applicable)

REASONS FOR TASK-CANDIDATE NON-ADMISSION. Figure 6 summarizes why 103 task-candidates were not admitted. Of these, 28 had high observed solve rates and fell outside the target difficulty range. An early selection rule, applied only to a subset of task-candidates, also excluded 13 candidates with no observed solves; this did not establish that those tasks were invalid.

Task scope and overlap accounted for further exclusions: 17 candidates were too broad and were replaced by narrower tasks, while nine had source PRs already covered by selected multi-PR tasks.

Execution constraints prevented admission of 14 candidates requiring hardware outside the benchmark's standardized hardware configuration, such as Blackwell GPUs, multiple GPUs, or multiple

nodes. Another 14 candidates could not faithfully execute the accepted implementation or exercise the intended production behavior within the benchmark environment, for example because of incompatible dependencies.

Verification issues accounted for three candidates without reliable fail-to-pass correctness checks, one with an instruction–verifier mismatch, and one with non-reproducible GPU numerical results. Finally, two candidates passed qualification, but their individual reasons for non-admission were not preserved.

A.1.6 RELEASE CANARY

Following the benchmark-canary convention used by BIG-bench and Terminal-Bench (Srivastava et al., 2023; Merrill et al., 2026), SWE-Serve includes a fixed benchmark-level marker in every `task.toml` file and repeats it in `README.md` and `CANARY.md`. The configuration parser ignores the marker because it is a TOML comment, so it does not affect task execution. Upon public release, the marker will help training-corpus builders filter SWE-Serve artifacts and provide a way to identify potential exposure through otherwise unexplained verbatim reproduction.

A.2 Experimental setup

Expanding on Section 4, we provide additional details on execution budgets, model configurations, score aggregation, and closed-book evaluation.

A.2.1 Cross-benchmark evaluation wall-clock time

To contextualize the 210-minute agent execution limit in Section 4, Figure 7 compares task wall-clock times on SWE-Serve and DeepSWE (Huang et al., 2026) using the nine of the 11 top-per-model configurations in Table 3 for which effort-matched trajectories are available in the DeepSWE release. All configurations were evaluated with the mini-SWE-agent harness. Within each benchmark, we first average wall-clock time across all timed trials for each task, then average across tasks. SWE-Serve has a mean task wall-clock time of 43.3 minutes, compared with 34.3 minutes for DeepSWE. Additionally, 24.5% of SWE-Serve tasks have average wall-clock times between one and four hours, compared with 3.5% of DeepSWE tasks. Differences in execution limits, endpoints, hardware, task mix, and verification make this a descriptive comparison.

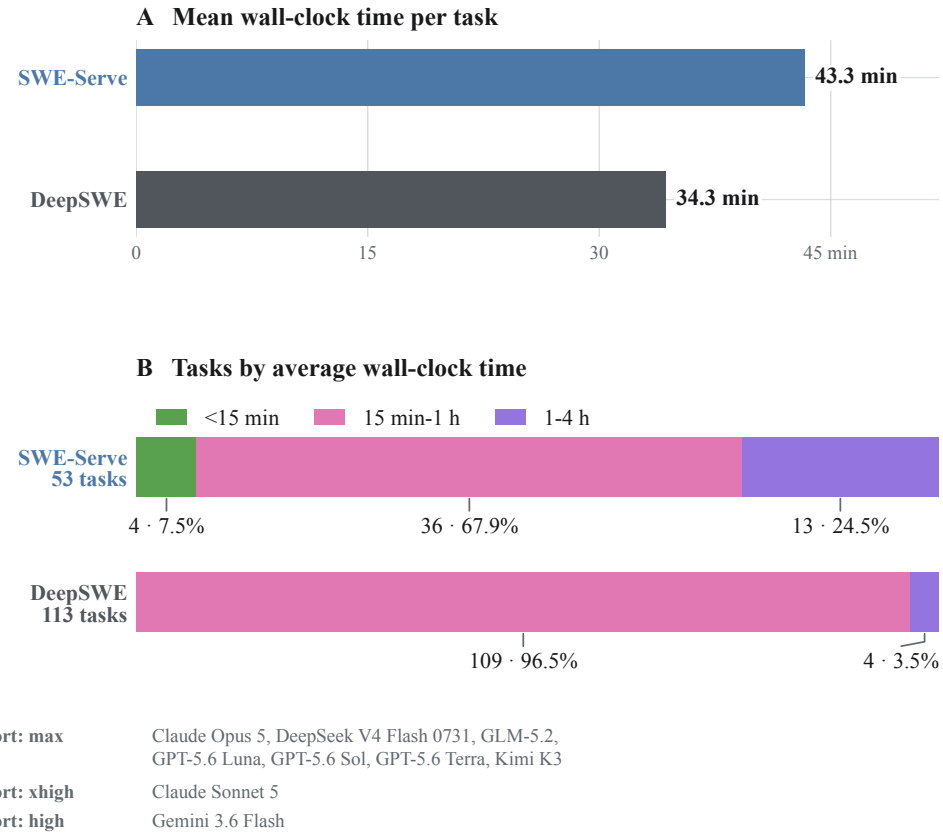


Figure 7: Average task wall-clock time for the nine of the 11 top-per-model configurations with effort-matched trajectories available in the DeepSWE release, all evaluated using mini-SWE-agent. Effort labels denote the resolved route settings. (A) The mean of the task-average wall-clock times. (B) Task counts and percentages of total tasks within each of three wall-clock time intervals. SWE-Serve averages are based on 53 tasks and 1,428 timed trials; DeepSWE averages are based on 113 tasks and 4,060 timed trials. The greater-than-four-hour bucket is omitted because neither benchmark has a task-average wall-clock time above four hours. This is consistent with both evaluations using agent-execution limits below four hours.

A.2.2 ATTEMPT OUTCOMES AND EXECUTION LIMITS

Figure 8 classifies all 1,749 attempts from the 11 top-per-model configurations in Table 3 into nine mutually exclusive outcome categories. Six categories distinguish whether all, some, or no F2P tests passed and whether all P2P tests passed. The remaining three categories capture agent-created patch or agent errors, wall-time limits, and step limits.

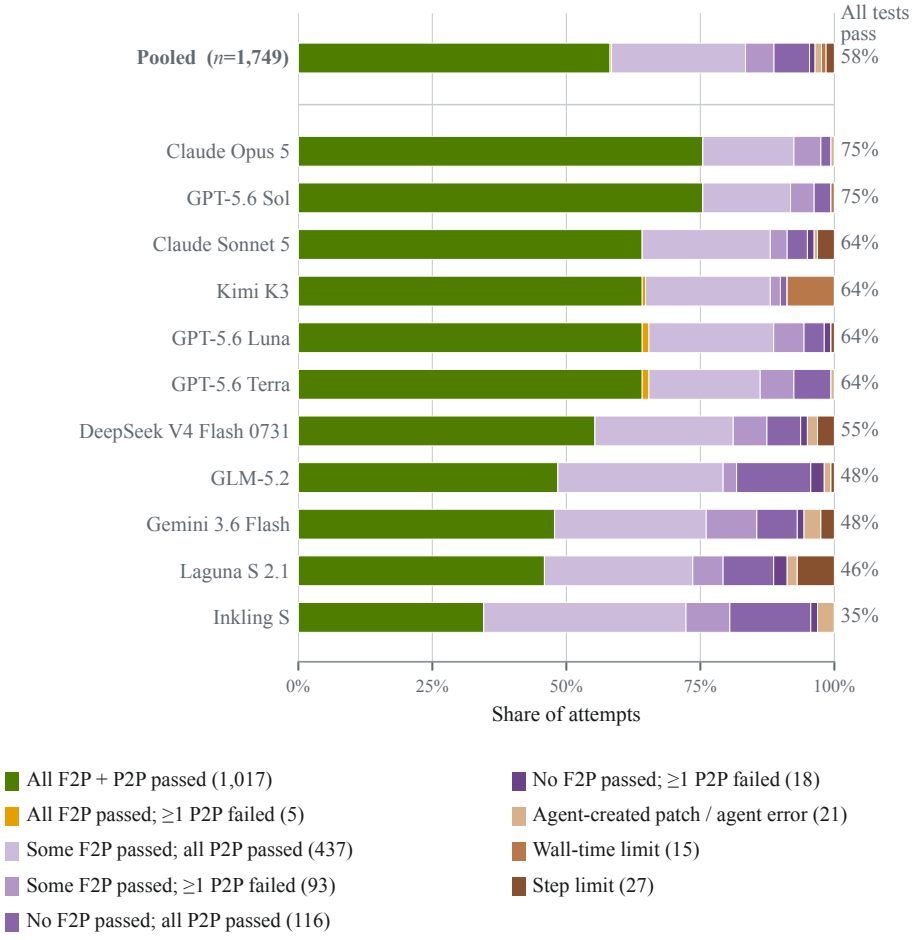


Figure 8: Outcomes for the 1,749 attempts from the 11 top-per-model configurations. The first six categories describe scored outcomes according to how many fail-to-pass (F2P) tests passed and whether all pass-to-pass (P2P) tests passed. "Some F2P" means strictly more than zero and strictly fewer than all F2P checks. Agent-created patch / agent error denotes errors caused by the agent-created patch or the agent that prevented the verifier from producing a pass/fail result. The wall-time and step-limit categories denote unsuccessful attempts reaching the 210-minute or 350-step agent-execution limit; patches that reached these limits and passed verification remain in the "All F2P + P2P passed" category. Model rows follow Table 3, ordered by mean pass@1, with 159 attempts across three runs per model. The right column repeats each model's mean pass@1 from Table 3.

Execution-limit failures account for 42 attempts (2.4%): 15 at the 210-minute wall-time limit and 27 at the 350-step limit. Wall-time limit failures therefore account for only 0.9% of attempts across the 11 top-per-model configurations. These limits constrain agent execution; the resulting patch is still evaluated, and attempts that pass verification remain in the "All F2P + P2P passed" category. To contextualize this low wall-time failure rate, the DeepSWE release's effort-matched results (Huang et al., 2026) for nine of our 11 top-per-model configurations contain 47 wall-time limit failures out of 4,060 attempts (1.2%); three additional timed-out attempts passed verification.

In 21 attempts (1.2%), errors caused by the agent or its patch prevented the verifier from producing a pass/fail result.

Among the remaining 669 scored failures, the most common outcome is passing some F2P checks while preserving all P2P checks: 437 attempts (65.3%). The proportion passing no F2P checks while preserving all P2P checks also tends to be larger for configurations with lower pass@1. Together, these failure patterns indicate that many unsuccessful patches preserve tested existing behavior (P2P tests) while failing to satisfy all of the task's newly required behavior (F2P tests).

A.2.3 Model, serving, and harness configurations

To support reproducibility, Table 7 records the model-serving and evaluation settings for the 11 top-per-model configurations in Table 3, together with the harness versions used in the native-harness sensitivity analysis (Section 5.3 and Appendix A.5). These settings include reasoning effort, API interfaces, context and output-token limits, and inference hardware for self-hosted models. All primary evaluations use mini-SWE-agent v2.4.3.

The reasoning-effort analysis uses the same settings; however, it varies the effort setting across `low`, `medium`, `high`, `xhigh`, and `max` for the two Claude and three GPT-5.6 models.

Table 7: Model-serving and evaluation settings for (a) provider-hosted and (b) self-hosted top-per-model configurations, and (c) the agent-harness sensitivity analysis. Context and output limits are reported separately in tokens; *native* indicates that mini-SWE-agent imposed no token-limit constraints. All evaluations outside the harness-sensitivity analysis use mini-SWE-agent v2.4.3 (Yang et al., 2024). Part (c) applies only to the harness-sensitivity analysis in Section 5.3 and Appendix A.5.

(a) Provider-hosted top-per-model configurations

Model	Reasoning effort	API interface	Context limit (tokens)	Output limit (tokens)
Claude Opus 5	`max`	Messages API	1,000,000	128,000
Claude Sonnet 5	`xhigh`	Messages API	1,000,000	128,000
GPT-5.6 Sol	`max`	Responses API	native	native
GPT-5.6 Luna	`max`	Responses API	native	native
GPT-5.6 Terra	`max`	Responses API	native	native
Gemini 3.6 Flash	`high`	Responses API	1,048,576	65,536

(b) Self-hosted top-per-model configurations

Model	Reasoning effort	Model-serving hardware	Context limit (tokens)	Output limit (tokens)
Kimi K3	`max`	16 GB200	1,048,576	64,000
GLM-5.2 (FP8)	`max`	8 GB200	400,000	131,072
DeepSeek V4 Flash 0731	`max`	4 GB200	1,048,576	384,000
Laguna S 2.1	`max`	4 GB200	1,048,576	131,072
Inkling S	`xhigh`	4 GB200	262,144	64,000

(c) Agent-harness sensitivity analysis

Model	Reasoning effort	Shared harness	Model-specific harness
GPT-5.6 Sol	`max`	mini-SWE-agent v2.4.3	Codex CLI 0.144.5 (OpenAI, 2026a)
Claude Opus 5	`max`	mini-SWE-agent v2.4.3	Claude Code CLI 2.1.241 (Anthropic, 2026a)

A.2.4 LEADERBOARD AGGREGATION

Each of the 11 top-per-model configurations in Table 3 is evaluated in three complete runs of the SWE-Serve benchmark. We calculate pass@1 as the mean of the three run-level pass rates and report a 95% confidence interval using the normal approximation, $\bar{r} \pm 1.96 s_r/\sqrt{3}$, where $\bar{r}$ is the mean pass rate and s_r is the sample standard deviation of the three run-level pass rates. These intervals describe repeated-run variability on the fixed benchmark. Additionally, both the pass@1 and 95% CI percentages in Table 3 are rounded to the nearest percentage point.

A.2.5 CLOSED-BOOK EVALUATION AND RETRIEVAL CALIBRATION

An open-book pilot motivated the closed-book policy in Section 4. The pilot ran without network isolation. It evaluated the SWE-Serve benchmark using mini-SWE-agent with GPT-5.6 Sol, Terra, and Luna under three reasoning-effort settings (`xhigh`, `high`, and `medium`), with one complete benchmark run per model–effort configuration. An *external fetch* is a request to obtain information from a public host outside the task's sandbox. *Confirmed out-of-band retrieval* requires recorded output containing either the task's own source pull-request content or an upstream file modified by its oracle patch. Table 8 reports the percentage of tasks exhibiting each behavior for each model and effort setting. The *Any model* column reports the task-level union at that effort setting: a task is counted once if at least one of the three model attempts exhibits the behavior specified by the row.

Table 8: External fetches and confirmed out-of-band retrieval by agents in the open-book pilot. External fetches (requests to public hosts) and confirmed out-of-band retrieval (recorded output containing the task's source pull-request content or an upstream file modified by its oracle patch) are reported as percentages of SWE-Serve tasks for each model at three reasoning-effort levels: `xhigh`, `high`, and `medium`. The final *Any model* column reports the percentage of tasks exhibiting each behavior in at least one of the three model attempts at the given effort level, counting each task once.

Reasoning effort	Sol	Terra	Luna	**Any model**
Any external fetch				
`xhigh`	41.5%	37.7%	34.0%	**54.7%**
`high`	30.2%	11.3%	13.2%	**34.0%**
`medium`	15.1%	13.2%	3.8%	**17.0%**
Confirmed out-of-band retrieval				
`xhigh`	13.2%	17.0%	15.1%	**32.1%**
`high`	15.1%	5.7%	5.7%	**15.1%**
`medium`	3.8%	3.8%	0.0%	**3.8%**

Out-of-band access is not isolated to one model or reasoning setting: all nine benchmark runs across different model–effort configurations contain external fetches, and all three models exhibit confirmed out-of-band retrieval. Incidence of both external fetches and out-of-band retrieval generally increases with reasoning effort.

All other SWE-Serve evaluations enforced network isolation.

A.3 AGENTIC PERFORMANCE AND RESOURCE TRADE-OFFS

This section extends Section 5.1 with results for all 31 model–effort configurations, resource trade-offs, and performance by engineering family.

A.3.1 PERFORMANCE ACROSS 31 MODEL–EFFORT CONFIGURATIONS

Table 9 extends the primary leaderboard in Table 3 to show both performance and consistency across all 31 model–effort configurations. Alongside mean pass@1, it reports pass@3 and passˆ3; the gap between these measures is the fraction of tasks solved in some but not all three attempts. The table includes five reasoning-effort settings for each of the two Claude and three GPT-5.6 models, and one configuration for each remaining model.

Table 9: Mean pass@1, pass@3, and passˆ3 for all 31 model–effort configurations, ordered by mean pass@1. Each configuration was evaluated with three complete runs of SWE-Serve. The pass@3 value is the fraction of tasks solved in at least one of the three runs; passˆ3 is the fraction solved in all three runs.

Configuration				
Model	Reasoning effort	Mean pass@1	pass@3	passˆ3
GPT-5.6 Sol	max	75.5%	83.0%	67.9%
Claude Opus 5	max	75.5%	81.1%	69.8%
Claude Opus 5	high	73.6%	79.2%	66.0%
Claude Opus 5	xhigh	72.3%	81.1%	62.3%
Claude Opus 5	medium	68.6%	77.4%	58.5%
GPT-5.6 Sol	xhigh	66.0%	75.5%	52.8%
GPT-5.6 Terra	max	64.2%	77.4%	54.7%
Claude Sonnet 5	xhigh	64.2%	75.5%	50.9%
Kimi K3	max	64.2%	75.5%	50.9%
GPT-5.6 Luna	max	64.2%	73.6%	50.9%
Claude Sonnet 5	max	61.0%	71.7%	50.9%
Claude Opus 5	low	60.4%	67.9%	54.7%
Claude Sonnet 5	high	57.9%	69.8%	43.4%
GPT-5.6 Sol	high	56.0%	69.8%	43.4%
DeepSeek V4 Flash (0731)	max	55.3%	73.6%	41.5%
Claude Sonnet 5	medium	55.3%	69.8%	39.6%
GPT-5.6 Luna	xhigh	55.3%	64.2%	45.3%
GPT-5.6 Sol	medium	54.1%	66.0%	37.7%
GPT-5.6 Terra	xhigh	52.8%	64.2%	43.4%
GPT-5.6 Terra	high	49.1%	62.3%	34.0%
Claude Sonnet 5	low	49.1%	60.4%	35.8%
GLM-5.2	max	48.4%	64.2%	34.0%
Gemini 3.6 Flash	high	47.8%	60.4%	30.2%
Laguna S 2.1	max	45.9%	64.2%	26.4%
GPT-5.6 Luna	high	45.3%	56.6%	35.8%
GPT-5.6 Terra	medium	44.7%	54.7%	34.0%
GPT-5.6 Terra	low	44.0%	54.7%	35.8%
GPT-5.6 Sol	low	40.9%	52.8%	28.3%
GPT-5.6 Luna	medium	37.7%	49.1%	24.5%
Inkling S	xhigh	34.6%	45.3%	24.5%
GPT-5.6 Luna	low	20.8%	28.3%	11.3%

A.3.2 PERFORMANCE AND RESOURCE TRADE-OFFS

Figure 9 compares mean pass@1 and inference cost across reasoning-effort settings. Higher effort consistently increases cost, while performance gains vary across models and effort settings.

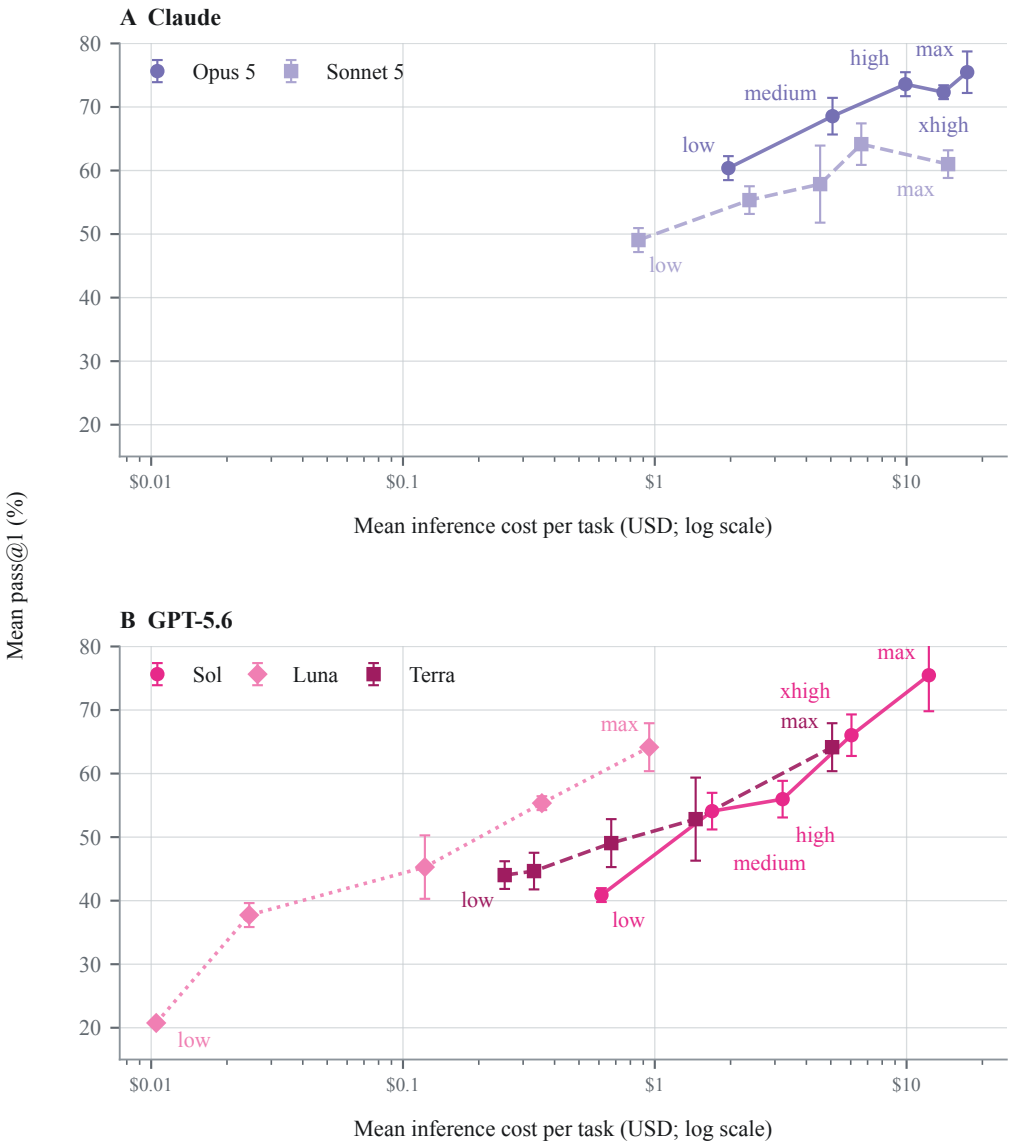


Figure 9: Mean pass@1 versus mean inference cost per task across reasoning-effort sweeps for (A) Claude Opus 5 and Claude Sonnet 5 and (B) GPT-5.6 Sol, Luna, and Terra. Axes are shared between (A) and (B); the cost axis is logarithmic. Vertical bars show one sample standard deviation across the three runs of SWE-Serve for each model–effort configuration. All five sweeps include `low`, `medium`, `high`, `xhigh`, and `max` reasoning effort. Higher reasoning effort is associated with higher mean inference cost, but not consistently with higher mean pass@1.

Figures 10–12 compare the 11 top-per-model configurations in Table 3, plotting mean pass@1 against mean inference cost, output-token use, and wall-clock time per task, respectively. Wall-clock time measures the end-to-end duration of a task's attempt, including setup, agent execution, and verification. Each plot shows the observed Pareto frontier among these selected configurations. Even under a shared agent harness, configurations with similar pass@1 can differ substantially in resource use, so configuration choice depends on the relative importance of performance, cost, output-token use, and latency.

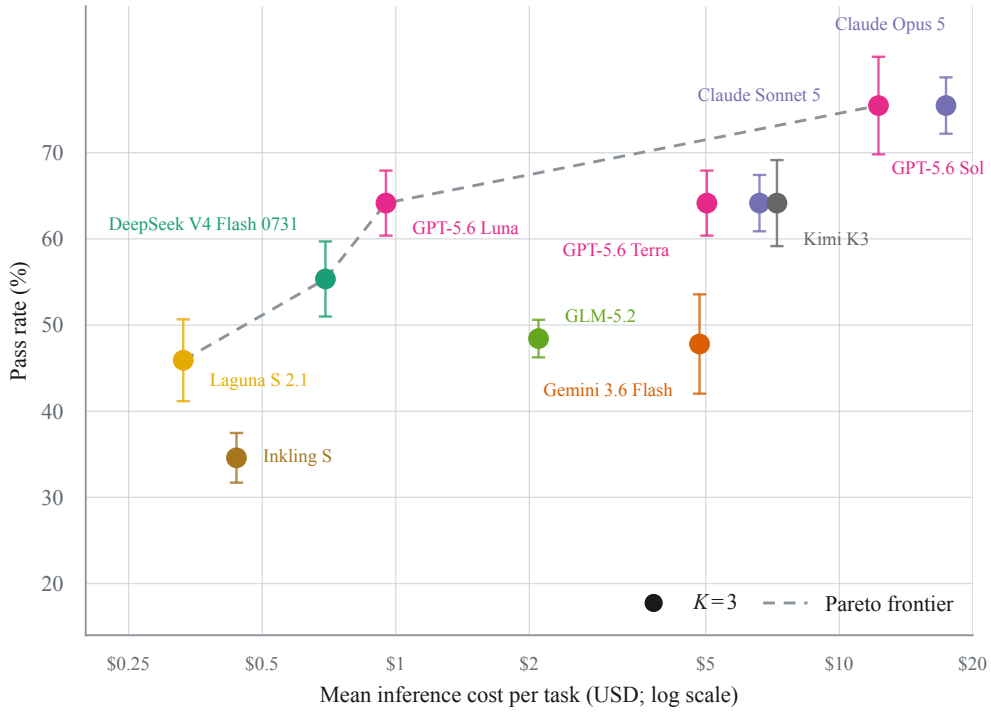


Figure 10: Mean pass@1 versus mean inference cost per task for the 11 top-per-model configurations. Vertical bars show one sample standard deviation across the three runs of SWE-Serve for each configuration. The cost axis is logarithmic. The dashed line connects the observed Pareto frontier for cost among these configurations. Higher mean inference cost is not consistently associated with higher mean pass@1, and configurations with similar mean pass@1 can differ in inference cost.

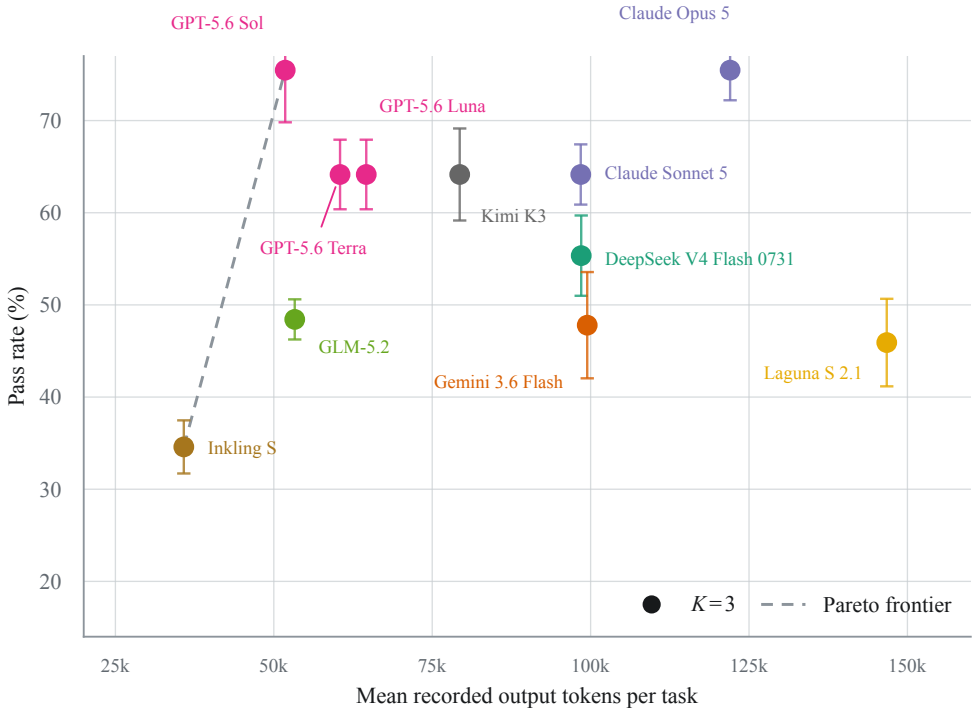


Figure 11: Mean pass@1 versus mean output tokens per task for the 11 top-per-model configurations. Vertical bars show one sample standard deviation across the three runs of SWE-Serve for each configuration. The dashed line connects the observed Pareto frontier for output-token use among these configurations. GPT-5.6 Terra's token count is based on 158 of 159 attempts as 1 value was missing and missing values are not extrapolated; all other configurations have complete token accounting. Higher mean output-token use is not consistently associated with higher pass@1: GPT-5.6 Sol matches Claude Opus 5's mean pass@1 while using fewer than half as many output tokens per task.

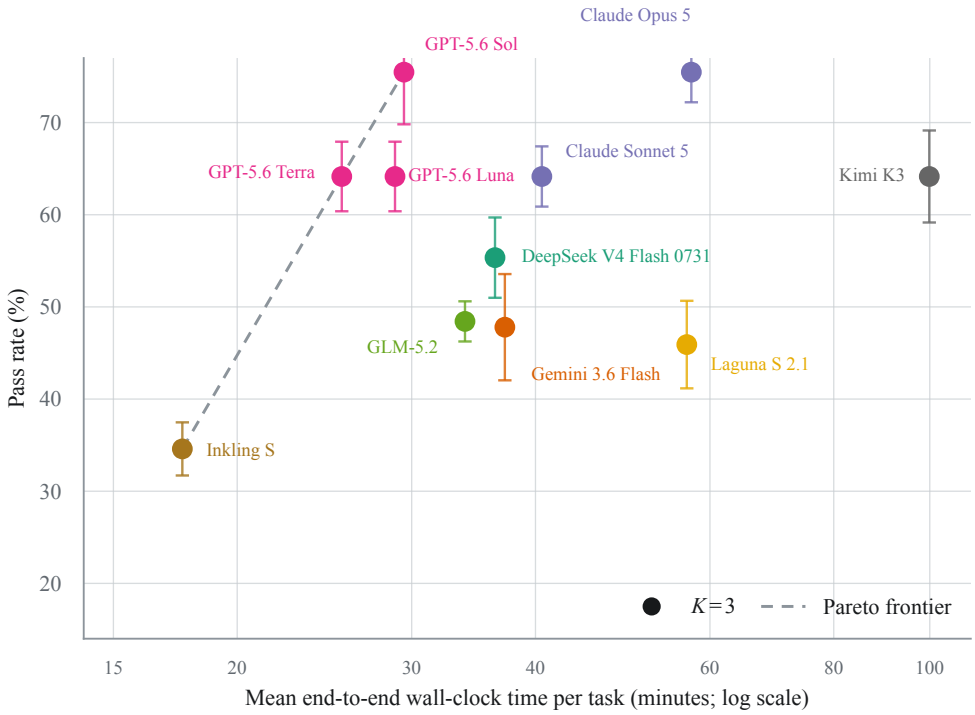


Figure 12: Mean pass@1 versus mean wall-clock time per task for the 11 top-per-model configurations, including setup, agent execution, and verification. Vertical bars show one sample standard deviation across the three runs of SWE-Serve for each configuration. The dashed line connects the observed Pareto frontier for wall-clock time among these configurations. Similar pass@1 can be associated with different latency: among the four configurations at approximately 64% pass@1, mean wall-clock time varies by 3.9×.

A.3.3 PERFORMANCE BY INFERENCE ENGINEERING FAMILY

Figure 13 reports pass@1 across the six inference engineering families for the 11 top-per-model configurations. Averaged across these configurations, pass@1 ranges from 42.9% for model and backend enablement to 74.6% for kernels, quantization, and performance. Claude Opus 5 and GPT-5.6 Sol, the two configurations with the highest overall mean pass@1, collectively achieve the highest pass@1 in five of the six families. Claude Sonnet 5 and GPT-5.6 Terra tie these leaders in model/backend enablement and caching/runtime state, respectively, while GPT-5.6 Luna leads in distributed/scheduling tasks.

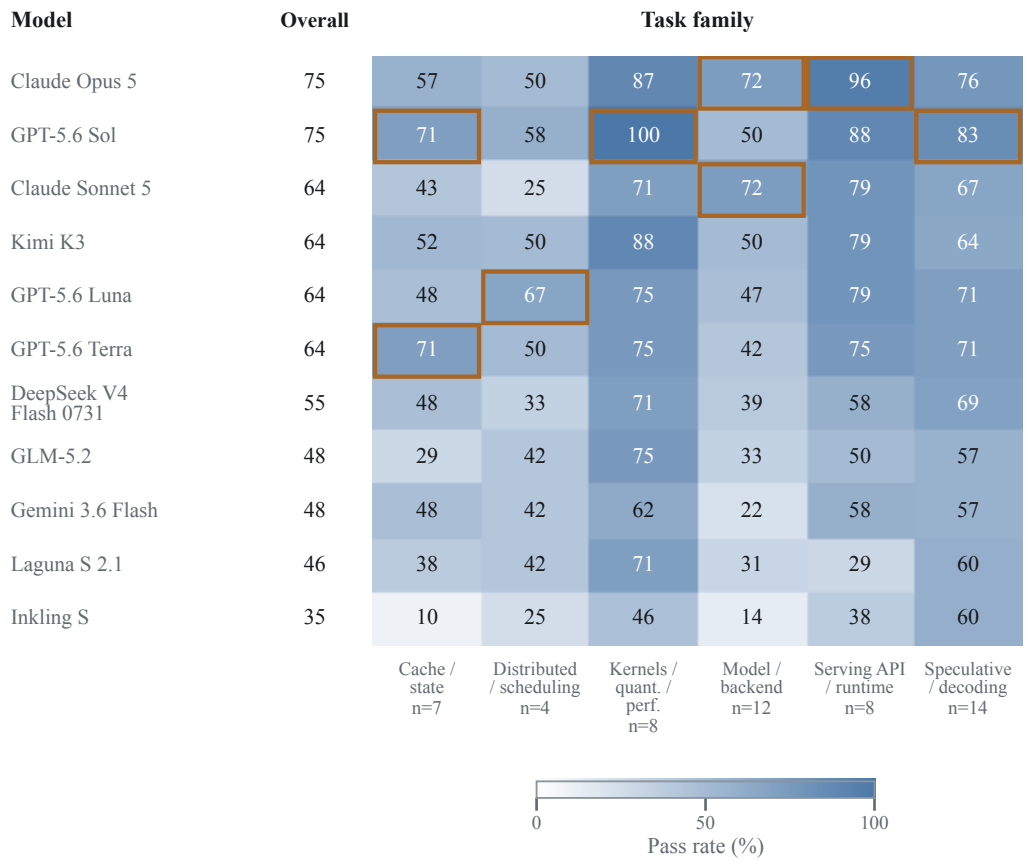


Figure 13: Engineering-family pass@1 for the 11 top-per-model configurations, ordered by mean pass@1 to match Table 3. Scores are averaged across three SWE-Serve runs for each configuration. Task counts appear below the engineering-family column labels. Orange outlines mark the highest observed pass@1 in each family, including ties. The Overall column reports each configuration's mean pass@1 across the entire SWE-Serve benchmark. The Claude Opus 5 and GPT-5.6 Sol configurations have the highest mean pass@1 across all tasks and collectively attain the highest mean pass@1 in five of the six task families, including ties. GPT-5.6 Luna has the highest mean pass@1 in distributed/scheduling tasks.

A.4 PRODUCTION CORRECTNESS

The following analyses expand the production correctness results in Section 5.2 and Figure 2, including task-level and per-configuration breakdowns, sensitivity to which tests are removed and which tasks are included, and comparisons of production correctness gaps across reasoning-effort settings.

A.4.1 LOCAL CORRECTNESS DOES NOT ENSURE PRODUCTION CORRECTNESS

Model-serving E2E gap. Table 10 provides the task-level test counts and patch outcomes behind the model-serving E2E gap in Figure 2A. It covers all 627 agent-created patches from the 11 top-per-model configurations on the 19 tasks that contain model-serving E2E tests. Of the 339 unsuccessful attempts under the full verifier, removing model-serving E2E tests from scoring changes 147 patches from fail to pass, while 178 remain failures. These 147 fail-to-pass transitions out of 627 attempts produce the 23.4-percentage-point increase in pass rate across the 19 tasks. The 14 attempts without a verifier pass/fail result remain counted as failures. This analysis rescores recorded test results without rerunning the verifier.

Figure 14A breaks down the aggregate model-serving E2E gap across the 11 top-per-model configurations in Figure 2A by configuration. For each of the configurations, it reports the increase in pass rate when the same patches on the 19 tasks are rescored without model-serving E2E tests. Pass rate increases for every configuration, and the magnitude varies from 3.5 to 45.6 percentage points.

Matched test-removal control. To test whether removing equal numbers of non-E2E tests produces a comparable effect to removing model-serving E2E tests, we compare both conditions on 12 eligible tasks and their 396 patches. These 12 tasks are eligible because we can remove matching numbers of E2E and non-E2E tests within the same F2P/P2P role. Of the 19 tasks with E2E tests, four have no non-E2E tests, and three others have both test types but never within the same role. Those seven tasks are therefore excluded.

For each task, we compare two versions of its score: one with some E2E tests excluded and another with the same number of non-E2E tests excluded. We match the removal counts separately for F2P and P2P tests, using the smaller available count. For example, if a task has two E2E F2P tests and five non-E2E F2P tests, we exclude both E2E tests in the first condition and two randomly selected non-E2E tests in the second. We apply the same rule to P2P tests. Across all 12 tasks, each scoring condition excludes 38 tests: 19 F2P and 19 P2P.

We repeat the comparison 10,000 times, sampling tests without replacement separately for each task, test type, and F2P/P2P role, using NumPy's PCG64 random-number generator with seed 20260826 (Harris et al., 2020). The tasks, patches, and removal counts remain fixed; only the excluded tests vary.

Sensitivity to task composition. To assess how the E2E pass-rate increase depends on which tasks are included, we resample the 19 tasks with model-serving E2E tests with replacement 10,000 times. We use NumPy's PCG64 random-number generator with a fixed seed of 20260825 for reproducibility (Harris et al., 2020). Each sampled task retains all 33 agent-created patches, comprising the 11 top-per-model configurations and their three attempts per task, so neither models nor attempts are independently resampled. We report a 95% percentile bootstrap confidence interval.

Sensitivity to tasks containing only E2E tests. To assess whether the E2E pass-rate increase persists when other tests remain in scoring, we exclude the four tasks containing only E2E tests (Table 10). Across the remaining 15 tasks, pass rate still rises from 191/495 (38.6%) to 304/495 (61.4%), yielding a 22.8-percentage-point increase, corresponding to 113 fail-to-pass transitions.

Production inference task properties. Figure 14B–D breaks down the aggregate production correctness gaps in Figure 2B–D by model configuration. The runtime-domain gap is consistent across all 11 top-per-model configurations: pass rates are 10.0–39.6 percentage points lower on tasks spanning multiple runtime domains than on single-domain tasks (Figure 14B).

Table 10: Task-level effects of removing model-serving E2E tests from scoring across 19 tasks with E2E tests. Each task has 33 attempts: three from each of the 11 top-per-model configurations. Test counts include only tests used for scoring. Outcome columns distinguish attempts that pass all tests, change from fail to pass after E2E removal, still fail after removal, or have no verifier pass/fail result. Attempts without a result remain failures. The first 15 tasks retain non-E2E tests after removal; the final four have no tests left in scoring.

	Test counts		Number of attempts			
19 tasks with model-serving E2E tests	# E2E tests	# Non-E2E tests	Pass with all tests	Fail → pass without E2E	Still fail without E2E	No pass/fail result
Tasks with non-E2E tests (15)						
Spec: forward-timeout ordering fix (EAGLE v1)	1	13	31	0	1	1
DFLASH v1 non-overlap serving	34	3	0	0	32	1
EAGLE v2 page-one tree runtime	19	2	6	25	2	0
Diffusion-LLM radix-graph serving	8	8	0	2	31	0
SDAR dense + MoE day-0 serving	2	3	10	17	6	0
Qwen3.5 dense + MoE core serving	16	5	9	16	8	0
Transformers-backend compute runtime	3	2,277	4	0	28	1
Transformers 5 MoE core serving	4	1	21	7	0	5
Gemma 4 speculative execution	4	1	26	7	0	0
Gemma 4 26B-A4B core serving	3	2	16	16	1	0
N-gram trie refactor + stateful matching	4	34	28	0	3	2
N-gram external SAM corpus	2	22	8	5	20	0
SequenceClassification Score API	3	37	14	0	16	3
Spec-V2 streaming-session KV commit	32	15	3	0	30	0
EAGLE3 overlap metadata lifecycle	110	7	15	18	0	0
Tasks without non-E2E tests (4)						
Mixed-chunk piecewise CUDA-graph replay	3	0	15	18	0	0
Gemma 4 piecewise CUDA-graph execution	6	0	29	4	0	0
GLM-V repetition-penalty sampling	13	0	27	6	0	0
Score API hybrid token + embedding inputs	9	0	26	6	0	1
Total	276	2,430	288	147	178	14

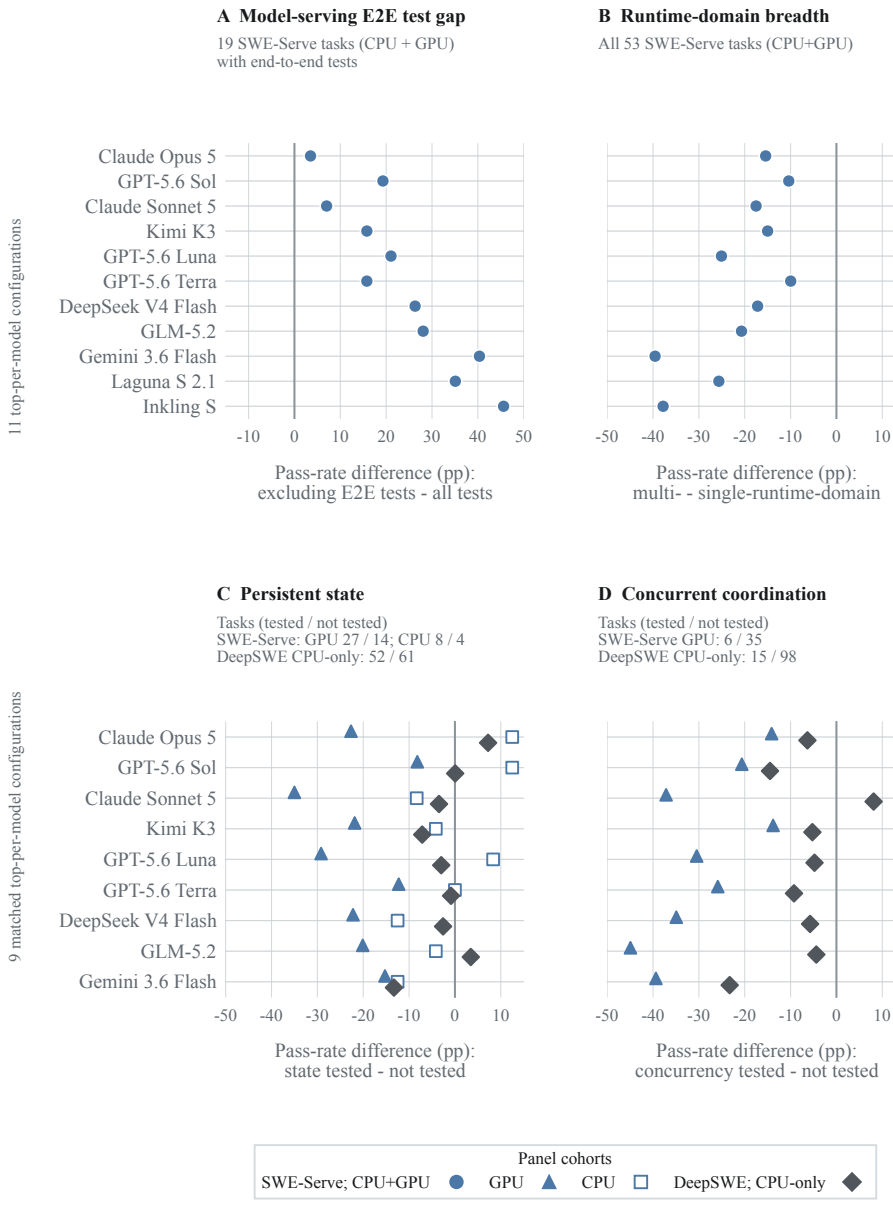


Figure 14: Production correctness gaps across top-per-model configurations. (A) Rescoring the same patches without model-serving E2E tests raises pass rate by 3.5–45.6 percentage points across the 11 top-per-model configurations. (B) Multi-runtime-domain tasks have pass rates 10.0–39.6 percentage points lower than single-runtime-domain tasks across the 11 top-per-model configurations. (C–D) Across nine configurations with effort-matched trajectories available in the DeepSWE release, SWE-Serve GPU tasks tested for persistent state or concurrent coordination have pass rates 8.2–35.0 and 13.8–44.9 percentage points lower, respectively, than tasks not tested for these properties. CPU comparisons generally show smaller differences and vary in direction: panel C includes SWE-Serve CPU and DeepSWE, while panel D includes DeepSWE only (Huang et al., 2026). Panel D omits SWE-Serve CPU tasks because none are tested for concurrent coordination. Positive values in A indicate increased pass rates after removing E2E tests; negative values in B–D indicate lower pass rates on tasks with the specified production property. Configurations are ordered by overall mean pass@1 to match Table 3.

For persistent state and concurrent coordination, Figure 14C–D shows results for each of the same nine top-per-model configurations used in Figure 2C–D, evaluated at matching reasoning-effort settings on SWE-Serve and DeepSWE (Huang et al., 2026). SWE-Serve GPU tasks testing persistent state or concurrent coordination have lower pass rates than tasks where the corresponding property is not tested, across all nine configurations. The corresponding differences on SWE-Serve CPU and DeepSWE are generally smaller in magnitude and are not consistently negative across configurations.

A.4.2 Reasoning-effort robustness of production correctness gaps

Figure 15 examines whether higher reasoning effort narrows production correctness gaps for GPT-5.6 Sol, Luna, and Terra across five settings: `low`, `medium`, `high`, `xhigh`, and `max`. It assesses all four production correctness gaps using the same task groups as Figures 2 and 14. The E2E comparison rescores the same patches with and without E2E tests; the other three comparisons measure pass-rate differences between task groups.

All four gaps remain at `max` reasoning effort. At this setting, the model-serving E2E gap is 16–21 percentage points, and pass rates on multi-runtime-domain tasks are 10–25 points below those on single-domain tasks. Also at `max` reasoning effort, SWE-Serve GPU tasks testing persistent state or concurrent coordination have pass rates 8–29 and 21–30 points lower, respectively, than tasks where the corresponding property is not tested.

The response to increasing effort differs across the four comparisons. From `low` to `max` reasoning effort, the E2E and runtime-domain gaps narrow for all three models. The persistent-state gap narrows for Sol and Terra but widens for Luna, while the concurrent-coordination gap narrows for Sol but widens for Luna and Terra. Even where gaps narrow overall, they do not always shrink at each successive effort setting.

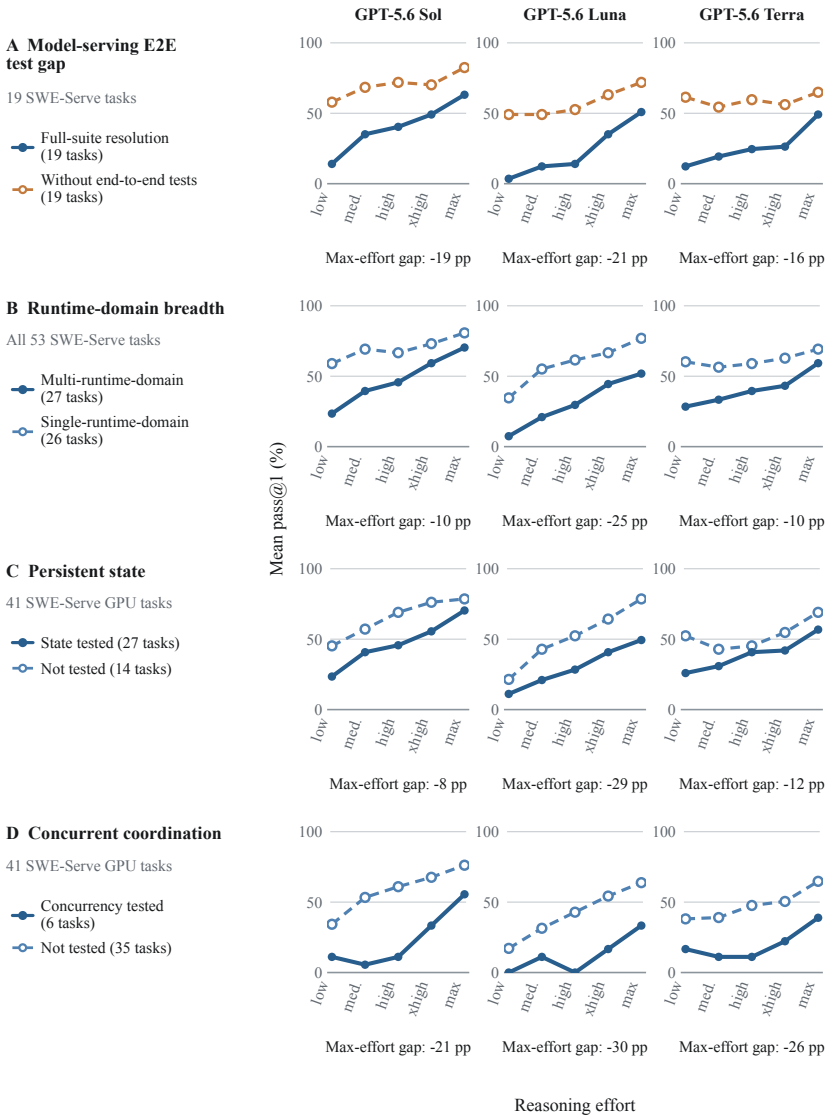


Figure 15: Higher reasoning effort narrows but does not eliminate the model-serving E2E gap across GPT-5.6 Sol, Luna, and Terra; task-property gaps also persist at maximum effort. Panels A–D show the model-serving E2E test gap, runtime-domain breadth, persistent state, and concurrent coordination across five reasoning-effort settings, with three attempts per task for each configuration. Dark blue solid lines with filled markers show full-suite pass rates (A), multi-runtime-domain tasks (B), and tasks tested for the specified property (C–D). Lighter blue dashed lines with hollow markers show single-runtime-domain tasks (B) or tasks not tested for that property (C–D). Orange dashed lines in A show pass rates after removing model-serving E2E tests from scoring while keeping the same patches. Each "Max-effort gap" is the dark blue rate minus its comparison rate at `max` reasoning effort, in percentage points.

A.5 SENSITIVITY TO AGENT HARNESS

This section expands the agent-harness sensitivity analysis in Section 5.3 with details of the analysis and how agent scaffold choice affects resource use. We compare mini-SWE-agent with the model-specific harnesses for GPT-5.6 Sol and Claude Opus 5, the two best-performing configurations, both at maximum reasoning effort. Figure 16 reports mean inference cost and wall-clock time per task, while Figure 17 breaks down mean token use into uncached input, cached input, and output tokens.



Figure 16: Comparing mean pass@1, cost, and wall-clock time for the two best-performing configurations, GPT-5.6 Sol and Claude Opus 5, when using their model-specific harness compared to the mini-SWE-agent harness. Each data point is based on three complete runs of SWE-Serve. Direct labels report each model-specific harness's change relative to mini-SWE-agent. Hollow points show the means for mini-SWE-agent, and filled points show the means for the model-specific harnesses. Costs for mini-SWE-agent and Claude Code are exact recorded endpoint costs, while Codex costs are estimated with approximately 1–2% precision. For both evaluated configurations, mean wall-clock time is lower, inference cost is higher, and observed mean pass@1 is lower with the model-specific harness than with mini-SWE-agent.

We evaluated GPT-5.6 Sol and Claude Opus 5, both at maximum reasoning effort, with three complete runs of SWE-Serve under their model-specific harnesses: Codex CLI 0.144.5 (OpenAI, 2026a) for GPT-5.6 Sol and Claude Code CLI 2.1.241 (Anthropic, 2026a) for Claude Opus 5. Using these harnesses changes the full agent scaffold, including the instructions, available tools, and how the agent organizes and completes its work. Whereas mini-SWE-agent performs repository inspection, editing, and testing through its only available tool, Bash, Claude Code and Codex expose broader tool sets. Claude Code, for example, provides dedicated tools for searching, reading, and editing files.

Both model-specific harnesses used the same SWE-Serve task and verifier setup, hardware allocation, closed-book policy, and 210-minute wall-time limit. The 350-step limit applied to mini-SWE-agent was not imposed on the model-specific harnesses.

For both evaluated configurations, GPT-5.6 Sol and Claude Opus 5, the model-specific harness has lower mean wall-clock time but higher mean inference cost than mini-SWE-agent (Figure 16), with lower observed mean pass@1, as reported in Section 5.3. For GPT-5.6 Sol, Codex has lower mean uncached input-token use but higher mean output-token use; Claude Code shows the opposite pattern for Claude Opus 5 (Figure 17). Faster execution is not necessarily associated with higher pass@1, lower inference cost, or fewer output tokens. For the two evaluated configurations, these results support using mini-SWE-agent for the primary leaderboard and show that measured performance and resource use differ across harnesses.



Figure 17: Mean per-task token use for GPT-5.6 Sol and Claude Opus 5, the two best-performing configurations, comparing their model-specific harnesses with mini-SWE-agent. Token use is separated into uncached input, cached input, and generated output tokens. Direct labels report each model-specific harness's relative change from mini-SWE-agent. Hollow points show means for mini-SWE-agent, and filled points show means for the model-specific harnesses. Relative to mini-SWE-agent, both model-specific harnesses have higher mean cached input-token use, but their other token-use changes differ: Codex has lower mean uncached input-token use and higher mean output-token use, while Claude Code shows the opposite pattern.

A.6 LIMITATIONS OF EXECUTABLE SCORING

As discussed in Section 6, passing the verifier does not establish every dimension of upstream readiness. Comparing an agent-created patch with the oracle patch also does not establish upstream readiness: the oracle is one reference implementation, and alternative correct solutions may differ in scope without being lower quality. However, because oracle patches derive from merged upstream changes, they provide a useful reference for examining implementation scope beyond executable scores. Figure 18 and Table 11 compare agent-created patches with oracle patch scope as coarse indicators of implementation complexity. Figure 18 visualizes missing scope, while Table 11 reports exact median measurements of both missing and extra scope relative to the oracle patches. These measurements can highlight differences for further review, but do not assess maintainability or architectural fit, nor replace maintainer review.

Before examining verifier outcomes, we selected for analysis the first run for each of the 11 top-per-model configurations and considered all task attempts from that run. Figure 18 and Table 11 include all attempts with both a usable patch capture and a verifier pass/fail result. This excludes nine attempts: one has an unusable patch capture, and eight encountered agent-created patch or agent errors that prevented the verifier from producing a pass/fail result. These exclusions apply to individual attempts rather than removing the corresponding tasks across all configurations.

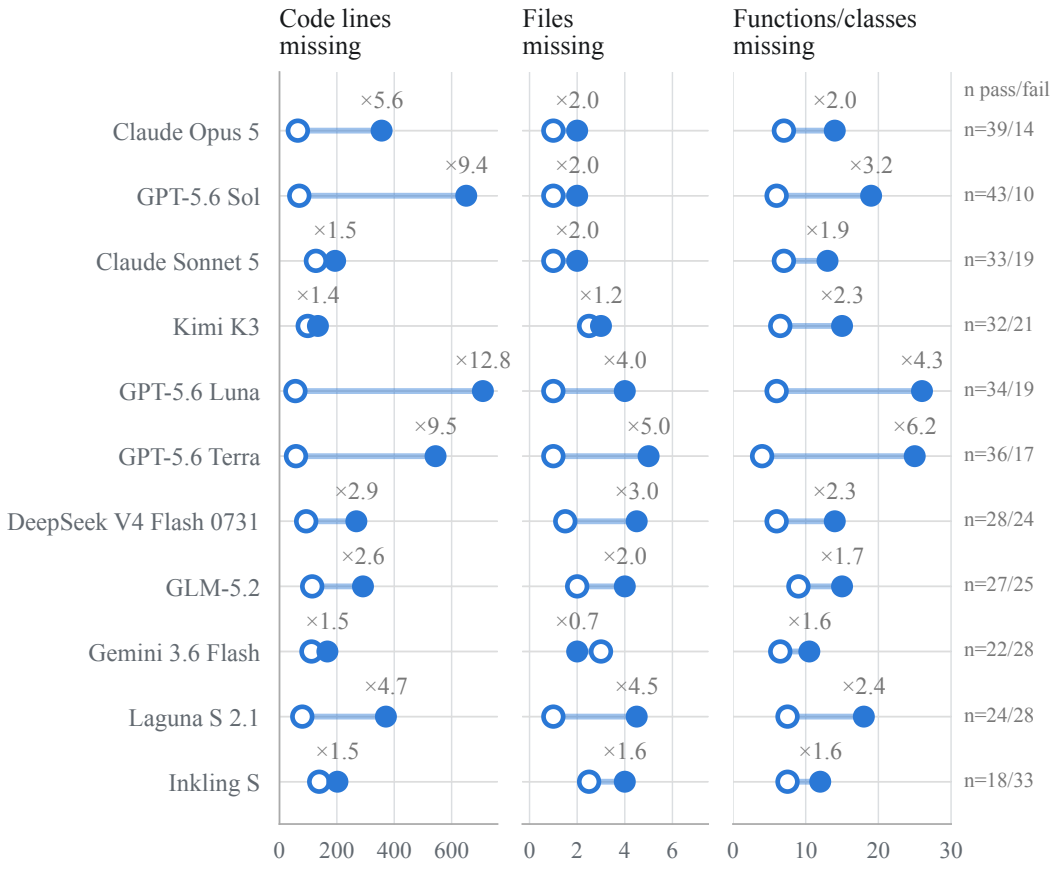


Figure 18: Missing scope differences relative to oracle patches for passing and failing agent-created patches from the first SWE-Serve run attempts of the 11 top-per-model configurations. Median missing scope is reported relative to the oracle patch in code lines, files, and estimated changed functions/classes. Hollow points show medians for passing attempts and filled points show medians for failed attempts. × reports the ratio of failed-attempt to passing-attempt medians for each scope measurement. n reports the number of usable patches (passing/failing) for each of the 11 top-per-model configurations. Configurations are ordered by overall mean pass@1 to match Table 3; scope measurements use only the first-run attempts, a subset of the attempts used to calculate that mean.

We align agent-created and oracle patches by file path and measure missing and extra scope relative to the oracle patch in production code lines, files, and estimated changed functions/classes. We estimate how many functions and classes each patch changes in two ways. First, we identify function and class names in diff headers, which can identify the function or class containing a change even when its declaration is unchanged. Second, we identify added or removed lines that declare a function or class. Within each file, each distinct function name and each distinct class name is counted once, even if it appears in multiple changes. Missing and extra scope measure differences in counts rather than identifying specific missing functionality. We then calculate median measurements separately for passing and failing attempts within each configuration. For every configuration, failed attempts have a higher median missing-line count than passing attempts. However, these groups contain different tasks, so the comparison does not isolate the relationship between patch scope and verifier outcome on the same tasks.

Table 11: Patch scope differences relative to oracle patches for passing and failing agent-created patches from the first SWE-Serve run attempts of the 11 top-per-model configurations. Median missing and extra scope is reported relative to the oracle patch in code lines, files, and estimated changed functions/classes; n reports the number of usable patches (passing/failing) for each of the 11 top-per-model configurations. Configurations are ordered by overall mean pass@1 to match Table 3; scope measurements use only the first-run attempts, a subset of the attempts used to calculate that mean.

Model	Outcome	n	Code lines		Files		Functions/classes	
			Missing	Extra	Missing	Extra	Missing	Extra
Claude Opus 5	Pass	39	64	82	1	0	7	7
	Fail	14	356	156.5	2	2	14	15.5
GPT-5.6 Sol	Pass	43	69	54	1	1	6	2
	Fail	10	651	58.5	2	0.5	19	5
Claude Sonnet 5	Pass	33	127	31	1	0	7	1
	Fail	19	194	28	2	0	13	1
Kimi K3	Pass	32	98.5	39	2.5	0	6.5	2
	Fail	21	134	114	3	1	15	4
GPT-5.6 Luna	Pass	34	55.5	53.5	1	0.5	6	2.5
	Fail	19	709	196	4	1	26	8
GPT-5.6 Terra	Pass	36	57.5	75	1	1	4	3.5
	Fail	17	544	175	5	1	25	9
DeepSeek V4 Flash (0731)	Pass	28	93	32.5	1.5	0	6	1
	Fail	24	268	37.5	4.5	0	14	2
GLM-5.2	Pass	27	114	21	2	0	9	0
	Fail	25	291	21	4	0	15	0
Gemini 3.6 Flash	Pass	22	111.5	26	3	0	6.5	0
	Fail	28	167	43	2	0.5	10.5	1
Laguna S 2.1	Pass	24	79.5	29	1	0	7.5	1
	Fail	28	371	43.5	4.5	1	18	3
Inkling S	Pass	18	138.5	4.5	2.5	0	7.5	0
	Fail	33	202	4	4	0	12	1