ACCELMARK-AGENT: BENCHMARKING INFERENCE-OPTIMIZATION AGENTS ACROSS ACCELERATORS AND WORKLOADS

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimizing an LLM serving deployment requires changes whose benefits depend on the workload, accelerator and starting configuration. Coding agents can search this space, but a high measured throughput alone does not establish that they delivered a valid improvement. We introduce ACCELMARK-AGENT, a benchmark for optimizing complete serving deployments under a controller that validates candidates, retains failed rounds and independently re-measures the final artifact after accuracy checking. The benchmark covers five workloads, five backbone models and two agent harnesses, with a fully crossed A100 matrix of $2{,}000$ search rounds and additional tracks on RTX 4090, V100 and Ascend 910C. In the A100 matrix, 26 of 50 cells delivered a changed artifact more than 2% faster than baseline. Failure to produce a measurable candidate accounted for much of the variation across agent setups. Workload medians ranged from -0.02% on standard decoding to $+12.11\%$ on multi-precision inference. The largest gains, $+211.72\%$ and $+137.82\%$, followed changes to sliding-window KV-cache allocation and admission; most searches did not find this combination. Additional studies show why these gains require context: one repeated search differed by eleven percentage points, and no Ascend condition produced a valid final delivery. ACCELMARK-AGENT reports the deployment, starting throughput, search outcomes and validated artifact together, making both improvements and failures available for comparison.

1 INTRODUCTION

Efficient LLM serving depends on decisions across the deployment: the engine and attention backend, compilation policy, KV-cache layout, scheduling and host-side processing (Kwon et al., 2023; Zheng et al., 2024; Agrawal et al., 2024; Ansel et al., 2024). These decisions interact. For example, increasing the KV-cache budget may have little effect while a scheduler overestimates the cache required by each request, yet substantially increase throughput once that estimate is corrected. The useful change also depends on the accelerator and workload. Finding it repeatedly across deployments requires expertise and offers a measurable objective for coding agents.

Agent benchmarks now cover repository repair (Jimenez et al., 2024; Yang et al., 2024; Wang et al., 2025), machine-learning engineering (Chan et al., 2025; Wijk et al., 2024) and performance optimization (Ouyang et al., 2025; Shetty et al., 2026; He et al., 2025). Within serving systems, ISO-Bench (Nangia et al., 2026) derives patch-level tasks from vLLM and SGLang changes, while InferenceBench (Yeon et al., 2026) evaluates complete server deployment and optimization. These benchmarks establish useful evaluation settings. A remaining question is how agent outcomes vary across workloads, accelerators and harnesses when the search and final measurement follow a common protocol.

Answering this question requires both environmental coverage and reliable measurement. A speedup from one starting configuration may disappear on a more tuned deployment. A search peak may also reflect measurement noise or a change to the evaluation workload. Comparing agents therefore requires recording the conditions of the search and checking the artifact it actually delivers.

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

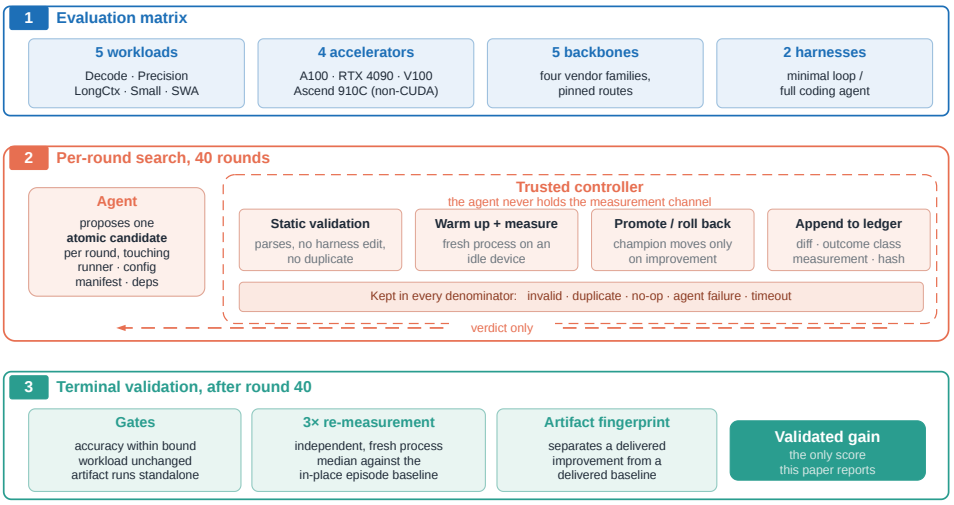


Figure 1: ACCELMARK-AGENT. Five workloads, four accelerators, five backbones and two harnesses define the evaluation scope; only the A100 matrix is fully crossed. The agent submits one atomic candidate per round; a trusted controller validates, measures, promotes or rolls back, and records it. A reported gain requires an accuracy-gated, fingerprinted artifact and three independent re-measurements.

We introduce ACCELMARK-AGENT (Figure 1) to study this setting. Five workloads expose different constraints, from mature decoding paths to sliding-window attention. A trusted controller measures submitted candidates, restores the incumbent after unsuccessful changes and retains every round in the ledger. Final gains use an in-place baseline and three independent re-measurements after accuracy and integrity checks. Artifact fingerprints distinguish a changed deployment from a re-measurement of the baseline.

The fully crossed A100 study evaluates five backbones under two harnesses on all five workloads. Of its fifty cells, twenty-six deliver a changed artifact with a gain above 2%. The results separate two difficulties: producing candidates that execute and finding improvements once execution succeeds. Gains also depend strongly on the starting deployment. In the sliding-window workload, the two largest gains follow the cache-admission interaction described above; eight other searches do not find the same combination. Follow-up studies examine how these observations depend on repeated search, starting configuration, device and validation protocol. They support comparisons among recorded setups, while limited replication prevents a stable ranking of backbone or harness quality.

Our contributions are:

- A deployment-optimization benchmark spanning five workloads, with a fully crossed A100 matrix and additional GPU and non-CUDA portability tracks (Section 3.2).
- A controller-mediated evaluation protocol that separates candidate search from final measurement and records failed rounds, accuracy checks and artifact identity (Section 3.1).
- An empirical analysis of $2{,}000$ matrix rounds and $3{,}401$ A100 rounds overall, including a case study of the largest gains and studies of the conditions that limit their interpretation (Sections 4–5).

2 REQUIREMENTS AND RELATED WORK

A serving pipeline shares memory, scheduling and execution resources across stages (Kwon et al., 2023; Zheng et al., 2024; Agrawal et al., 2024). An optimization that helps one stage can lose its benefit when another stage becomes limiting. Evaluation must therefore cover workloads with different constraints and report the starting configuration. Measurement requires separate controls: selecting the best round of a noisy search introduces selection bias (Cawley & Talbot, 2010), and a faster candidate may have changed the computation or workload. The final artifact must be checked and re-measured independently, with failed searches retained in coverage statistics.

Existing benchmarks address these requirements at different scales (Appendix A). Component benchmarks evaluate isolated GPU kernels (Ouyang et al., 2025; Lange et al., 2025; Younesian et al., 2026); ISO-Bench evaluates inference-stack patches derived from historical changes (Nangia et al., 2026); and InferenceBench evaluates complete servers, primarily on H100, with an A100 transfer study (Yeon et al., 2026). InferenceBench also reports that budget-matched random search outperforms its evaluated agents. Automated configuration methods provide further non-agentic alternatives (Lindauer et al., 2022; Bergstra et al., 2011). Our small configuration search is a reference point rather than a budget-matched comparison (Appendix B.2).

ACCELMARK-AGENT studies complete deployments across five workloads and three GPU generations, with a separate Ascend portability track. It treats the harness as an experimental factor (Vats & Golev, 2026; Nangia et al., 2026) and combines independent final measurements with artifact audits and local noise calibration (Mytkowicz et al., 2009; Kalibera & Jones, 2013). Its baselines are configured serving engines. Reported gains consequently describe improvements over those deployments and cannot be compared directly with gains over an unoptimized framework.

3 THE ACCELMARK-AGENT BENCHMARK

The benchmark pairs a measurement protocol with a set of deployment tasks. We first define how an episode produces a validated result, then describe the workloads, agents and studies to which that protocol is applied.

3.1 MEASURING A DEPLOYMENT PLAN

An episode gives the agent a working deployment, a workload and R search rounds. In each round, the agent submits one candidate edit to the runner, engine configuration, runtime manifest or local dependencies. Together these components form the delivered artifact.

The controller owns the measurement channel. It rejects candidates that fail static checks, touch protected evaluation code or duplicate a previous artifact. It measures accepted candidates in a fresh process on an idle device, promotes a candidate if it exceeds the incumbent's throughput, and otherwise restores the incumbent. Every outcome enters the ledger. After search, the champion must pass terminal checks, including an accuracy gate that bounds deviation from the episode baseline. Three fresh-process measurements then determine the final gain:

$$G_{\text{final}} = 100 \left(\frac{\text{median}\{f^{(1)}, f^{(2)}, f^{(3)}\}}{b} - 1 \right), \tag{1}$$

where $f^{(i)}$ are the three re-measurements and b is the median of three baseline measurements taken in place at the start of the same episode.

The accuracy gate compares a fixed MMLU subset under the candidate and baseline configurations, allowing an absolute score deviation of at most 0.03. For PRECISION, each of the five required formats is checked separately, and dropping a format fails the gate. This relative check separates optimization-induced changes from differences in the served models' baseline accuracy. It does not certify behavior outside the subset; workload integrity and standalone execution require separate checks (Appendix C).

Four rules govern interpretation of this gain. Baselines are measured at the start of each episode to account for drift. Failed, duplicate, unchanged and timed-out rounds remain in the denominator of search statistics. An unvalidated champion has no final result; its search peak cannot substitute for one. Finally, fingerprints record artifact identity and support audits of protected changes. Section 5.4 examines cases where these rules change the reported outcome; Appendix C specifies the checks.

We also report the effective-round ratio $R_{\text{eff}} = N_{\text{measured}}/R$ and promotion ratio $R_{\text{prom}} = N_{\text{promoted}}/R$. The first measures how often candidates reach measurement; the second measures how often they improve the incumbent. A cell is a *success* only if its artifact differs from the baseline and $G_{\text{final}} > 2\%$, above every observed gain from an unchanged delivery (Section 5.4). A deterministic rule assigns each measured candidate to one of eight pipeline stages: engine, compilation, kernel, system, memory, scheduling, host, or environment and build (Appendix C.3).

3.2 BENCHMARK DESIGN

Workloads. Table 1 varies both the main performance constraint and the maturity of the serving path. DECODE tests a heavily tuned path; SWA tests interactions between window semantics, KV storage and scheduling. The remaining workloads stress numeric-format support, long-context prefill and small-model overhead. This mix lets us examine gains across different amounts of remaining headroom. We use established Llama 3, Qwen2.5 and Mistral checkpoints (Grattafiori et al., 2024; Hui et al., 2024) selected for the workload and evaluated stack support (Appendix C). Only SMALL discloses a reference throughput to the agent: $35{,}341$ tokens/s.

Table 1: The five workloads. Each is scored by the end-to-end throughput of the delivered artifact under its own concurrency profile, behind the same terminal gates. Lengths are in tokens and `c` denotes request concurrency.

Workload	Model and profile	Main constraint and path maturity
DECODE	Llama-3 8B, BF16 4096 ctx, 512 out, c8–128	Long decode, CUDA Graphs, KV capacity. *Mature.*
PRECISION	Llama-3.1 8B, BF16–W8A8 c1–64	Quantized kernels and backend coverage across formats. *Moderate.*
LONGCTX	Llama-3.1 8B, BF16 28K ctx, 256 out, c1–4	Prefill memory, chunking, the OOM boundary. *Moderate.*
SMALL	Qwen2.5 0.5B, BF16 2048 ctx, 256 out, c64	Launch, host and scheduling overhead. *Mature engine, untuned runner.*
SWA	Mistral-7B, true SWA 10K input, 4096 window	Window semantics, SWA KV storage, compilation. *Young.*

Accelerators. A100-SXM4-80GB carries all five workloads. Additional tracks use V100 in FP16, RTX 4090 with a 24 GB memory limit, and Ascend 910C with vLLM-Ascend in BF16. The Ascend track evaluates portability; it has no validated final gain.

Agents. We cross five backbones (GLM-5.2, Claude Opus 4.8, Codex 5.6, DeepSeek V4 Pro and DeepSeek V4 Flash) with `mini-swe-agent`, a minimal shell loop, and `Claude Code`, a coding harness with its own tools. The models are accessed through fixed routes, with high reasoning effort for Codex (Appendix D). Within each workload, both harnesses receive a byte-identical task prompt. Every matrix search has $R = 40$ rounds.

3.3 EXPERIMENTAL DESIGN

The A100 matrix is the fully crossed study used for agent comparisons (Table 2). Supporting studies examine search variability, starting configuration, device and protocol. The A100 cohorts used here contain $3{,}401$ rounds (Appendix C.4); cross-accelerator tracks have separate coverage counts. We analyze the recorded experiments without re-running them for this manuscript.

Table 2: The studies in this paper. All studies use the controller; only supported final measurements enter gain statistics. Coverage and exceptions are reported by cohort.

Study	What it varies	Scale	Results
A100 matrix	5 workloads × 5 backbones × 2 harnesses, one 40-round search per cell	50 cells; 2,000 rounds	§4.1–4.3
Repeat searches	search variability: two DECODE cells searched again under the same prompt	2 cells; 80 rounds	§5.1
Stage ablation	starting runner: SMALL, one pipeline stage open per arm, from a reference and an exhausted runner	13 arms + 1 joint search	§5.2
Cross-accelerator	device: DECODE and SMALL on V100, RTX 4090 and Ascend 910C	per track (Table 5)	§5.3
Earlier cohort	protocol: unbalanced SMALL searches, used only to audit it	15 traces; 600 rounds	§5.4

4 MAIN RESULTS: THE A100 MATRIX

Table 3 reports all fifty A100 matrix finals. We first examine whether agents produce measurable candidates, then analyze the changes that improve throughput. Gains refer to validated finals unless explicitly identified as search measurements. Appendix B provides the settings and supporting analyses.

Table 3: A100 matrix: accuracy-gated final gain G_{final} (%) against each episode's in-place baseline. Grey cells delivered an artifact fingerprint-identical to the baseline; their spread, -1.95% to $+1.48\%$, shows the observed variability of unchanged deliveries. *Succ.* counts changed artifacts above $+2\%$. ‡No model tokens were recorded in any of the forty rounds. §The first search failed final validation; its repeat is shown. Bold: best cell per workload.

Backbone	Harness	DECODE	PRECISION	LONGCTX	SMALL	SWA	Median	Succ.
GLM-5.2	`Claude Code`	0.03	**35.25**	**4.97**	0.11‡	**211.72**	4.97	3/5
	`mini-swe-agent`	10.93	11.27	4.62	−0.78	7.19	7.19	4/5
Opus 4.8	`Claude Code`	11.40	34.82	4.22	**21.02**	9.07	11.40	5/5
	`mini-swe-agent`	−0.09	32.89	1.96	4.10	137.82	4.10	3/5
Codex 5.6	`Claude Code`	−0.15	29.32	−0.11‡	10.46	10.87	10.46	3/5
	`mini-swe-agent`	**12.34**	12.95	3.60	8.86	2.62	8.86	5/5
DS V4 Pro	`Claude Code`	−0.07	−1.06	0.50	7.53	−0.02‡	−0.02	1/5
	`mini-swe-agent`	−0.21	10.95	0.03	−0.13§	0.63	0.03	1/5
DS V4 Flash	`Claude Code`	−0.30	−1.17	−0.01	11.19	0.17	−0.01	1/5
	`mini-swe-agent`	0.14	−1.95	0.42	1.48	0.04	0.14	0/5
Workload median		−0.02	+12.11	+1.23	+5.81	+4.91		
Successes / 10		3	7	4	6	6		26/50

4.1 WHAT SEPARATES AGENTS

The clearest difference between agent setups was their ability to submit measurable candidates. Codex and Opus succeeded in eight of ten cells each, GLM-5.2 in seven, DeepSeek V4 Pro in two and V4 Flash in one. GLM-5.2, Opus and Codex reached measurement in 44–55% of rounds, compared with 5% and 4% for the two DeepSeek backbones (Figure 2a). The latter submitted unchanged artifacts in 188 and 259 of their 400 rounds. Outside SMALL, they produced only four measured candidates in 640 rounds, leaving little evidence of optimization quality in those settings. Seven of their eight `Claude Code` cells on these workloads consumed 18–22M tokens each without producing a measured candidate.

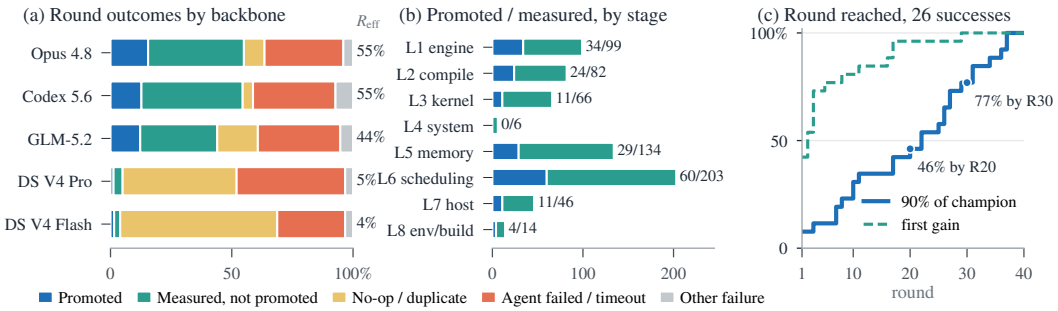


Figure 2: **Anatomy of the 2,000 A100 rounds.** (a) Outcome class of every round, by backbone; R_{eff} is the share that produced a legal measurement. (b) Measured rounds by the stage the controller assigned, split into promoted and not promoted; the fraction is the acceptance rate. (c) Of the 26 successful searches, the share whose best-so-far had reached 90% of its round-40 champion (solid) or any improvement (dashed) by each round.

These failures depended on the full setup. DeepSeek V4 Flash/`Claude Code` measured no candidates on four workloads but delivered $+11.19\%$ on SMALL, with 15 measured rounds on the same A100. V4 Pro/`Claude Code` delivered $+7.53\%$ there. The SMALL sandbox and prompt also differ, so the contrast cannot isolate the workload itself. Conversely, both GLM-5.2 searches on SMALL failed in every round; the `Claude Code` search recorded no model output. These cells remain in the denominators but provide no evidence about GLM-5.2's optimization ability. Among the thirty cells of GLM-5.2, Opus and Codex, effective-round ratio was only weakly associated with final gain (Spearman $\rho = 0.32$): measuring more candidates did not reliably produce a larger improvement.

Token use further distinguishes execution failures from inexpensive no-op search. The twenty-five `Claude Code` cells consumed 331M recorded tokens at a total provider charge of $679. DeepSeek V4 Flash used 100.6M tokens for one successful cell, whereas Codex used 29.7M for three. These are costs under the recorded routes and prices, not a normalized comparison of model efficiency. Separately, 257 `Claude Code` rounds recorded no model tokens, including three en-

tire cells. Missing gateway responses prevent attributing those failures to the model or harness, and `mini-swe-agent` lacks equivalent per-round usage records (Appendix B.1).

Harness comparisons also varied by workload. Of sixteen pairs separated by more than one percentage point, `Claude Code` led in eleven and `mini-swe-agent` in five. The direction reversed within four of the five backbones. Opus performed better under `Claude Code` on four workloads, yet under `mini-swe-agent` on SWA by 129 points. These are descriptive differences between individual searches. The repeat study in Section 5.1 shows why they should not be treated as stable harness rankings.

4.2 Where the Gains Come From

Engine and scheduling edits accounted for 94 of 173 promotions (Figure 2b). Their promotion rates were $34/99$ and $60/203$, compared with $11/66$ for kernel edits and $0/6$ for system-topology edits. The two DECODE repeats each gained about 11% by changing one engine argument: `stream_interval` or chunked-prefill size (Appendix B.5). Neither required a rewrite of runner logic.

Useful changes continued to appear late in search. Among successful cells, 81% found their first improvement by round 10, but only 46% reached 90% of their round-40 champion's gain by round 20 and 77% by round 30 (Figure 2c). A twenty-round budget would have missed the first improvement of Codex/`Claude Code` on PRECISION (round 29; final $+29.32\%$) and the main jump of Opus/`mini-swe-agent` on SWA (round 22; final $+137.82\%$). These trajectories show sensitivity to budget within the observed searches; they do not establish what a longer search would achieve.

4.3 Case Study: How the Largest Gains Were Found

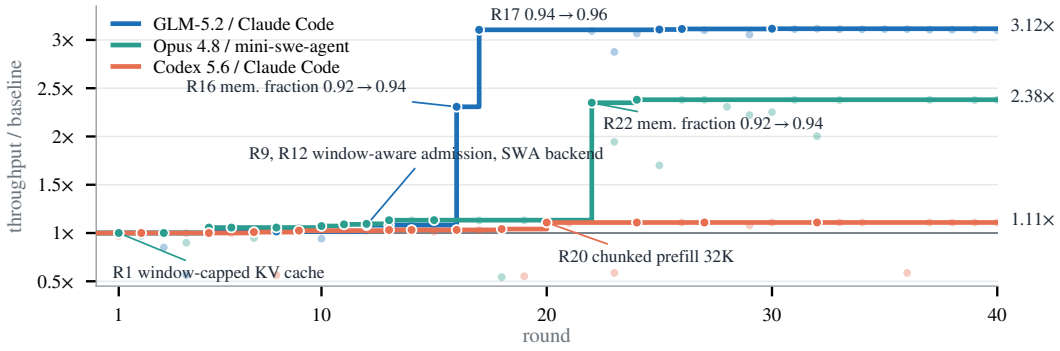


Figure 3: **Three SWA searches, round by round.** Step lines are the champion, light dots every measured candidate and white-ringed dots the promotions, all as throughput over the episode baseline; unmeasured rounds are omitted. Both trajectories rise sharply after combining window-aware KV handling with a larger static memory allocation.

Figure 3 compares the two SWA searches above $+130\%$ with Codex/`Claude Code`, the strongest search that used generic tuning alone. All three explored graph capture, compilation, chunked prefill and scheduler settings; none exceeded $1.14\times$ baseline by round 15. The large-gain searches then combined window-aware KV handling with a larger memory pool.

Their ledger notes identified a mismatch: the baseline used sliding-window attention but admitted requests as though each needed KV storage for its full 10K-token context. GLM-5.2/`Claude Code` adjusted the admission estimate in round 9 and selected an A100-compatible SWA backend in round 12. Opus/`mini-swe-agent` enabled SGLang's window-capped hybrid KV cache in round 1. These changes alone produced little improvement. Raising `mem_fraction_static` from 0.92 to 0.94 then roughly doubled throughput within one round, at round 16 for GLM-5.2 and round 22 for Opus. GLM-5.2's next increase to 0.96 added about 35%. Codex/`Claude Code` increased the KV-pool fraction without changing admission and saw no comparable gain; its final improvement was $+10.87\%$.

This sequence is consistent with a capacity threshold created by the interaction between admission and KV allocation. The interpretation rests on the ledgers and agent notes because the delivered runners are absent from the archive (Appendix B.4). The gains are also specific to the offline throughput objective: neither artifact met the workload's online time-to-first-token target at any tested rate.

The case illustrates why the starting deployment matters. A small argument change produced the largest gains only after the search identified the relevant interaction. The host-stage study in Section 5.2 provides a second example: batching tokenization within the timed path accounted for most of the improvement.

5 Sensitivity Analyses and Protocol Audit

The matrix fixes the round budget and controller but samples each setup only once. We next examine repeated searches, alternative starting configurations and device tracks, then audit how validation changes the reported result. These studies have different controls; their limitations are stated with each comparison.

5.1 Repeated Searches

Two DECODE cells were searched again with the same task prompt and forty-round budget. Codex/`mini-swe-agent` changed from $+12.34\%$ to $+11.64\%$. GLM-5.2/`Claude Code` changed from $+0.03\%$ to $+11.17\%$, finding its improvement through one engine argument at round 27. The `Claude Code` version also changed between runs, so the latter difference cannot be attributed solely to stochastic search (Appendix B.5).

Takeaways

Search variability. An eleven-point difference between repeated searches is comparable to observed harness gaps. Single-cell comparisons do not establish a stable ranking.

5.2 Starting Configuration

On SMALL, a mature engine still left host-side overhead in the reference runner. Stage-restricted searches delivered $+20.76\%$ from host edits and $+13.09\%$ from environment edits (Table 4). From a previously tuned runner at 46.4K tokens/s, no arm delivered more than $+0.05\%$; one accuracy-valid backend switch reduced throughput by 65.61%. Backbone and prompt also changed between these studies. The contrast therefore does not isolate starting-point effects, but it establishes an absolute ordering: no reference-start search reached the throughput of the tuned baseline.

Table 4: Stage-restricted searches from two starting configurations. SMALL on one A100, one stage open per arm. *Lost*: champion lost before final validation, reported as no result. Exhausted-start 0.00% entries indicate no delivered improvement, not a measured zero effect (Appendix B.3).

Stage opened	From the reference runner ($\approx$ 34K tokens/s)	From the exhausted runner (46.4K tokens/s)
L1 engine	peak $+15.30\%$, lost	-65.61%: accuracy-valid backend switch
L2 compilation	peak $+14.84\%$, lost	not run: used to build this runner
L3 kernel	peak $+12.05\%$, lost	0.00%: no evaluable candidate in 41 rounds
L4 system	no candidate above baseline	0.00%: 58 changes, none above baseline
L5 memory	peak $+14.43\%$, lost	
L6 scheduling	peak $+17.47\%$, lost	0.00%: mean -10.2% over 60 rounds
L7 host	$+\mathbf{20.76}\%$ validated	not run: runner already has the host fix
L8 environment	$+\mathbf{13.09}\%$ validated	not run
All stages	–	$+0.05\%$ validated after 120 rounds

The reference-start host arm helps explain this difference in headroom. Batching per-request tokenization increased the measured search gain to $+14.41\%$ in round 2. Batched chat-template rendering, token counting and related changes raised the champion to $+21.65\%$ by round 8; the validated final was $+20.76\%$. Of the next 32 rounds, 25 reached measurement but added only 0.35 percentage points to the champion. The previously tuned runner already included this batched host path, so repeating the same change offered no comparable opportunity (Appendix B.3).

A similar contrast appears on RTX 4090. DeepSeek V4 Flash gained $+56.43\%$ from 2,804 tokens/s, $+0.27\%$ from 15,311, and -0.27% from 37,886. The largest relative improvement ended at roughly one ninth of the third baseline. Precision and harness also differ across these runs, which precludes attributing the contrast to either the device or backbone alone.

Takeaways

Starting point. Relative gains should be reported alongside baseline and final throughput. A larger percentage need not yield a faster deployment.

5.3 DEVICE

Validation coverage fell sharply on some additional device tracks (Table 5). V100 yielded four valid finals in twenty registered conditions, all within 1.5% of baseline. RTX 4090 outcomes varied across precision, harness and starting configuration. Ascend 910C yielded no valid final; all five summarized runs that changed the runner also changed the protected evaluation configuration. The tracks therefore document delivery and portability outcomes under their recorded settings, without supporting a hardware ranking.

Table 5: **Cross-accelerator tracks**: validated finals over registered conditions and their gains. The A100 row is this study's reference track, not the matrix. †One search includes a namespace repair.

Accelerator	Track (workloads; precision)	Finals	Median	Min.	Max.
A100	DECODE/SMALL; BF16	21/23	$+1.48\%$	-0.78%	$+21.02\%$
RTX 4090	`Claude Code`, DECODE/SMALL; FP16	8/10	$+28.35\%$	$+0.26\%$	$+56.43\%$
RTX 4090	`mini-swe-agent`, DECODE/SMALL; BF16	5/10	$+0.27\%$	-3.62%	$+17.30\%$
RTX 4090	`Claude Code`, SMALL; BF16 (additional)	$5/5^{\dagger}$	-0.55%	-1.20%	-0.26%
V100	DECODE/SMALL; FP16	4/20	-0.25%	-1.42%	$+0.07\%$
Ascend 910C	DECODE/SMALL; BF16 (portability)	0/20	–	–	–

Takeaways

Portability. Report validated deliveries over registered conditions before comparing cross-device gains; missing finals are not zero gains.

5.4 PROTOCOL

Table 6 contrasts the largest recorded reading with the supported outcome. Across the fifty A100 matrix cells, search peaks and finals differed by at most 3.05 percentage points, without changing the 2% classification (Figure 4). Larger discrepancies occurred in the legacy cohort and additional device tracks. The V100 caching case also shows that an accuracy gate alone cannot establish workload integrity.

Table 6: **Best search readings versus supported outcomes.** All rows but the last are on SMALL; *legacy*: an earlier cohort (Appendix E). CC: `Claude Code`; mini: `mini-swe-agent`.

Cohort	Backbone / harness	Naive	Protocol outcome and the check responsible
V100	GLM-5.2 / mini	$\approx 255\times$	excluded: cached the workload; gate *passed*
Ascend 910C	GLM-5.2 / mini	$+241.6\%$	no result: protected evaluation configuration edited
A100 legacy	Codex 5.6 / CC	$+16.71\%$	no result: no valid accuracy-gate receipt
A100 stage arms	GLM-5.2 / CC, 5 arms	$+12$–17%	no result: champion lost before final validation
A100 legacy	Opus 4.8 / CC	$+20.92\%$	$+15.12\%$ after three re-measurements
V100	Opus 4.8 / mini	$+4.01\%$	-0.13% after three re-measurements
A100 matrix	all 50 cells	peak	within 3.05 points of G_{final}; none crosses 2%

Twenty unchanged matrix artifacts had final readings from -1.95% to $+1.48\%$. Resampling ten measurements of a fixed runner gave a 95% interval of about $\pm 1.5\%$ for a three-repeat median (Appendix B.2). The 2% success threshold exceeds these observed positive fluctuations and additionally requires a changed artifact. Baselines are measured within each episode because their spread reached 3.4% across the ten SMALL cells.

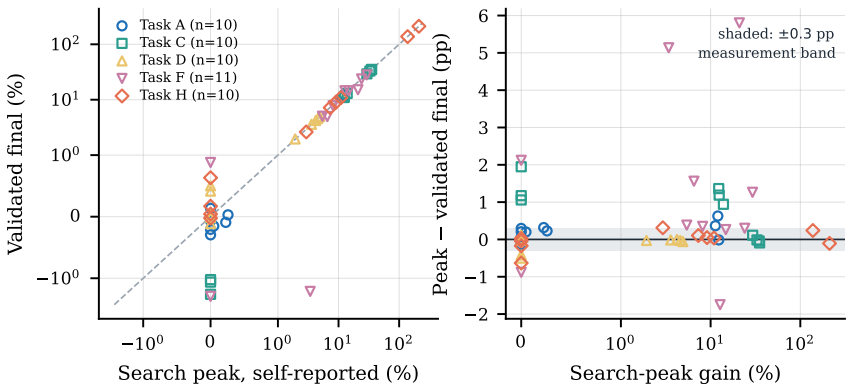


Figure 4: **Search peak against validated final** for the fifty matrix cells and eleven legacy traces. **Left:** dashed line, agreement; shaded band, the twenty unchanged deliveries. **Right:** what a naive reading overstates. Symmetric-log gain axes are linear within $\pm 1\%$; the difference axis is linear.

Takeaways

Validation. Accuracy checks, independent re-measurement and artifact audits address different failure modes. Passing one check does not establish a valid speedup.

Scope of the evidence. The benchmark measures the performance of complete agent setups, including their ability to operate the supplied environment. Keeping failed rounds in the denominator makes this operational cost visible, but it does not identify the cause of every failure. The recorded model routes share one commercial endpoint, and the archive cannot independently verify the weights served or distinguish all gateway failures from harness failures. Similarly, the starting-point studies vary backbones and prompts together, while the device tracks vary precision and software. Their contrasts describe observed deployments rather than isolated causal effects. Finally, the ten-attempt configuration search is not budget-matched to the agent searches. The present results therefore do not establish that code-generating agents outperform automated configuration search. Replicated searches with matched starts and budgets are needed to answer that question and to assess stable backbone and harness differences. Appendix F discusses these constraints in detail.

Reading a benchmark entry. These distinctions suggest a practical order for comparing results. First establish which registered conditions produced supported final measurements, retaining incomplete searches in the coverage count. Then compare the absolute baseline and final throughput under matched workload and precision settings. Relative gain alone can favor a weaker starting deployment, as the RTX 4090 examples show. Finally, inspect whether the delivered artifact changed and whether the improvement survives independent measurement. The round outcomes provide a separate diagnosis: a search that repeatedly submits unchanged files needs a different intervention from one that measures many valid but slower candidates. Stage labels can help locate these behaviors, although they do not establish the mechanism of a speedup. Under this reading, the benchmark reports both the optimization achieved and the evidence available to explain it, without treating a missing final, an unchanged delivery and a measured regression as interchangeable outcomes.

6 Conclusion

AccelMark-Agent evaluates agents as optimizers of complete serving deployments. Its A100 matrix distinguishes execution failures from unsuccessful optimization, while supporting studies show how outcomes depend on the starting configuration, search run and device stack. The largest gains involved sliding-window admission and KV allocation; validation excluded altered workloads and undelivered champions. Scores should therefore identify the deployment, baseline throughput and validated artifact. Limited replication and unmatched device tracks leave stable agent rankings and causal device effects unresolved. The released tasks, controller, prompts and ledgers support further evaluation with repeated searches and budget-matched configuration baselines.

AI USE STATEMENT

LLM-based agents are the systems evaluated in this study; their role in generating candidate serving configurations is described in the experimental methodology. Separately, we used generative AI assistance to revise manuscript organization and wording, check consistency across tables and text, and help interpret archived experiment and validation records. AI-assisted suggestions were checked against the underlying records before inclusion. The authors made the final decisions about the reported claims and take responsibility for the full content of this paper.

REFERENCES

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 117–134, 2024.

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM international conference on architectural support for programming languages and operating systems, volume 2*, pp. 929–947, 2024.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Gavin C Cawley and Nicola LC Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *The Journal of Machine Learning Research*, 11:2079–2107, 2010.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics: Methodology and distribution*, pp. 569–593. Springer, 1992.

Spandan Garg, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. Perfbench: Can agents resolve real-world performance bugs? In *Proceedings of the 2026 International Workshop on Agentic Engineering*, pp. 165–172, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Yin Zheng, Zejian Yuan, and Zejun Ma. Swe-perf: Can language models optimize code performance on real-world repositories? *arXiv preprint arXiv:2507.12415*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Tomas Kalibera and Richard Jones. Rigorous benchmarking in reasonable time. In *Proceedings of the 2013 international symposium on memory management*, pp. 63–74, 2013.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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.

Robert Tjarko Lange, Qi Sun, Aaditya Prasad, Maxence Faldor, Yujin Tang, and David Ha. Towards robust agentic cuda kernel benchmarking, verification, and optimization. *arXiv preprint arXiv:2509.14279*, 2025.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23 (54):1–9, 2022.

Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. Swe-fficiency: Can language models optimize real-world repositories on real workloads? *arXiv preprint arXiv:2511.06090*, 2025.

Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F Sweeney. Producing wrong data without doing anything obviously wrong! *ACM Sigplan Notices*, 44(3):265–276, 2009.

Ayush Nangia, Shikhar Mishra, Aman Gokrani, and Paras Chopra. Iso-bench: Can coding agents optimize real-world inference workloads? *arXiv preprint arXiv:2602.19594*, 2026.

Anne Ouyang, Simon Guo, Simran Arora, Alex L Zhang, William Hu, Christopher Ré, and Azalia Mirhoseini. Kernelbench: Can llms write efficient gpu kernels? *arXiv preprint arXiv:2502.10517*, 2025.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. Gso: Challenging software optimization tasks for evaluating swe-agents. *Advances in Neural Information Processing Systems*, 38, 2026.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 10–19, 2019.

Naman Vats and Oleg Golev. The scaffold effect in coding agents: Harness choice as a hidden variable in coding-agent evaluation. *arXiv preprint arXiv:2607.22585*, 2026.

Ruhan Wang, Yucheng Shi, Zongxia Li, Zhongzhi Li, Yue Yu, Junyao Yang, Kishan Panaganti, Haitao Mi, Dongruo Zhou, et al. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, et al. Re-bench: Evaluating frontier ai r&d capabilities of language model agents against human experts. *arXiv preprint arXiv:2411.15114*, 2024.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Jehyeok Yeon, Ben Rank, and Maksym Andriushchenko. Inferencebench: A benchmark for open-ended llm inference optimization by ai agents. *arXiv preprint arXiv:2607.20468*, 2026.

Sharareh Younesian, Wenwen Ouyang, Sina Rafati, Mehdi Rezagholizadeh, Sharon Zhou, Ji Liu, Yue Liu, Yuchen Yang, Hao Li, Ziqiong Liu, et al. Agentkernelarena: Generalization-aware benchmarking of gpu kernel optimization agents. *arXiv preprint arXiv:2605.16819*, 2026.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Extended Related Work

Section 2 positions AccelMark-Agent against the two requirements of coverage and verified measurement. Table 7 summarizes the comparison and this appendix gives the full discussion.

Table 7: Scope of representative performance benchmarks. Entries describe their evaluation targets; they do not imply that an unlisted control or hardware experiment is absent.

Benchmark	Primary evaluation object	Scope relevant to this comparison
KernelBench (Ouyang et al., 2025)	GPU kernel implementations	Correctness and speed against reference implementations
GSO, SWE-Perf (Shetty et al., 2026; He et al., 2025)	Repository performance tasks	Optimization against executable workloads
ISO-Bench (Nangia et al., 2026)	Inference-stack optimization tasks	Tasks derived from historical vLLM and SGLang changes
InferenceBench (Yeon et al., 2026)	Complete serving deployment	Main H100 evaluation; A100 transfer ablation; quality and integrity checks
AccelMark-Agent	Complete serving deployment	Five workloads; three accelerator generations; controller ledger and artifact audit

Repository-scale agent evaluation. SWE-bench (Jimenez et al., 2024) provides a widely used test-based benchmark for repository-scale issue resolution. SWE-agent and OpenHands (Yang et al., 2024; Wang et al., 2025) provide agent interfaces and execution environments evaluated on such tasks. MLAgentBench (Huang et al., 2023), MLE-bench (Chan et al., 2025) and RE-Bench (Wijk et al., 2024) extend agent evaluation to machine-learning experimentation and research engineering, using task-specific feedback and grading protocols. Performance-based evaluation additionally requires control of measurement bias and uncertainty (Mytkowicz et al., 2009; Kalibera & Jones, 2013).

Component evaluations. KernelBench and its successors (Ouyang et al., 2025; Lange et al., 2025; Younesian et al., 2026) score generated GPU kernels against a reference implementation. These evaluations make reference-based correctness checking possible, but kernel-level speedups alone do not establish a serving-system speedup. End-to-end serving additionally depends on KV-cache management and request scheduling (Kwon et al., 2023; Agrawal et al., 2024).

Patch-level evaluations. GSO (Shetty et al., 2026), SWE-Perf (He et al., 2025) and SWE-fficiency (Ma et al., 2025) evaluate repository-level performance optimization against executable workloads. They target general codebases rather than serving stacks, and therefore address a broader class of runtime improvements than the serving stages in Figure 5. ISO-Bench (Nangia et al., 2026) does apply the patch-level form to serving, curating optimization tasks from vLLM and SGLang pull requests, including changes across several serving stages. Its historical-change formulation differs from our open-ended deployment assembly, but does not in itself prohibit edits spanning multiple stages.

End-to-end evaluation. InferenceBench (Yeon et al., 2026) establishes end-to-end evaluation of the assembled pipeline: an agent must deploy and optimize a complete OpenAI-compatible server on one H100 within a two-hour budget, under a quality gate and an integrity judge, with speedup

measured over a naive PyTorch baseline; an appendix additionally tests transfer to A100. It reports that budget-matched random search outperforms every agent configuration it evaluates, a comparison relevant to Section 4. Our experiments separately examine the headroom associated with different starting configurations. Our reference point is a configured serving engine rather than an unoptimized framework, so the reported percentages have different denominators and are not directly comparable. Our episodes use a round budget, and a separate controller executes all scored measurements.

Non-agentic configuration search. Automated configuration search optimizes exposed deployment parameters without generating code. General methods such as SMAC3 and tree-structured Parzen estimation provide established approaches to automated configuration and hyperparameter optimization (Lindauer et al., 2022; Bergstra et al., 2011). We ran a small configuration search over the serving engine's exposed arguments on SMALL and report it as a reference point rather than a budget-matched baseline (Appendix B.2).

Harness and measurement integrity. Reported agent scores can depend on the harness (Vats & Golev, 2026; Nangia et al., 2026). We therefore vary the harness explicitly and record task-prompt hashes to verify that paired searches receive the same task statement. Harness Handbook studies how agents locate and modify harness behavior (Wang et al., 2026), and PerfBench evaluates real-world performance bugs (Garg et al., 2026). Separately, work on performance evaluation documents measurement bias and the need to quantify uncertainty (Mytkowicz et al., 2009; Kalibera & Jones, 2013). Our calibration measures this variability in the evaluated setting (Section 5.4).

B Experimental Settings and Supporting Analyses

This appendix collects the settings and derivations behind Sections 4 and 5. The analyses use the archived ledgers and validation summaries; no experiment was re-run for this manuscript.

B.1 The A100 Matrix

The matrix crosses five workloads, five backbones and two harnesses, one forty-round search per cell. Within each workload the two harnesses received a byte-identical task prompt (Table 14). Three details apply to single cells or columns.

SMALL. This workload runs in a sandbox that discloses the public reference throughput of $35{,}341$ tokens/s to the agent; the other four workloads disclose no reference. The first DeepSeek V4 Pro/`mini-swe-agent` search on SMALL delivered the baseline and then failed final validation; Table 3 shows its repeat (-0.13%, again an unchanged delivery). Two further SMALL searches ran DeepSeek V4 Pro and V4 Flash under DeepSeek's own agent harness (a pre-release version): Pro delivered $+2.19\%$, and Flash reached a search peak of $+8.60\%$ but failed final validation. They are outside the matrix because the harness exists for one vendor only.

Rounds without model output. Token accounting is recorded for every `Claude Code` round. 257 of the $1{,}000$ `Claude Code` rounds (144 agent failures and 113 agent timeouts) recorded zero model tokens with usage evidence present, consistent with the absence of a recorded model response. Three cells consist entirely of such rounds (Codex/`Claude Code` on LONGCTX, GLM-5.2/`Claude Code` on SMALL, DeepSeek V4 Pro/`Claude Code` on SWA; ‡ in Table 3); the archive does not retain the gateway responses needed to tell an endpoint failure from a harness failure. The GLM-5.2/`mini-swe-agent` search on SMALL also failed in all forty rounds (30 agent failures, 10 timeouts); `mini-swe-agent` records no usage, so it cannot be classified the same way, and neither SMALL cell supports a conclusion about GLM-5.2's optimization ability. We keep these cells and rounds in every denominator, as the protocol prescribes, and flag them so that they are not read as statements about the backbone. `mini-swe-agent` does not record per-round usage, so the same check is not possible for it.

The four-workload subset. Restricted to the forty cells of DECODE, PRECISION, LONGCTX and SWA, the success counts under the definition of Section 3.1 are $7/8$ (GLM-5.2), $6/8$ (Opus 4.8 and Codex), $1/8$ (DeepSeek V4 Pro) and $0/8$ (V4 Flash), and the outcome distribution is that of Table 15

without SMALL. Adding SMALL changes one qualitative conclusion: the DeepSeek execution failures depend on the setup; the workload, sandbox and prompt change together.

B.2 MEASUREMENT CALIBRATION AND THRESHOLDS

Repeatability. Ten repetitions of a fixed runner on an idle A100 (SMALL reference runner, 38,271 tokens/s median at the primary concurrency) gave coefficients of variation of 1.00% at concurrency 64, 1.76% at 16 and 2.03% at 4 (Figure 8). The bootstrap (Efron, 1992) 95% interval of the ten-run median is 37,863 to 38,469 (-1.1% to $+0.5\%$). Resampling three runs from the same ten puts the 95% interval of a three-repeat median at about $\pm 1.5\%$; the mean absolute deviation of a three-repeat median from the ten-run median is 0.39%. The latter is a typical deviation rather than an interval, and read as a band it would understate the uncertainty of a single reported final.

The empirical null. Twenty matrix cells delivered an artifact whose fingerprint equals the baseline's. Their finals, which differ from zero only through measurement, span -1.95% to $+1.48\%$ (median -0.015%); the three unchanged PRECISION cells all read between -1.95% and -1.06%, which suggests a downward measurement shift in those episodes but does not establish the direction of error in changed artifacts. The success threshold of 2% on changed artifacts sits above every one of these readings and above the three-repeat interval. It also coincides with the success label the controller records for each episode. Between 1% and 2% lies one changed cell (Opus 4.8/`mini-swe-agent` on LONGCTX, $+1.96\%$) and one unchanged cell (DeepSeek V4 Flash/`mini-swe-agent` on SMALL, $+1.48\%$); a gain-only 1% threshold would count the latter, whereas the artifact-change requirement would still exclude it.

Baseline drift. Baselines are measured in place because they move. Across the ten cells of a workload the in-place baseline medians spread by 0.19% (PRECISION), 0.53% (LONGCTX), 0.54% (DECODE), 1.08% (SWA) and 3.37% (SMALL); the same SMALL reference runner measured between 33,531 and 34,841 tokens/s across the eight stage-restricted arms. The host-bound workload moves most, which is why a table of fixed reference numbers would bias every SMALL gain.

Promotion. The controller promotes a candidate whenever its single measurement exceeds the incumbent's, with no margin. Under the null of no real improvement, a forty-round search therefore promotes noise; the maximum of n null measurements at the calibrated noise level is expected between $+1.5\%$ and $+2.2\%$ for the round counts we observe. Independent final measurements reduce this selection effect because promotion only nominates a champion: the reported number is the post-search, gated three-repeat median, and the peak-to-final comparison of Figure 4 shows how far the two can differ.

Configuration search. A non-agentic search over ten settings of the engine's exposed arguments (memory fraction, CUDA-graph batch bound, chunked-prefill size) on the SMALL reference runner selected a configuration whose validated final was $+0.16\%$ over the 38,271 tokens/s reference. Its eighth attempt differed from the ninth only in the CUDA-graph batch bound (128 instead of 256) and measured 13,149 tokens/s, 65.6% below the reference, while remaining legal and accuracy-valid. This is a different event from the -65.61% SGLang-to-vLLM switch of Section 5.2; the similarity of the two numbers is a coincidence.

B.3 STAGE-RESTRICTED SEARCHES FROM TWO STARTING POINTS

Two starting runners. The *reference* runner is the public SMALL runner (about 34K tokens/s). The *exhausted* runner (46,390 tokens/s) is the product of the earlier layered searches on the same workload: an offline fast path, batched chat-template rendering, tokenization and decoding on the host, a Triton decode backend with FlashAttention-3 prefill, and a piecewise CUDA-graph policy. It therefore already contains the batched host path that the reference-start host arm later rediscovered, which is why the host and environment stages were not run from it.

Reference start. Each of the eight arms ran forty rounds under GLM-5.2/`Claude Code` (harness 2.1.267) with its own stage-restriction prompt, generated from one template. Two arms completed the protocol: host ($+20.76\%$, 42,075 tokens/s) and environment and build ($+13.09\%$). Five arms reached search champions of $+15.30\%$ (engine), $+14.84\%$ (compilation), $+12.05\%$ (kernel), $+14.43\%$ (memory) and $+17.47\%$ (scheduling), four of them in round 1 or 2, and lost them before final validation. The system arm never produced a candidate above baseline. All six ended with-

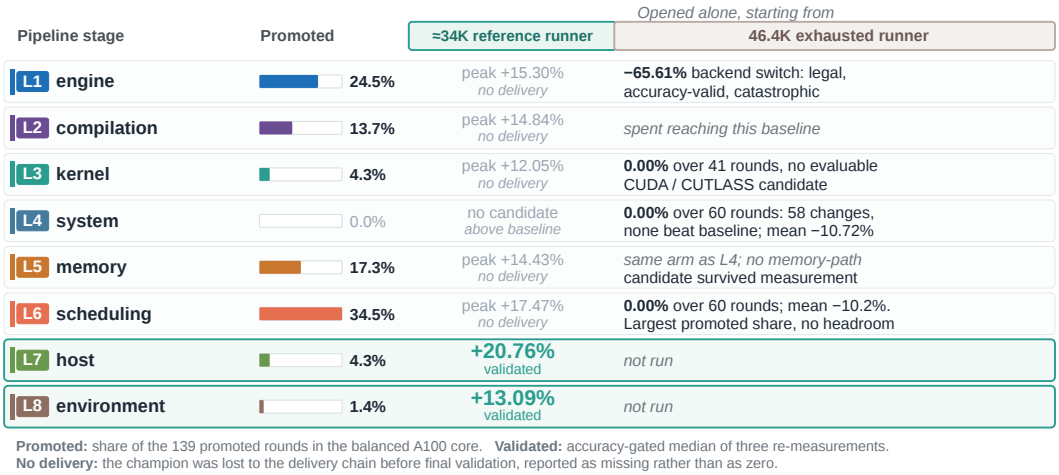


Figure 5: Stage-restricted searches and pipeline coverage. The eight stages the controller uses to classify every measured candidate, each stage's share of promoted rounds (balanced four-workload core; the five-workload shares are in Table 12), and what each returned when opened alone, with the rest frozen, from two starting configurations on SMALL. Backbones and prompt wording also differ between the two blocks (see text). *Not run*: the exhausted runner already contained the host-stage fix (see text). The classifier keywords per stage are in Appendix C.3.

out a final, five to an absent artifact at final validation and one to a device that was not idle when the controller tried to start; the six missing finals are reported as missing, not as zero; recovering them would require the corresponding artifacts to be available for final validation; fingerprints alone cannot reconstruct missing files.

Exhausted start. Five single-stage arms ran from the exhausted runner, with an earlier generation of stage prompts written separately for each arm and three different backbones (Kimi K2.7, DeepSeek V4 Flash and V4 Pro): system and memory (60 rounds, 58 measured changes, none above baseline, mean -10.72%), scheduling (60 rounds, mean -10.2%), kernel via CUDA/CUTLASS (41 rounds, no evaluable candidate), model algorithm (best candidate $+1.50\%$, not delivered) and an engine-backend arm whose SGLang-to-vLLM switch was legal, accuracy-valid and 65.61% slower. A 120-round search with every stage open (DeepSeek V4 Flash) produced 116 valid measurements and delivered $+0.05\%$; its runner source hash never changed, while 66 rounds edited both configuration files (Figure 9).

Interpretation. Starting configuration, backbone and prompt wording vary together. The contrast cannot isolate their effects. The exhausted-start arms span three backbones, and the joint search produced measurements in 116 of 120 rounds, but neither observation rules out a backbone effect. The supported comparison is the absolute throughput ordering: no reference-start search in the reported cohorts reached the exhausted runner's baseline.

Host-stage trajectory. In round 2, replacing per-request tokenization inside the timed window with one batched call gained $+14.41\%$ (Figure 6). Batched chat-template rendering and token counting, their consolidation, and removal of an unused attention mask raised the champion to $+21.65\%$ by round 8. In the remaining 32 rounds the agent tried direct calls to the Rust tokenizer backend, garbage-collector and allocator tuning, thread-pool limits and bytecode-level changes to the result loop; 25 reached measurement, 23 were not promoted, and together they moved the champion by 0.35 points.

B.4 THE TWO LARGEST GAINS

Table 8 gives the promoted rounds behind the two gains above $+130\%$ traced in Section 4.3 and Figure 3. Both searches first made the KV cache or its admission estimate respect the $4{,}096$-token window, which on its own changed little, and then crossed a capacity threshold by raising the static memory fraction to 0.94, at which throughput roughly doubled in one round. The same GLM-5.2

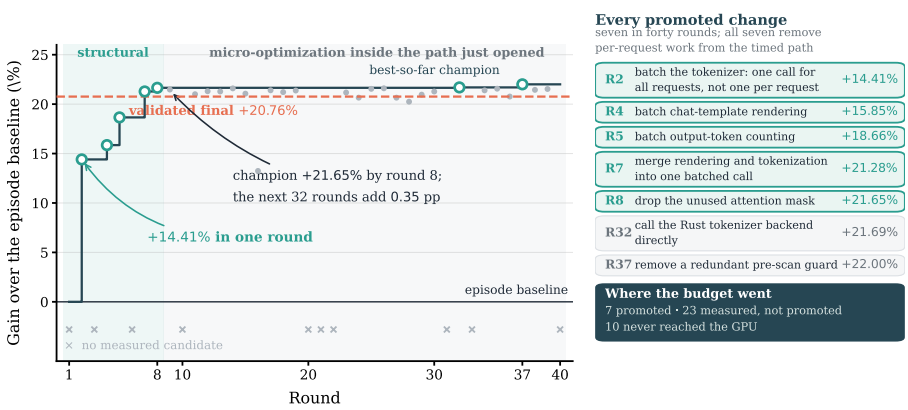


Figure 6: Host-stage search from the reference runner: GLM-5.2 under `Claude Code` reached a validated $+20.76\%$ in forty rounds. The plot shows measurements and promotions; the right panel identifies the promoted changes. Batched tokenization produced most of the gain by round 8.

Table 8: Promoted rounds of the two SWA searches above $+130\%$, abridged from the ledger. Gain is the round's single measurement against the episode baseline. The ledger text of GLM-5.2 round 16 repeats the agent's round-12 note; the agent's own round-17 note identifies the round-16 change as the $0.92 \rightarrow 0.94$ memory-fraction step, and the controller's L3 label for that round reflects the stale note (Appendix C.3).

Search	Round	Gain	Promoted change
GLM-5.2/`Claude Code`	9	$+1.66\%$	scheduler admission estimate set to the window-sized KV footprint (`schedule_conservativeness` 0.5)
	12	$+8.10\%$	A100-compatible SWA attention backend (FlashAttention-2) instead of the default FlashInfer kernel
	16	$+130.74\%$	`mem_fraction_static` $0.92 \rightarrow 0.94$ (see caption)
	17	$+210.48\%$	`mem_fraction_static` $0.94 \rightarrow 0.96$
	30	$+211.61\%$	prefill token budget and chunk size; final $+211.72\%$
Opus 4.8/`mini-swe-agent`	1	$+0.02\%$	SGLang window-capped hybrid KV cache (`hybrid_kvcache_ratio` 0)
	10–15	$+13.32\%$	page size, `torch.compile`, mixed chunking, admission (`schedule_conservativeness` 0.3), CUDA-graph capture list
	22	$+134.95\%$	`mem_fraction_static` $0.92 \rightarrow 0.94$
	24	$+138.06\%$	KV page size $16 \rightarrow 32$; final $+137.82\%$

search had lowered the memory fraction to 0.82 in round 3 and lost 15%; no other SWA search records a window-aware KV change. Throughput at the three concurrencies is flat after the change (GLM-5.2: 817, 819 and 820 tokens/s at concurrency 8, 32 and 128), consistent with a prefill-bound regime once all fifty requests fit. Neither artifact meets the online TTFT service-level objective at any tested rate, a secondary SWA metric that the offline promotion scalar does not optimize. The delivered runners themselves are not part of the archive; the description rests on the per-round ledger and the agents' notes.

B.5 Repeat Searches and a Successor Backbone

Two DECODE cells were searched a second time, after the matrix was complete, under the byte-identical DECODE prompt and the same forty-round protocol (`Claude Code` had moved from 2.1.220 to 2.1.267). Figure 10 shows both trajectories and Tables 16 and 17 their round outcomes and cost.

- **Codex 5.6/`mini-swe-agent`**: $+12.34\%$ in the matrix, $+11.64\%$ on repeat. The repeat's gain came in round 17 from raising the chunked-prefill size.
- **GLM-5.2/`Claude Code`**: $+0.03\%$ in the matrix, with 37 of 40 rounds measured; $+11.17\%$ on repeat, from raising SGLang's `stream_interval` in round 27.

The GLM-5.2 pair differs by eleven points with the backbone route, harness family, prompt, workload and device unchanged, but with a newer `Claude Code` version. This difference combines repeat-search variability with any version effect. It is comparable to the matrix's DECODE har-

ness gap (+10.93% under `mini-swe-agent` versus +0.03% under `Claude Code`). A partial matrix for the successor backbone GLM-5.3 points the same way: +12.31% (`Claude Code`) and −0.23% (`mini-swe-agent`) on DECODE, the reverse of GLM-5.2's matrix ordering, and +33.81% and +34.82% on PRECISION; its LONGCTX search stopped after 16 rounds. The two repeats do not estimate search variance or isolate version effects; the observed reversals limit conclusions about a stable harness preference.

B.6 COMPLETE PER-CELL RESULTS

Table 9: All fifty A100 matrix cells. Throughput in tokens/s at each workload's primary concurrency. R_{eff} is the percentage of the 40 rounds with a legal measurement; *Prom.* counts promotions. TTFC is the first round whose best-so-far exceeds the baseline and T90 the first round whose best-so-far reaches 90% of the round-40 champion's gain. Tokens (millions) and cost (USD) are recorded for `Claude Code` (CC) only; mini is `mini-swe-agent`. °unchanged delivery; ‡no model tokens recorded; §repeat of a search that failed final validation.

Backbone	Harn.	Baseline	Peak	Final	G_{final} (%)	R_{eff}	Prom.	TTFC	T90	Tok./USD
DECODE										
GLM-5.2	CC	3,848.98	3,858.95	3,850.14	+0.03	92.5	3	1	7	17.1 / 35.2
GLM-5.2	mini	3,851.33	4,286.46	4,272.35	+10.93	42.5	3	3	26	– / –
Opus 4.8	CC	3,855.44	4,319.36	4,295.14	+11.40	47.5	4	1	1	8.1 / 20.2
Opus 4.8	mini	3,848.17	3,856.85	3,844.61	−0.09	62.5	3	3	17	– / –
Codex 5.6	CC	3,853.92	3,855.75	3,848.04	−0.15	55.0	2	7	16	5.1 / 30.1
Codex 5.6	mini	3,860.81	4,336.93	4,337.42	+12.34	72.5	5	3	7	– / –
DS V4 Pro	CC	3,845.92	3,845.92	3,843.32	−0.07°	0.0	0	–	–	21.7 / 26.6
DS V4 Pro	mini	3,848.29	3,848.29	3,840.37	−0.21°	2.5	0	–	–	– / –
DS V4 Flash	CC	3,857.36	3,857.36	3,845.98	−0.30°	0.0	0	–	–	20.4 / 25.5
DS V4 Flash	mini	3,840.15	3,840.15	3,845.37	+0.14°	0.0	0	–	–	– / –
PRECISION										
GLM-5.2	CC	2,766.90	3,739.73	3,742.30	+35.25	55.0	11	3	10	7.9 / 16.9
GLM-5.2	mini	2,765.59	3,110.03	3,077.20	+11.27	20.0	3	1	1	– / –
Opus 4.8	CC	2,765.66	3,727.59	3,728.73	+34.82	40.0	7	17	26	7.4 / 17.5
Opus 4.8	mini	2,766.07	3,675.45	3,675.80	+32.89	57.5	7	16	22	– / –
Codex 5.6	CC	2,761.70	3,574.55	3,571.45	+29.32	35.0	3	29	37	4.1 / 22.0
Codex 5.6	mini	2,764.81	3,148.98	3,122.94	+12.95	32.5	4	1	27	– / –
DS V4 Pro	CC	2,764.99	2,764.99	2,735.74	−1.06°	0.0	0	–	–	19.8 / 24.3
DS V4 Pro	mini	2,766.00	3,106.67	3,068.97	+10.95	5.0	1	11	11	– / –
DS V4 Flash	CC	2,763.11	2,763.11	2,730.69	−1.17°	0.0	0	–	–	18.4 / 23.1
DS V4 Flash	mini	2,765.65	2,765.65	2,711.77	−1.95°	0.0	0	–	–	– / –
LONGCTX										
GLM-5.2	CC	69.84	73.27	73.31	+4.97	82.5	9	1	7	23.6 / 44.6
GLM-5.2	mini	69.91	73.12	73.14	+4.62	32.5	6	1	17	– / –
Opus 4.8	CC	69.95	72.90	72.90	+4.22	30.0	7	2	34	4.2 / 11.6
Opus 4.8	mini	70.04	71.39	71.41	+1.96	35.0	4	9	16	– / –
Codex 5.6‡	CC	70.17	70.17	70.09	−0.11°	0.0	0	–	–	– / –
Codex 5.6	mini	69.93	72.44	72.45	+3.60	25.0	6	1	8	– / –
DS V4 Pro	CC	69.80	69.80	70.15	+0.50°	0.0	0	–	–	19.7 / 24.4
DS V4 Pro	mini	69.91	69.91	69.93	+0.03°	2.5	0	–	–	– / –
DS V4 Flash	CC	69.94	69.94	69.93	−0.01°	0.0	0	–	–	18.2 / 22.6
DS V4 Flash	mini	69.86	69.86	70.15	+0.42°	0.0	0	–	–	– / –
SMALL										
GLM-5.2‡	CC	35,539.54	35,539.54	35,579.82	+0.11°	0.0	0	–	–	– / –
GLM-5.2	mini	35,596.61	35,596.61	35,317.45	−0.78°	0.0	0	–	–	– / –
Opus 4.8	CC	34,577.76	41,990.17	41,846.85	+21.02	82.5	9	2	37	17.7 / 33.5
Opus 4.8	mini	35,520.14	37,172.73	36,975.63	+4.10	27.5	2	1	31	– / –
Codex 5.6	CC	35,622.16	39,518.94	39,347.07	+10.46	77.5	6	8	25	10.7 / 57.4
Codex 5.6	mini	34,741.87	38,876.79	37,818.88	+8.86	80.0	7	3	31	– / –
DS V4 Pro	CC	35,419.03	38,171.92	38,085.64	+7.53	37.5	4	17	27	18.7 / 38.6
DS V4 Pro§	mini	35,519.34	35,519.34	35,471.47	−0.13°	2.5	0	–	–	– / –
DS V4 Flash	CC	34,460.39	38,552.67	38,316.79	+11.19	37.5	6	5	36	24.2 / 46.4
DS V4 Flash	mini	35,209.06	35,209.06	35,730.96	+1.48°	2.5	0	–	–	– / –
SWA										
GLM-5.2	CC	263.01	819.57	819.85	+211.72	80.0	11	1	17	18.9 / 39.8
GLM-5.2	mini	265.66	285.04	284.76	+7.19	35.0	3	1	3	– / –
Opus 4.8	CC	265.72	289.95	289.82	+9.07	82.5	7	1	10	16.0 / 39.9
Opus 4.8	mini	262.98	626.06	625.43	+137.82	85.0	12	1	22	– / –
Codex 5.6	CC	263.17	291.88	291.78	+10.87	87.5	12	2	20	9.8 / 53.6
Codex 5.6	mini	265.83	273.63	272.80	+2.62	80.0	6	3	29	– / –
DS V4 Pro‡	CC	263.03	263.03	262.97	−0.02°	0.0	0	–	–	– / –
DS V4 Pro	mini	264.09	264.09	265.76	+0.63°	0.0	0	–	–	– / –
DS V4 Flash	CC	264.60	264.60	265.05	+0.17°	0.0	0	–	–	19.4 / 25.5
DS V4 Flash	mini	263.01	263.01	263.12	+0.04°	0.0	0	–	–	– / –

C CONTROLLER AND METRIC SPECIFICATION

C.1 PER-ROUND CONTROLLER RULES

Each round proceeds in five stages. The agent submits one atomic candidate. The controller runs static validation, rejecting candidates that fail to parse, that modify the measurement harness, the accuracy gate, the dataset or the timing code, or that duplicate an earlier submission byte for byte. It then runs the measurement in a fresh process on a dedicated device under a wall-clock timeout. It promotes the candidate to champion if its measurement exceeds the incumbent's, with no margin, and otherwise restores the incumbent bundle; the reported result is decided later, by the accuracy gate and the three final re-measurements (Appendix B.2). Finally it appends the round to an append-only ledger recording the candidate diff, the outcome class, the measurement, the token accounting and the resulting champion hash.

Three recorded controls isolate each measurement from the previous execution and help detect device contention. Every measurement is a separate process launch, including each of the three final repeats. Each launch is refused outright if any compute process is resident on the device, the refusal entering the ledger as an outcome class rather than a measurement taken under contention. The device is checked again on completion to identify contention introduced during measurement.

The controller records one of ten outcome classes per round, and all ten remain in the denominator of every rate reported in the paper. Two mean the candidate produced a legal measurement: *promoted* and *measured, not promoted*. Eight mean it did not: *no effective change* (the submission left the artifact fingerprint unaltered), *duplicate artifact* (byte-identical to an earlier submission), *static rejection* (a parse failure, an invalid artifact, or an edit touching a protected path), *agent failed*, *agent timeout*, *measurement failed*, *measurement timeout*, and *controller interrupted*. We report the classes the controller emits rather than a coarser taxonomy, because the two backbones that rarely reach measurement in Section 4 are separated from the others by which class they land in rather than by how often they fail; Table 15 gives the distribution over the A100 matrix.

Protocol rules. Baselines are measured in place at the start of each episode rather than read from a table, because changes in the software and execution environment can bias performance comparisons (Mytkowicz et al., 2009; Kalibera & Jones, 2013); Appendix B.2 reports how far they drift. Rounds that fail validation, duplicate an earlier artifact, turn out to be no-ops, time out or exhaust memory stay in the denominator of every rate: deleting them would inflate the rates, and scoring them as zero would corrupt the throughput statistics. A champion that is not validated yields no result rather than falling back to its search peak.

Model choice and portability. The served model was fixed within each workload. A sliding-window workload requires a model that uses sliding-window attention, whereas a launch-overhead workload requires a model small enough that host and scheduling costs remain visible. The model must also load across the evaluated software stacks. Weight formats, tokenizers, and attention kernels can require newer drivers, PyTorch versions, or framework releases unavailable on V100; a checkpoint supported on A100 may therefore be unusable on the FP16 compatibility track. We selected established checkpoints with broad stack support, accepting that this choice dates the model set. The resulting differences in model size and task prevent direct comparison of absolute accuracy across workloads.

The accuracy gate. After the final round the champion must clear an accuracy gate before it is eligible to report a result, and only then is it re-measured three times as in Equation 1. The gate bounds *deviation*, not absolute accuracy: the delivered artifact is scored on a fixed MMLU (Hendrycks et al., 2020) subset and must land within 0.03 of the same subset's score under the episode's own baseline configuration. This is deliberate. Each workload serves the model its scenario calls for (Section 3.2), so absolute subset scores are not comparable across workloads, and a common absolute threshold would conflate model capability with changes introduced during optimization. The relative gate tests whether the optimization changes the subset score; it cannot establish correctness outside the tested subset or replace a workload-integrity audit. Table 10 reports the recorded scores: no delivered artifact in the A100 matrix exceeded the bound, and the largest observed deviation is 0.03, on SWA and SMALL.

PRECISION gates each numeric format separately, since a multi-format artifact can preserve accuracy in one precision and lose it in another. Its record carries a score and a deviation for each of BF16, FP8, W8A8, W8A16 and W4A16, and additionally requires format coverage: an artifact that silently drops a required precision fails the gate rather than being scored on the formats it kept.

Table 10: Accuracy-gate records over the A100 matrix, regenerated from the ledgers. The bound is $|\Delta| \leq 0.03$ against the episode's own baseline configuration in every workload; the absolute subset score varies with the model the workload serves and is reported only so that the deviation is auditable. One SMALL cell cleared the gate on its repeat search (Appendix B.1).

Workload	Served model	Formats gated	Subset score	Max $\lvert\Delta\rvert$	Outcome
DECODE	Llama-3 8B	1 (BF16)	0.60–0.62	0.02	10/10 cleared
PRECISION	Llama-3.1 8B	5	0.55–0.61	0.02	10/10 cleared
LONGCTX	Llama-3.1 8B	1 (BF16)	0.54–0.56	0.02	10/10 cleared
SWA	Mistral-7B	1 (BF16)	0.40–0.43	0.03	10/10 cleared
SMALL	Qwen2.5 0.5B	1 (BF16)	0.38–0.41	0.03	10/10 cleared

C.2 METRIC DEFINITIONS

G_{final} is the relative difference between the final and baseline sample medians. Both vary across executions even with fixed configurations and a dedicated GPU; Appendix B.2 quantifies this variability. The ratio of sample medians should not be interpreted as an unbiased estimate of the ratio of mean throughputs. Let s_r be the champion throughput after round r and b the baseline median. Beyond G_{final}, the controller records for every episode

$$\mathrm{AUC}_{40} = \frac{1}{40}\sum_{r=1}^{40} 100\left(\frac{s_r}{b} - 1\right), \qquad R_{\text{eff}} = \frac{N_{\text{measured}}}{40}, \qquad R_{\text{prom}} = \frac{N_{\text{promoted}}}{40}. \quad (2)$$

TTFC is the first round with $s_r > b$, and T90 the first round whose champion reaches 90% of the round-40 champion's gain. Because TTFC can be triggered by a change smaller than the measurement band, it is never read without T90 and AUC_{40} beside it. The main text reports G_{final}, R_{eff} and R_{prom}; TTFC and T90 (computed against the round-40 champion) are in Table 9, and the remaining quantities are released with the ledgers.

C.3 PIPELINE-STAGE CLASSIFICATION RULES

The compilation and kernel stages cover technologies such as `torch.compile` (Ansel et al., 2024), Triton (Tillet et al., 2019), and attention kernels such as FlashAttention (Dao et al., 2022); the memory and scheduling stages cover KV-cache management and request scheduling (Kwon et al., 2023; Zheng et al., 2024; Agrawal et al., 2024). The controller assigns each measured candidate exactly one stage label by a deterministic rule with three tiers. An explicit stage or mechanism declaration in the round narrative takes precedence, and the last declaration wins because the agent's notes accumulate across rounds. Failing that, a keyword audit runs over the narrative joined with the changed paths, in the priority order of Table 11; priority resolves overlapping vocabulary, so that "CUDA Graph memory" classifies as compilation while "KV cache memory" classifies as memory. A round without a valid measured candidate is labelled *none* regardless of what the agent declared. Because the first tier reads the agent's notes, a stale note carries forward: GLM-5.2/`Claude Code` round 16 on SWA is labelled kernel because its note repeated round 12, although the agent's next note identifies a memory-fraction change (Table 8).

A round enters an episode's stage distribution when it produced a measured candidate and the controller assigned it a stage. Rounds labelled *none* and rounds that never reached measurement are excluded, because a stage label on an unmeasured round records only what the agent claimed. For an episode whose labelled rounds distribute over stages as $p_1, \ldots, p_8$, we report stage coverage, the number of stages with at least one such round, and normalized Shannon entropy $H = -\sum_\ell p_\ell \log p_\ell / \log 8$. Over the fifty matrix cells, per-backbone averages are 5.5 stages at $H = 0.67$ for Codex, 3.9 at $H = 0.46$ for Opus and 2.6 at $H = 0.30$ for GLM; DeepSeek V4 Pro reaches 0.9 stages and V4 Flash 0.4, because almost no round of theirs reached measurement (Table 15). Per-harness coverage is 2.88 for `mini-swe-agent` against 2.44 for `Claude Code`.

Table 11: The eight pipeline stages, in the priority order used by the keyword audit, with the terms that trigger each label.

Stage	Name	Triggering terms
L3	kernel	`@triton.jit`, triton kernel, cuda kernel, custom kernel, kernel fusion, `.cu`, cutlass, flashinfer kernel
L2	compilation	`torch.compile`, cuda graph, aot, compile, capture size, graph capture
L5	memory	kv cache, memory pool, `mem_fraction`, `gpu_memory_utilization`, page size, paged, offload, cache layout
L6	scheduling	scheduler, continuous batching, batch size, `max_num_seqs`, `max_num_batched_tokens`, chunked prefill, decode batch, token budget, radix cache
L7	host	tokenizer, chat template, ipc, serialization, pinned memory, cpu thread, affinity, numa, host-side
L8	environment/build	requirements, dependency, build script, cmake, ninja, jit cache, environment, local wheel, extension build
L1	framework/engine	switch engine, framework, sglang, vllm, tensorrt-llm, attention backend, engine backend, runtime selector
L4	system	multi-instance, process topology, worker process, request routing, instance routing, pipeline, system topology

Table 12 gives, per stage, the measured candidates, the promoted ones and their share of all promotions. Scheduling and the engine account for more than half of all promotions between them, kernel changes are accepted at half the rate of engine changes, and the system stage never produced a promotion.

Table 12: Measured and promoted candidates per stage over the fifty-cell A100 matrix (650 measured, 173 promoted).

Stage	L1 engine	L2 compile	L3 kernel	L4 system	L5 memory	L6 sched.	L7 host	L8 env.
Measured	99	82	66	6	134	203	46	14
Promoted	34	24	11	0	29	60	11	4
Acceptance	34%	29%	17%	0%	22%	30%	24%	29%
Share of promotions	19.7%	13.9%	6.4%	0.0%	16.8%	34.7%	6.4%	2.3%

Stage labels support behaviour auditing and search-strategy comparison but are not causal annotations: the label records where an edit was made rather than why it worked, and a candidate labelled scheduling may take effect through an interaction with memory state. The stage-restricted studies and case study provide additional mechanism evidence (Section 4.3; Appendix B.4), but manual interpretation of logs does not establish a causal effect for every labelled round.

C.4 What the Round Count Covers

The 3,401 rounds in Table 13 count the A100 cohorts used for the matrix, protocol and stage analyses, not every record in the archive. Cross-accelerator cohorts, including the additional RTX 4090 BF16 searches and the forty-round Ascend 910C cohort, are accounted for separately by registered conditions and validation status in Section D.4; they are not included in this subtotal. The A100 pilot and supplementary backbone sweep are also outside it. A further six-condition export repackages runs that already belong to the matrix (the four SWA cells, the GLM-5.2/`Claude Code` PRECISION cell and the DeepSeek V4 Flash/`mini-swe-agent` DECODE cell) and adds no searches or rounds. The two repeat DECODE searches contribute the 80 rounds listed below.

Table 13: A100 round census for the matrix, protocol and stage analyses. Cross-accelerator coverage is reported separately in Appendix D.4.

Cohort	Cells	Rounds
A100 matrix, five workloads × five backbones × two harnesses (Section 4)	50	2,000
Further SMALL searches: failed-validation first attempt, two searches under DeepSeek's own harness (Appendix B.1)	3	120
Legacy SMALL cohort with recorded peaks (Appendix E)	15	600
Repeat searches of two DECODE cells (Appendix B.5)	2	80
Stage-restricted matrix from the reference runner (Appendix B.3)	8	320
Stage-restricted arms from the exhausted runner (Appendix B.3)	3	161
Joint search from the exhausted runner (Appendix B.3)	1	120
Total	82	3,401

C.5 Artifact Fingerprinting

Every final bundle is fingerprinted with a canonical SHA-256 over the runner source, the effective configuration, the runtime manifest and the local dependency set. Equal fingerprints mean exact content equality. For bundles that differ, we report a line-set Jaccard distance as a descriptive magnitude only; it is not evidence of semantic equivalence or difference.

D Complete Results and Supporting Analyses

D.1 Pinned Backbones and Harness Versions

The frozen matrix configuration specifies each backbone route, harness and reasoning-effort setting. Episode summaries additionally record the harness version and task-prompt SHA-256 (Table 14). All five backbones were accessed through one commercial routing endpoint. A route identifies the requested model, but does not independently verify the weights that served a request. Searches outside the matrix are kept separate even when they use the same route (Appendix E). Within each workload, matching prompt hashes confirm that the two harnesses received a byte-identical task statement. The SMALL prompt additionally discloses the public reference throughput (Appendix B.1).

Table 14: Backbones, harness versions and prompt hashes for the A100 matrix. Each backbone was called through a fixed route on one routing endpoint, recorded with every episode in the released ledgers. Reasoning effort is set where the endpoint exposes it. The prompt hash is the SHA-256 the controller recorded for the task statement, shown to eight hex digits; it is identical across the two harnesses within each workload.

Backbone	Reasoning effort	Harnesses
DeepSeek V4 Pro	default	`mini-swe-agent`, `Claude Code`
DeepSeek V4 Flash	default	`mini-swe-agent`, `Claude Code`
GLM-5.2	default	`mini-swe-agent`, `Claude Code`
Opus 4.8	default	`mini-swe-agent`, `Claude Code`
Codex 5.6	high	`mini-swe-agent`, `Claude Code`
Harness versions		
`mini-swe-agent`	2.4.5, forty-round controller protocol	
`Claude Code`	2.1.220, 24-tool limit, 900 s agent timeout	
Task-prompt SHA-256 prefix, identical across harnesses		
DECODE	`40314955`	
PRECISION	`f71a3014`	
LONGCTX	`42dec57d`	
SMALL	`97b672f8`	
SWA	`37260df0`	

D.2 Round Outcome Classes

Round outcomes distinguish unsuccessful optimization from a failure to reach measurement. Table 15 reports the 2,000 matrix rounds, including failed and unchanged submissions; Table 16 reports the two repeat searches.

The two DeepSeek backbones produced 20 and 16 measured rounds in 400, all but four of them on SMALL; on the other four workloads their budget went to rounds that changed nothing (374 of 640) and to agent failures. Their measured candidates include 25 that did not improve on the incumbent, so the protocol did observe them optimizing, mostly on SMALL; elsewhere there is too little to judge. The three other backbones spend 32–42% of their rounds on candidates that were measured and not promoted, the outcome that carries information about search quality.

The harnesses differ in both effective-round ratio and failure composition. `Claude Code` loses budget predominantly to rounds that attempted something and did not finish (354 agent failures and 113 timeouts), `mini-swe-agent` to rounds that finished and proposed nothing distinct (398 no-effective-change and 53 duplicates). 257 of the `Claude Code` failures and timeouts recorded no model tokens at all (Appendix B.1); `mini-swe-agent` records no per-round usage, so its failures cannot be split the same way.

Table 15: Round outcome classes over the A100 matrix, 50 cells × 40 rounds. *Eff.* is the effective-round ratio and *Prom.* the promotion ratio, each over the row's full denominator. Twenty rare rounds are omitted from the columns and retained in the denominators: eleven static rejections or invalid artifacts, four protected-path edits, four measurement timeouts and one controller interruption.

	Measured		No valid measurement					Rates	
	Prom.	Not prom.	No-op	Dup.	Agent fail	Timeout	Meas. fail	Eff.	Prom.
By backbone, 400 rounds each									
GLM-5.2	49	127	43	24	71	65	20	44.0%	12.2%
Opus 4.8	62	158	4	30	80	50	10	55.0%	15.5%
Codex 5.6	51	167	16	1	49	87	26	54.5%	12.8%
DeepSeek V4 Pro	5	15	188	0	142	37	11	5.0%	1.2%
DeepSeek V4 Flash	6	10	259	0	104	8	5	4.0%	1.5%
By harness, 1,000 rounds each									
`Claude Code`	101	268	112	2	354	113	41	36.9%	10.1%
`mini-swe-agent`	72	209	398	53	92	134	31	28.1%	7.2%

Table 16: Round outcome classes for the two repeat searches on DECODE, forty rounds each.

Outcome	GLM-5.2 × `Claude Code`	Codex 5.6 × `mini-swe-agent`
Promoted	3	5
Measured, not promoted	25	17
Measured subtotal	*28*	*22*
No effective change	0	12
Duplicate artifact	0	4
Measurement failed	6	0
Agent timeout	6	1
Agent failed	0	1
Effective-round ratio	70.0%	55.0%
Promotion ratio	7.5%	12.5%

D.3 TOKEN AND COST ACCOUNTING

The controller records token consumption and provider cost for every `Claude Code` round. Over the twenty-five `Claude Code` cells of the matrix this is 331M billed tokens and $679 at recorded prices. The distribution is not aligned with outcome: the five DeepSeek V4 Flash cells consumed 100.6M tokens ($143) for one success, and the five Codex cells 29.7M ($163) for three. Per-cell figures are in Table 9. Table 17 reports the two repeat searches.

Table 17: Token and cost accounting for the two repeat searches on DECODE. Cost is the controller-recorded provider charge; it is token accounting under one provider's prices and not a normalized cross-provider comparison.

Search	Tokens	Cost (USD)	Validated gain	Cost / point
GLM-5.2 × `Claude Code`	10,721,428	29.87	+11.17%	$2.67
Codex 5.6 × `mini-swe-agent`	4,008,891	11.75	+11.64%	$1.01
Total	14,730,319	41.61		

We report cost as token accounting rather than as a provider price ranking: prices are not normalized across providers, and a recorded zero cost can mean missing usage rather than free usage. For the same reason there is no cost column in the main results table.

D.4 COVERAGE, CALIBRATION AND CROSS-ACCELERATOR DETAIL

What the cross-accelerator arm supports. Gains normalize each final measurement by its own episode baseline; this does not remove differences in precision, harness or starting configuration. V100 has four candidate finals supported by the saved accuracy, repeat and artifact records, all on SMALL under `mini-swe-agent` and within 1.5% of baseline. Their median gain is −0.25%; the remaining sixteen registered conditions have no validated candidate delivery. One further V100 condition (GLM-5.2/`mini-swe-agent`, SMALL) passed the accuracy gate but was excluded be-

cause rounds 38–39 changed the measured workload through a caching and deduplication path; its search peak read 1,980,934 tokens/s against a 7,761 baseline. This is a saved-record audit, not a fresh execution of the complete protocol.

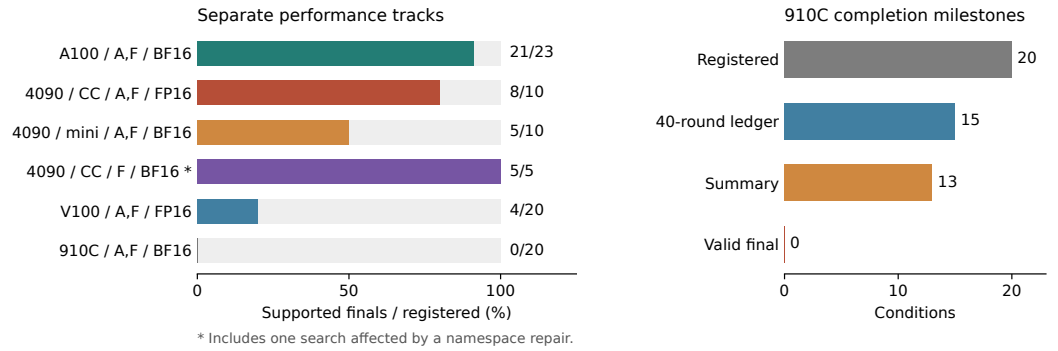


Figure 7: Cross-accelerator coverage: conditions with supported final measurements over registered conditions in each track (Table 5); every track registers DECODE and SMALL except the additional RTX 4090 cohort (∗), which covers only SMALL and includes one search with a namespace repair. The right panel partitions the twenty Ascend conditions by the validation states in Table 19. The Ascend zero denotes no valid final delivery, not zero throughput.

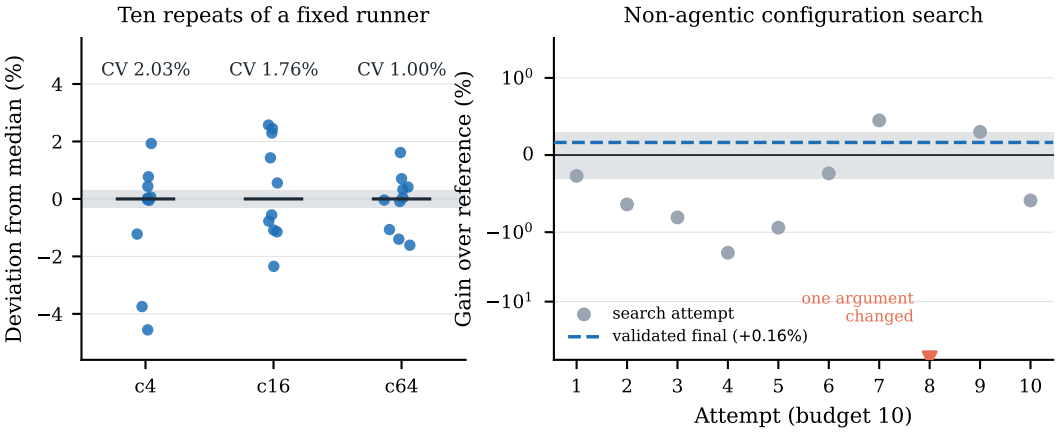


Figure 8: **Left:** ten independent re-measurements of one fixed runner on an idle A100, as deviation from the median at each concurrency. **Right:** the non-agentic configuration search over the engine's exposed arguments under a budget of ten attempts, with its accuracy-gated three-repeat final. The shaded $\pm 0.3\%$ band is a visual reference, not an uncertainty interval. The measured mean absolute deviation of a three-repeat median is 0.39%, and its resampled 95% interval is about $\pm 1.5\%$ (Appendix B.2). The eighth attempt differs from the ninth only in the CUDA-graph batch bound and costs 65.6% of throughput while remaining a legal, accuracy-valid configuration.

An additional RTX 4090 BF16 cohort. Five `Claude Code` searches on SMALL each retain 40 rounds and an accuracy-gated three-repeat final (Table 18). All use Qwen2.5-0.5B-Instruct in BF16, primary concurrency 64 and one RTX 4090, with offline output-only tokens/s as the metric. Episode baselines range from 36,511 to 38,342 tokens/s, distinct from both original tracks. None of the five final medians exceeds its baseline: gains range from -1.20% to -0.26%, with median -0.55%. This cohort therefore provides no observed positive final gain; it does not establish an optimization ceiling or a significant regression for each model.

The Codex 5.6 episode includes a repair to writable namespace mounts during search, with the first eight rounds already invoked at the deployment boundary. Those rounds remain in its budget. Opus reached a tool-turn limit and later incurred API failures; its search peak of 38,531.85 tokens/s was not retained by the 36,885.62 final median. These measured outcomes are retained with their execution qualifications, not treated as five fault-free replications. Excluding the Codex episode gives a four-condition median of -0.41%, still with no positive final gain. The Pro and Opus searches of

this cohort called routes that differ from the matrix routes of Table 14, and neither is equated with its core counterpart.

Table 18: Supported SMALL finals behind the V100 and additional RTX 4090 summaries. Throughput is offline output-only tokens/s at concurrency 64; gains use the same row's baseline. V100 uses FP16 and `mini-swe-agent`; RTX 4090 uses BF16 and `Claude Code`. Each final is a median of three independent measurements after accuracy checking. *Alt. route*: a route that differs from the matrix route for the same backbone; †: namespace repair during search; ‡: tool-turn limit and subsequent API failures. All recorded rounds are retained.

Accelerator	Backbone	Baseline	Final	Gain
V100	Codex 5.6	7,703.17	7,674.55	-0.37%
V100	Opus 4.8	7,703.17	7,692.94	-0.13%
V100	DeepSeek V4 Flash	7,540.40	7,433.44	-1.42%
V100	DeepSeek V4 Pro	7,703.17	7,708.23	$+0.07\%$
RTX 4090	DeepSeek V4 Flash	37,885.55	37,784.11	-0.27%
RTX 4090	DeepSeek V4 Pro (alt. route)	37,038.73	36,594.32	-1.20%
RTX 4090	Codex 5.6†	38,342.02	38,013.86	-0.86%
RTX 4090	GLM-5.2	36,511.09	36,310.00	-0.55%
RTX 4090	Opus 4.8 (alt. route)‡	36,980.03	36,885.62	-0.26%

Ascend 910C validation. The Ascend 910C cohort registers 20 conditions: DECODE/SMALL, five backbones and two harnesses, one search each, with the Opus conditions resubmitted through an alternate Opus route after gateway failures. Seventeen conditions completed their search with a summary, and none of the seventeen has a valid final. Table 19 separates the states.

Five summarized runs changed the runner, and all five also modified the protected configuration. On SMALL, GLM-5.2/`mini-swe-agent` recorded a candidate median of $25{,}697.39$ against a $7{,}522.95$ baseline, a nominal $+241.59\%$; GLM-5.2/`Claude Code` recorded $20{,}240.98$ against $7{,}999.25$ ($+153\%$) with a large set of untracked kernel-output files; Codex 5.6/`mini-swe-agent` recorded $3{,}275.56$ against $8{,}407.36$ and its accuracy score fell from 0.38 to 0.00. On DECODE, the two Opus runs recorded $+3.5\%$ (`mini-swe-agent`) and $+0.9\%$ (`Claude Code`). None enters the gain statistics. The other twelve summarized conditions retained the starting runner and failed final accuracy execution. On this stack, every candidate that changed the runner also changed the evaluation contract; these nominal ratios cannot be treated as gains under the registered evaluation contract.

Table 19: Mutually exclusive states of the 20 registered Ascend 910C conditions in the forty-round cohort. No row provides a valid final gain; incomplete conditions remain in coverage accounting.

State	Cells	Evidence and interpretation
Summary; changed runner	5	SMALL: GLM-5.2 (`mini-swe-agent`, `Claude Code`), Codex 5.6 (`mini-swe-agent`); DECODE: Opus 4.8 (`mini-swe-agent`, `Claude Code`). All five modified the protected configuration, so integrity fails.
Summary; starting runner retained	12	Runner identical to the starting runner; final accuracy execution failed; no valid candidate final.
40 rounds; no summary	1	DECODE, DeepSeek V4 Pro/`mini-swe-agent`: forty no-op rounds; stopped after a runtime-selection integrity failure.
Not started	2	SMALL, Opus 4.8 under both harnesses: model path unavailable at resubmission.
Registered total	**20**	**17 completed summaries; 0 valid finals.**

The Ascend ledger lacks per-round primary scores, layer labels, explicit promotion events and token costs. Consequently, matrix-style gain trajectories, promotion rates and stage summaries cannot be reconstructed from it; missing events are not counted as zero promotions. An older 10-measurement, 80-step Ascend cohort with a different budget and visibility setting is not pooled with these conditions. Across all devices, incomplete coverage and differences in software, precision, harness, starting point and instrumentation preclude a hardware ranking.

D.5 STAGE-RESTRICTED DETAIL FROM THE EXHAUSTED BASELINE

Backbone and prompt differences prevent this block from isolating the effect of the starting runner. Appendix B.3 reports the reference-start study and the qualifications needed to compare them.

Table 20: The right-hand block of Figure 5 in full: stage-restricted optimization on SMALL, A100, from the exhausted baseline. Each arm opens one stage and freezes the rest. The backend-switch and joint-search gains have validated measurements. A displayed 0.00% in the other arms denotes no delivered improvement, not a measured zero-effect estimate. The local three-repeat calibration gives a 95% interval of about ±1.5% (Appendix B.2).

Stage opened	Backbone	Rounds	Gain	Outcome
L4+L5 system, memory	Kimi K2.7	60	0.00%	58 changes, none beat baseline; mean −10.72%
L6 scheduling	DeepSeek V4 Flash	60	0.00%	mean −10.2%; no improvement found
L1 engine backend	backend-switch arm	–	−65.61%	legal and accuracy-valid; slower
L3 kernel (CUDA/CUTLASS)	Kimi K2.7, DeepSeek V4 Pro	41	0.00%	no evaluable candidate produced
Model algorithm (aux.)	model-algorithm arm	–	0.00%	best candidate +1.50%, not delivered
All stages jointly	DeepSeek V4 Flash	120	+0.05%	116/120 measured; peak +0.44%

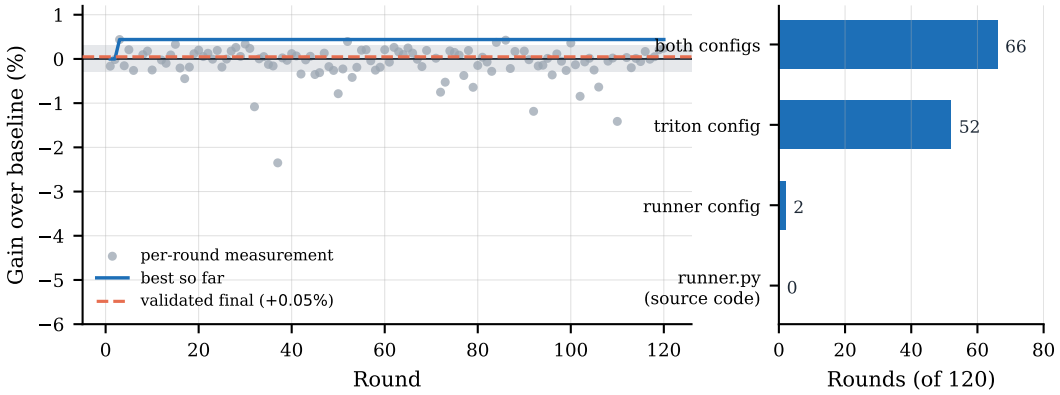


Figure 9: The 120-round joint search from the exhausted SMALL baseline, with the runner source, the engine configuration and the kernel configuration all editable. **Left:** per-round measurements, best-so-far and the accuracy-gated three-repeat final. **Right:** which files each round changed. The runner source was never modified.

D.6 SUPPORTING FIGURES

Harness distribution shape. The observed `Claude Code` distribution includes both the largest gain and several unchanged deliveries. These shapes describe single searches and do not estimate the variability of either harness.

Artifact divergence. On DECODE, PRECISION, LONGCTX and SWA, the ten cells of a workload yield seven, eight, six and seven distinct final artifacts, counting all unchanged deliveries as one; the mean line-set Jaccard distance between the two artifacts of a same-backbone pair is only 0.037. Small textual differences can accompany large throughput differences. Artifact distance therefore complements, but does not replace, an account of the changed execution path.

E LEGACY COHORT WITH RECORDED SEARCH PEAKS

The legacy cohort contains fifteen forty-round searches on SMALL (Qwen2.5 0.5B, BF16, 2,048-token context, 256-token output, primary concurrency 64). It used the same controller as the A100 matrix, but the backbone combinations are incomplete and partly repeated. Four searches lack a valid final accuracy-gate receipt; two of these never promoted a candidate, indicating a delivery or validation failure rather than evidence of an accuracy–throughput trade-off. This unbalanced cohort contributes no per-factor statistics. It supports the protocol audit: Codex 5.6/`Claude Code` reached a +16.71% search peak without a validated final, and two validated searches lost more than five percentage points between peak and final.

The peak-versus-final comparison itself is not confined to this cohort. The per-round ledgers record the search peak for every cell of the A100 matrix as well, so Figure 4 is drawn over all sixty-one cells for which both quantities exist: the fifty matrix cells and the eleven validated legacy traces.

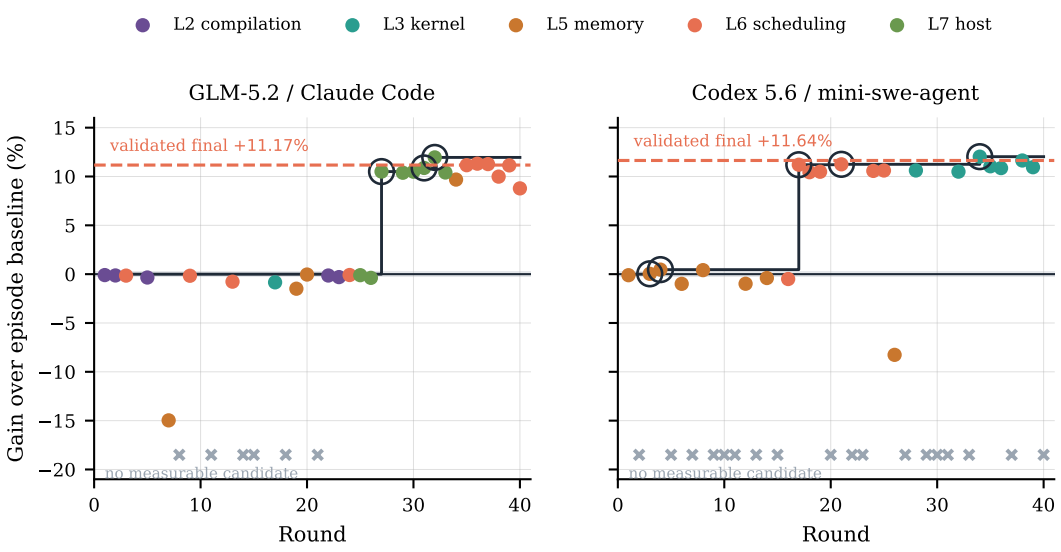


Figure 10: The two repeat searches on DECODE (Appendix B.5), under the matrix's DECODE prompt, reaching $+11.17\%$ and $+11.64\%$. Dots are per-round measurements coloured by the stage the controller assigned; circled dots are promotions; the step line is the best-so-far champion; crosses at the bottom mark rounds that produced no measurable candidate. Each search gains mainly from one engine argument, with a different argument selected in each run.

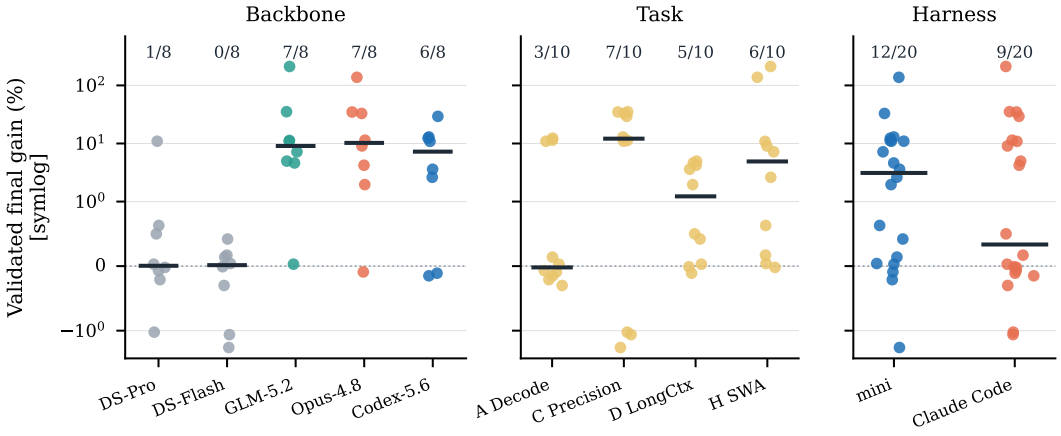


Figure 11: Descriptive gain distributions by workload, backbone and harness. Each dot is one of the forty cells on DECODE, PRECISION, LONGCTX and SWA (the SMALL column is in Table 3), each bar a group median, and each annotation the number of successes in that group (a changed artifact above $+2\%$; Section 3.1). Task labels A, C, D and H denote DECODE, PRECISION, LONGCTX and SWA. The vertical axis is symmetric-log and shared across panels.

The comparison includes the variability observed across all five workloads rather than assuming a common noise level.

Backbones are named as in the matrix. Four rows used two further routes of the Codex 5.6 family that the matrix does not use (variants 1 and 2), and one Opus 4.8 row used a route that differs from the matrix's (alt. route). Rows that share a backbone and route with the matrix are still never pooled with it: the cohort ran at a different time through the same routing endpoint, and without routing evidence a route does not establish which weights answered.

Two rows have a search peak equal to their baseline, meaning the episode never promoted a candidate, and re-measurement nonetheless assigned them finals of $+0.88\%$ and -2.12%. Their spread illustrates measurement variation across unchanged deliveries; two episodes do not define a confidence interval for the workload.

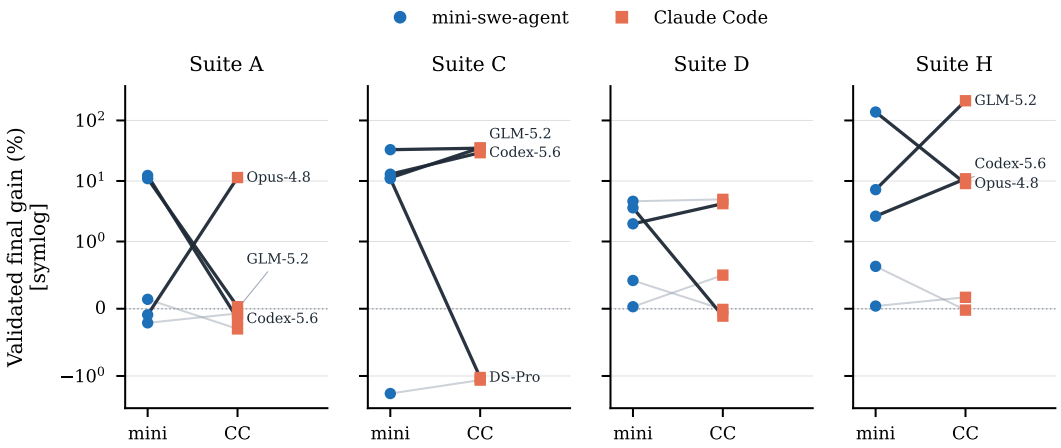


Figure 12: Validated gain of each backbone under the two harnesses, one panel per workload for DECODE, PRECISION, LONGCTX and SWA. Dark lines mark pairs separated by more than one percentage point; crossing lines are harness preference reversals, and they appear in all four panels. Panel labels A, C, D and H denote those four workloads in order. The vertical axis is symmetric-log.

Table 21: Legacy cohort. Only rows with a final have an accuracy-gated three-repeat final; the rest are reported as having no result rather than being assigned their search peak.

Backbone	Harness	Baseline	Peak	Rd.	Final	Status
Opus 4.8	`Claude Code`	34945.66	42256.55	17	40228.57	+15.12%
Opus 4.7	`mini-swe-agent`	35450.08	35450.08	0	–	no final
Opus 4.8 (alt. route)	`mini-swe-agent`	34906.38	35670.01	5	–	no final
DeepSeek V4 Flash	`Claude Code`	34878.08	34878.08	0	35185.21	+0.88%
DeepSeek V4 Flash	`Claude Code`	33498.02	35682.82	20	35159.89	+4.96%
DeepSeek V4 Flash	`mini-swe-agent`	34971.60	34971.60	0	–	no final
DeepSeek V4 Flash	`mini-swe-agent`	34906.76	34906.76	0	34166.42	−2.12%
GLM-5.2	`Claude Code`	35228.29	45557.79	10	45112.46	+28.06%
Codex 5.6 (variant 1)	`Claude Code`	34307.95	37101.44	9	36981.13	+7.79%
Codex 5.6 (variant 1)	`mini-swe-agent`	35607.73	37538.16	12	37401.28	+5.04%
Codex 5.6	`Claude Code`	34690.31	40488.16	9	–	no final
Codex 5.6	`mini-swe-agent`	34211.45	35377.97	40	33620.65	−1.73%
Codex 5.6 (variant 2)	`Claude Code`	34113.93	42302.57	35	42200.36	+23.70%
Codex 5.6 (variant 2)	`mini-swe-agent`	34375.58	38759.76	35	39360.88	+14.50%
Qwen3.7-Plus	`Claude Code`	34396.07	39472.48	33	39381.47	+14.49%

F LIMITATIONS

Limited replication. Most matrix cells contain one search. Three final measurements characterize execution variability, but do not estimate variation across agent searches. In the two repeated DECODE cells, final gains differed by 0.7 and eleven percentage points. The latter comparison also changed the `Claude Code` version (Appendix B.5). Neither comparison isolates stochastic search variance. We report descriptive agent comparisons without confidence intervals for factor effects or significance tests; the calibration intervals concern repeated execution of a fixed runner. Replicated searches across several workloads are needed to assess stable backbone and harness differences.

Uneven cross-device coverage. The A100 matrix crosses backbones and harnesses under the same controller protocol. Additional device tracks differ in precision, runtime, harness and starting configuration. The additional RTX 4090 BF16 cohort covers only SMALL; one search includes a namespace repair, and the Opus search includes API failures. V100 has four supported candidate finals out of twenty registered conditions, all on SMALL under `mini-swe-agent`. These records describe the evaluated deployments but cannot isolate device effects or support a hardware ranking.

No validated Ascend final. The 910C cohort registers twenty conditions and contains seventeen completed summaries, with no valid final delivery. Five summarized runs changed the runner and also modified protected configuration; twelve retained the starting runner and failed final accuracy execution. One further condition has a forty-round ledger without a summary, and two did not start. Missing per-round scores, promotion events, stage labels and token accounting prevent reconstruc-

tion of several diagnostics. These failures concern the recorded deployment and validation process. They do not establish an optimization limit for the accelerator or backbones, and older Ascend experiments with different budgets cannot fill the missing conditions.

Confounded starting-point study. The reference-start arms used GLM-5.2/`Claude Code`, whereas the exhausted-start arms used Kimi K2.7 and the DeepSeek backbones, with different prompts. The studies were run only on SMALL. The joint exhausted-start search produced 116 valid measurements in 120 rounds, but its lack of improvement still cannot isolate starting-point effects from agent effects. The supported observation is an absolute ordering: the best reference-start arm delivered $42{,}075$ tokens/s, below the tuned runner's $46{,}390$ baseline. A matched study should repeat both starting points with the same backbones, prompts and budgets, then test additional workloads. We use "exhausted" to name the previously tuned runner; the term does not establish a global optimization ceiling.

Limited non-agentic comparison. The configuration search used ten attempts, compared with forty or 120 agent rounds, and started from a separate $38{,}271$ tokens/s runner. Its $+0.16\%$ final gain cannot be treated as a budget-matched comparison with the agent searches. Exposed arguments accounted for both repeated DECODE gains, so systematic configuration search is a relevant baseline. Evaluating it under matched starts and budgets is necessary to determine the additional value of code-generating agents (Yeon et al., 2026).

Missing stage-study finals. Six of eight reference-start arms lack a validated final. Five had search champions between $+12.05\%$ and $+17.47\%$; the system arm never exceeded baseline. Five finalization attempts failed because the artifact was absent, and one because the device was not idle. These arms contribute missing outcomes, not zero gains. Re-validation would require both the recorded fingerprints and the corresponding artifact files; a fingerprint cannot recover a missing artifact.

Operational failures and model identity. No-op submissions, duplicate artifacts, harness failures, timeouts and missing final artifacts have distinct causes. For example, 257 `Claude Code` rounds recorded no model tokens, and the archive lacks gateway responses needed to distinguish endpoint and harness failures. Moreover, all matrix backbones were requested through one commercial routing endpoint. The recorded route does not independently verify the serving weights. We retain affected rounds in the denominators and avoid attributing all operational failures to backbone capability.

Baseline and budget choices. Gains are relative to configured serving engines, not uniformly expert-tuned references. Their size therefore depends on the remaining headroom. The forty-round budget also affects outcomes: only 46% of successful searches reached 90% of their eventual champion's gain by round 20 and 77% by round 30. At least two important improvements occurred after round 20 (Figure 2c). The study does not sweep budgets or establish the gains attainable with more search.

Accuracy and workload integrity. The accuracy gate bounds MMLU-subset deviation from the episode baseline. It does not guarantee correctness on other inputs or preserve every serving metric. The V100 caching and deduplication episode passed this gate but changed the measured workload and was excluded by an integrity audit. The two largest SWA artifacts also failed the online time-to-first-token target at all tested rates despite improving the offline throughput objective. These distinctions limit what a validated throughput gain establishes.

Mechanism evidence and cohort scope. Stage labels derive from declared mechanisms and changed paths; they are audit labels rather than causal annotations. Stale notes can misclassify a round. The SWA case study relies on ledgers and agent notes because its delivered runners are not in the archive. The legacy cohort is unbalanced and used only for the protocol audit. In the main matrix, SMALL differs in its sandbox, prompt and disclosure of the public reference throughput. These differences prevent attributing every cross-workload contrast to model size or serving bottleneck.

G BROADER IMPACT

Agents that find valid serving optimizations could reduce deployment effort and computation per served token. This study measures throughput, however, and does not directly measure energy savings or production operating cost. A practical risk is accepting a speedup that changes the workload, reduces quality or fails outside the tested setting. Independent measurements, workload-integrity checks and reviewable artifacts help identify these failures. Deployment still requires validation against the intended workload and service-level objectives.