DELTASKILL: RESIDUAL SKILL COMPILATION AND EVIDENCE-GUIDED CONTROL FOR LONG-HORIZON TERMINAL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Skills have enabled a new class of language-model agents that draw on modular procedures and execution strategies to solve complex tasks. However, poorly adapted skill use can prolong task execution and waste tokens. In this work, we study adaptive skill use in language-model agents, combining capability-aware residual skill compilation with evidence-guided execution control. We call this framework **DeltaSkill**. We evaluate DeltaSkill against seven baselines on three long-horizon benchmark tasks, comparing execution time, task scores, and token consumption. Our findings reveal that lower execution costs can accompany improved or near-best solution quality. These benefits include improved reward on Generals, near-best quality on Spot with 57.2% fewer execution tokens and 84.7% less recorded wall time than Structured Local, and complete artifact delivery on DuckDB, where the seven baseline executions remain incomplete. To our knowledge, this is the first study to couple receiver-conditioned residual skill compilation with the retention of validated task artifacts and evidence-guided continuation for long-horizon terminal agents. Finally, we outline directions for extending DeltaSkill to continual capability profiling and cross-task skill adaptation, toward more efficient and generalizable agents.

1 INTRODUCTION

Large language model (LLM) agents increasingly use skills to access procedural knowledge and guide task execution. These skills package domain knowledge and operational guidance into modular resources. To solve a task, agents can retrieve relevant procedures from skill libraries (Wang et al., 2023; Kang et al., 2026), compose them into coordinated execution workflows (Li et al., 2026a), and adapt the instructions to task requirements and execution environments (Li et al., 2026b). The potential of skill-based agents to tackle complex tasks has attracted growing interest in the research community.

However, skill use can also introduce unnecessary resource consumption. Recent studies have shown that directly injecting skills can increase token usage without improving task success (Han et al., 2026), while mismatches with task requirements, execution interfaces, or receiver capabilities can lead to repeated attempts and additional inference (Li et al., 2026b; Chen et al., 2026).

In this work, we argue that **with capability-aware residual skill compilation and evidence-guided execution control, agents can achieve high-quality solutions at lower execution cost.** We introduce **DeltaSkill** and highlight four core challenges that distinguish our focus on efficient execution from skill provision alone:

1. **Profile–receiver mismatch.** A profile that misrepresents the receiving agent's capabilities can lead to missing guidance or unnecessary support.
2. **Redundant skill context.** Overlapping skill instructions lengthen the context, increasing input-token consumption whenever they are included in a model request.
3. **Unnecessary model requests.** Repeated checks or redundant revisions can trigger additional model calls without resolving unmet requirements or improving the solution.

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.

4. **Ineffective stopping.** Without progress-aware stopping, execution can continue through idle waits or repeated cycles after progress stalls and no actionable work remains.

These challenges raise a critical question: can we ensure that skill-guided agents improve solution quality without wasting tokens or prolonging execution? The answer is far from certain, as poorly adapted guidance can add overhead throughout execution. For instance, a scheduling agent may repeatedly process overlapping planning instructions, increasing input tokens without receiving additional support. Similarly, an agent with a valid solution may continue requesting checks and revisions after progress stalls, consuming resources without improving its output.

To systematically investigate these challenges, we examine four dimensions: (1) In **profile matching**, we compare matched and mismatched profiles across two receivers. (2) In **skill redundancy**, we trace instruction changes during compilation alongside observed token use. (3) In **request efficiency**, we analyze request counts, input sizes, and candidate progress along execution traces. (4) In **stopping control**, we compare adaptive stopping with fixed-budget continuation under compiled and raw skill inputs.

Our empirical analysis reveals that adaptive skill use can improve the quality–cost trade-off of long-horizon agents. DeltaSkill achieves improved or competitive solution quality with lower execution overhead, showing that efficiency gains need not require sacrificing performance. Additionally, the component analysis shows that the quality–cost trade-off of stopping depends on the supplied support, highlighting the value of jointly adapting procedural guidance and governing continued execution.

The key contributions of our study can be summarized as follows:

- **Receiver-conditioned skill compilation.** We formulate residual procedural support relative to task requirements and configuration-specific capability evidence, and organize overlapping instructions through shared control and explicit compilation checks.
- **Evidence-guided execution control.** We connect candidate validation and artifact retention to continuation decisions, allowing further exploration around a validated deliverable.
- **Implications and future directions.** Our findings highlight the potential of adaptive skill use to improve the quality-cost trade-off, and motivate future research on continual capability profiling and cross-task skill adaptation.

2 Related Work

Skill libraries and adaptation. Voyager develops executable skills; AgentSkillOS organizes and orchestrates skill collections (Wang et al., 2023; Li et al., 2026a). SkillRet studies retrieval, and SWE-Skills-Bench uses paired skill comparisons (Kang et al., 2026; Han et al., 2026). SkillAligner adapts retrieved skills to task requirements and execution environments, while SkVM combines model–harness-specific compilation with runtime adaptation (Li et al., 2026b; Chen et al., 2026). DeltaSkill couples receiver-conditioned residual compilation with validated task-artifact retention and evidence-guided continuation.

Prompt and context management. LLMLingua and LongLLMLingua compress prompts (Jiang et al., 2023a;b); LLMLingua-2 learns token selection for compression (Pan et al., 2024). MemGPT manages information across memory tiers (Packer et al., 2023). DeltaSkill makes procedural dependencies and protected content explicit when transforming skills. Unresolved coverage leads to retention; duplicate relations guide consolidation. These transformations complement runtime context management.

Feedback and evaluation. ReAct interleaves reasoning and actions, and Reflexion uses feedback to inform subsequent trials (Yao et al., 2022; Shinn et al., 2023). Our execution design connects public evidence to candidate validity, retained-artifact identity, and continuation decisions. AgentBench, WebArena, and SWE-bench evaluate interactive and software tasks (Liu et al., 2023; Zhou et al., 2023; Jimenez et al., 2023); SEA-Eval examines success and token use across task sequences (Jiang et al., 2026). We evaluate skill adaptation through both delivered quality and execution resources. Appendix L provides further comparisons.

3 METHOD

3.1 DELTASKILL OVERVIEW

In this work, a skill is a structured package of procedural instructions and associated resources for agent execution. Let A denote the agent configuration, T the visible task contract, $\mathcal{S}$ the source skills, and h_t the interaction prefix. The contract specifies deliverables, public checks, and constraints. Given configuration-bound capability evidence P_A, compilation returns

$$(\mathcal{S}^{\Delta}_{A,T}, \Pi) = \mathrm{Compile}(\mathcal{S}, T, P_A), \tag{1}$$

where a procedural obligation supplies a required procedure or check, $\mathcal{S}^{\Delta}_{A,T}$ contains the obligations supplied to the receiver, and Π links compilation decisions to sources and evidence.

Figure 1 summarizes the framework. DeltaSkill first establishes configuration-bound capability evidence, then compiles residual procedural support from the task contract and that evidence. During execution, public checks, artifact retention, and visible progress govern candidate continuation. Routing and global linking may invoke a model; profile updates and guarded unit decisions are deterministic. Appendices A and M give the interfaces and detailed process views.

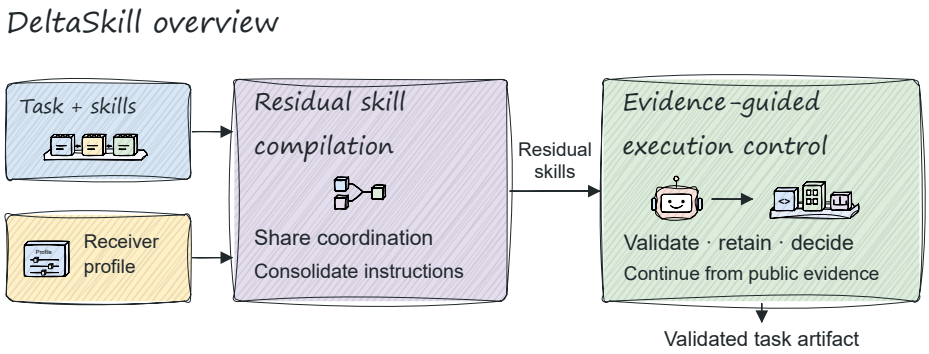


Figure 1: Two coupled components: task- and receiver-conditioned compilation supplies residual procedural support; evidence-guided execution validates candidates, retains eligible artifacts, and governs continuation.

3.2 CAPABILITY PROFILING

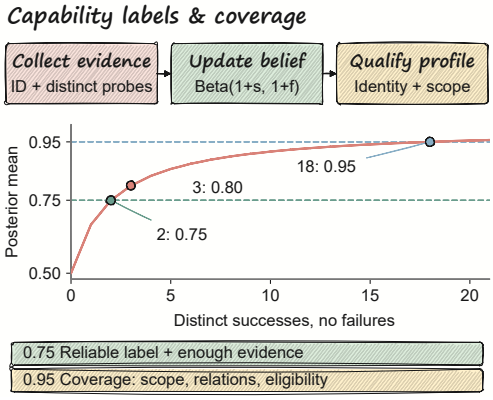


Figure 2: All-success Beta$(1, 1)$ curve. Reliability (0.75) differs from removal coverage (0.95), which also requires scoped checks.

We first construct a capability profile for the receiving configuration. It records eleven control capabilities: planning, batching, waiting, polling, verification, repair, termination, routing, state reuse, precedence, and fallback. The profile identity $K(A)$ contains the agent, model, harness, tool-schema, and visible-contract hashes. Task-independent probe variants run through the configured adapter; a registered verifier summarizes each seed into one observation, and the resulting evidence conditions compilation (Figure 3).

Each capability starts with Beta$(1, 1)$; successes and failures update α and β, giving posterior mean $\widehat{p}_c = (1 + n_c^+)/(2 + n_c^+ + n_c^-)$. Replayed identifiers add no count. The updater labels evidence-free or under-sampled capabilities unknown or unstable, and labels a qualified mean at least 0.75 reliable, at most 0.25 unsupported, and intermediate values unstable. Compilation uses these labels (Appendix A.2).

Qualification combines identity and evidence coverage with reliability. The generic minimum is two observations; the reference builder stops a capability once it is reliable or unsupported, or after three observations. Removal additionally requires scoped evidence and transformation checks at coverage 0.95; Figure 2 shows the corresponding two- and eighteen-success thresholds.

3.3 Residual Skill Compilation

After profiling, we compile the support supplied to the receiver. Given T, P_A, and $\mathcal{S}$, routing selects sources for unmet obligations, while global linking assigns shared planning, verification, stopping, and recovery to one coordinator and preserves local task procedures. The linked text is lowered into source-hashed units with control kinds, dependencies, critical terms, and protection flags. Relations encode coverage, refinement, conflict, duplication, dependency, or independence; coverage remains scoped, and protected content includes code, templates, paths, flags, placeholders, and numerical literals.

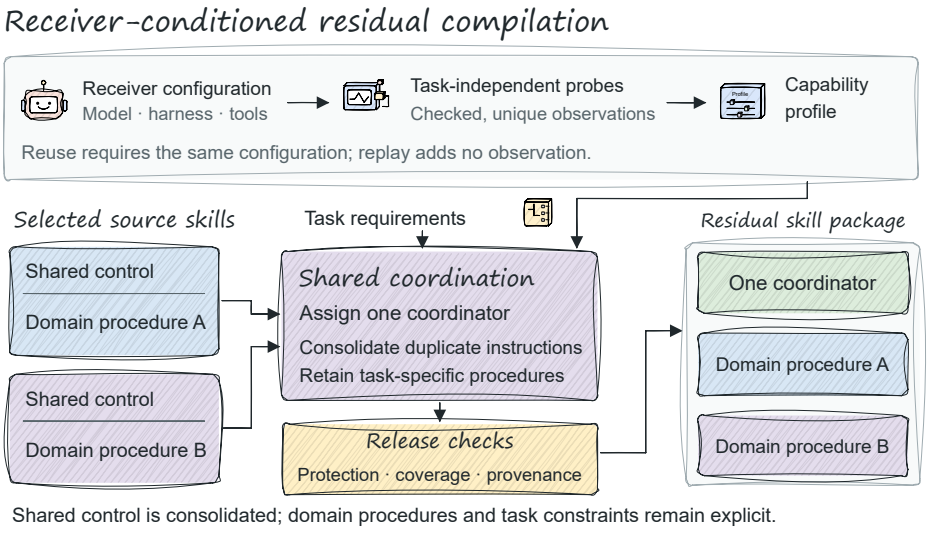


Figure 3: Task requirements and capability evidence guide shared coordination and duplicate consolidation while retaining required local procedures. Configuration-bound probes supply the profile; protection, coverage, and provenance checks precede release. The skill decomposition is schematic.

The actions are *Keep*, *Merge*, *Reinforce*, and *Remove*. Keep preserves an instruction; Merge records a canonical source and suppresses an exact duplicate; Reinforce retains scaffolding or adds interface-supported guidance; Remove applies only to an eligible non-domain unit with scoped coverage at least 0.95. Protection, conflict precedence, and dependency checks determine the release decision, with unresolved relations retained.

Source-linked decisions verify completeness, provenance, and eligibility before release and runtime expansion (Appendix G).

In Spot, global linking assigns `production-scheduling` as coordinator and `constrained-optimization` a local role. Active compilation merges two duplicate units, reducing linked input from 1,816 to 1,512 decoded characters while preserving constraints and domain procedures; source and output hashes bind the result to the delivered Skill (Appendices D.2 and G.1).

3.4 Evidence-Guided Execution

We then execute the residual package under evidence-guided control. Given the task contract and visible prefix h_t, the runtime screens revisions for public validation and retains an eligible deliverable during further search. A receipt binds quality, contract checks, and artifact identity; submission restores the retained artifact when needed (Figure 4).

The experimental runtime stops on request-value exhaustion, two consecutive genuine batches without global-best improvement, or a declared constraint boundary. A genuine batch needs a new iden-

tity, typed hypothesis, coverage, validation, and objective measurement; receipts record the path (Appendix A.4).

A standalone controller implements the ordered reference rules in Appendix A.4, distinguishing continuation, qualified expansion, submission, and stopping. Continue may perform local work or issue a model request, while both controllers consume public prefix evidence and record the retained artifact, validation state, and termination reason.

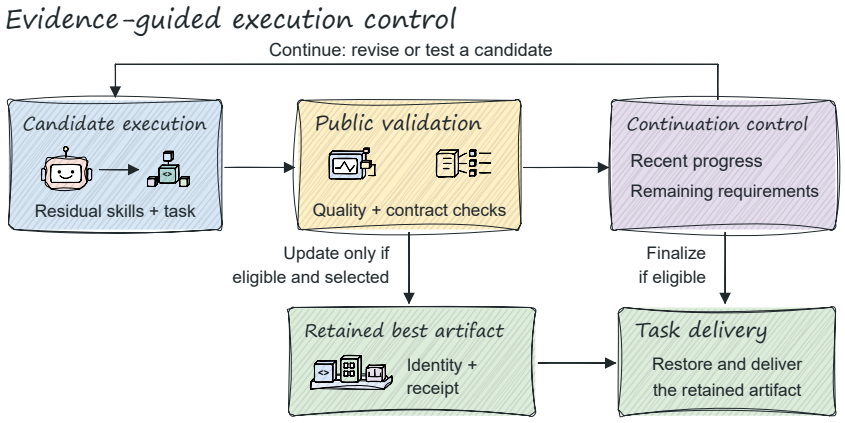


Figure 4: Public validation separates candidate exploration from artifact retention. Progress evidence and remaining requirements guide continuation; eligible retained output is restored for delivery. Appendix A.4 specifies the stopping conditions and the separate reference controller.

4 Experiments

We evaluate task outcomes (Section 4.2), examine profile matching and component contributions (Section 4.3), and analyze execution efficiency (Section 4.4).

4.1 Experimental Setup

Tasks and implementation. The tasks cover strategy-bot development (Generals), deadline- and cost-constrained scheduling (Spot), and query-optimizer development (DuckDB). Evaluation combines task reward with artifact-contract completion, with task-specific scales retained; Appendix B.1 gives the task contracts and harness.

The primary panel uses Terminus-2 2.0.0, `gpt-5.4-mini`, and medium reasoning. Conditions share CPU, memory, and total execution budgets; task-dependent command limits, DuckDB support, recovery paths, and revisions are listed in Appendices B.2 and B.

Baselines. Original and No Skill compare supplying the source skills with omitting them; Truncation and Random removal reduce content by deletion; LLM compression and LLMLingua-2 provide compression alternatives; Structured Local applies structured local transformations. Appendix B.2 defines their preparation and the project's Structured Local implementation.

Evaluation and metrics. We assess final reward and artifact-contract completion and measure elapsed time, requests, observed execution tokens, and summed API duration. Token totals include available input and output usage, including retries; preparation costs are reported separately where available (Appendix B.3).

4.2 Main Results

DeltaSkill improves Generals reward, attains near-best Spot quality with 60% fewer requests, and completes the DuckDB artifact contract while all seven baseline contracts remain incomplete (Table 1; Figures 5 and 6). The execution records connect these outcomes to two complementary

Method	Generals		Spot		DuckDB	
	Reward ↑	Wall (min) ↓	Reward ↑	Wall (min) ↓	Reward ↑	Wall (min) ↓
Original	0.0000	241.71	0.9446	180.72	N/A	360.70
No Skill	0.2800	244.99	0.9395	180.77	N/A	360.79
Truncation	0.0150	242.20	0.8929	180.80	N/A	360.78
Random removal	0.0650	242.09	0.8968	180.80	N/A	360.78
LLM compression	0.2700	241.36	0.8458	180.79	N/A	382.07
LLMLingua-2	0.1300	242.59	0.9561	180.80	N/A	360.79
Structured Local	0.0000	241.55	**0.9597**	180.80	N/A	360.79
DeltaSkill (ours)	**0.7700**	**71.90**	0.9585	**27.60**	**0.7924**	**210.57**

Table 1: Comparison with seven baselines on Generals, Spot, and DuckDB in final reward and wall time (minutes). Detailed measurements are provided in Appendix C.1.

behaviors: compilation consolidates repeated procedural guidance, while progress-based control governs further search around a retained artifact. In the primary Spot run, finalization restores that artifact after later candidates deteriorate, separating exploration from delivery. Section 4.4 examines instruction changes, input growth, and timing.

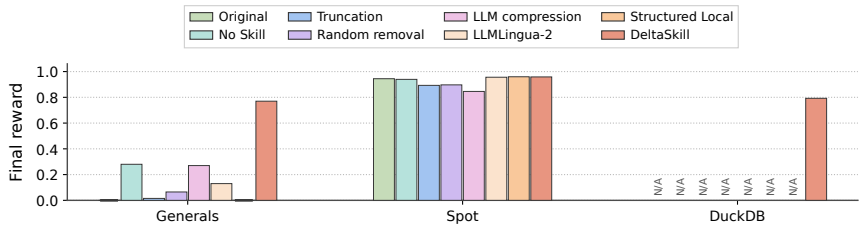


Figure 5: Final reward across three task panels: (a) Generals, (b) Spot, and (c) DuckDB. Colored legend entries identify the eight methods; N/A denotes an ineligible reward and zero denotes an observed zero.

DeltaSkill is the only method with a positive Generals reward and the only method with an eligible DuckDB artifact; on Spot, it remains close to the strongest endpoint. The aligned panels show the task-specific quality outcomes of the same compiled support and evidence-guided control.

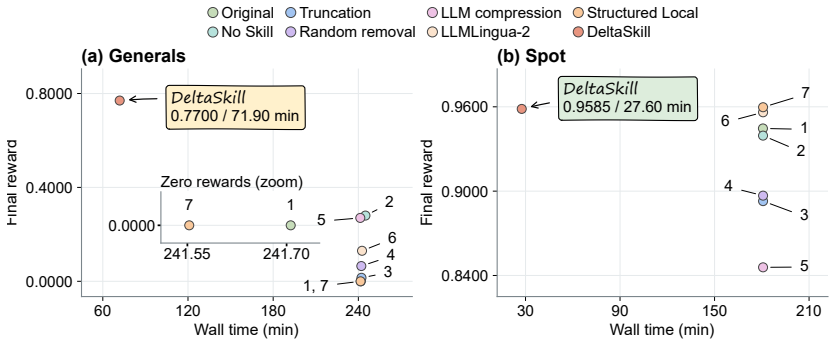


Figure 6: Quality–time comparison: (a) Generals and (b) Spot. Colored markers identify methods; the inset separates the true-zero Generals endpoints. DuckDB baselines remain N/A.

DeltaSkill moves the high-quality Spot endpoint toward a much shorter recorded trial and reaches the completed DuckDB contract under a heavier request workload. The panel-wise markers separate method identity from task scale, while retained-artifact control explains the resulting quality–time frontier.

4.3 Ablation Study

We examine receiver-profile matching and the interaction between compilation and stopping.

Profile-conditioned support. We compare the receiver's own profile (matched) with the other receiver's profile (mismatched) for Spot execution support. Both receivers use GPT-5.5 with medium reasoning and a shared R13 core. Within each receiver, the base Skill body and common tools are fixed; the intervention selects additional execution support.

Both profiles select batching and state reuse; planning and precedence are added for Terminus-2 and omitted for OpenCode. The resulting guidance differs by 514 UTF-8 bytes and is bound to the original first requests (Appendix H.1). Table 2 uses the same frozen evaluator.

Matching improves the recorded artifact reward by 0.0404 in OpenCode and 0.0104 in Terminus-2. The Terminus-2 matched artifact includes saved-progress continuation and separate verification after provider interruptions; Appendix H.1 specifies recovery conditions and the complete record, including no-profile outcomes.

Receiver	Matched	Mismatched
Terminus-2	**0.9278**	0.9174
OpenCode	**0.9124**	0.8720

Table 2: Profile-matching ablation on Spot with two receivers. Each cell reports one evaluated final-artifact reward. Protocol details are in Appendix H.1.

The process analysis connects supplied support to search and retention through 79 source-bound public-validation receipts and tool returns (Appendices J.2 and J).

Compilation and stopping. We cross compiled support versus routed raw Skills with adaptive stopping versus a fixed three-hour budget on Spot, using Terminus-2 and GPT-5.5 (medium). Each condition runs once with shared task, resources, common helper, evaluator, and best-artifact finalization. The input factor includes profile-conditioned support; Appendix K specifies both interventions.

Input	Stopping	Reward ↑	Time (min) ↓	Req. ↓	Tokens (M) ↓
Compiled + support	Adaptive	0.9457	16.92	**10**	**0.338**
Compiled + support	Fixed	**0.9538**	180.18	132	4.155
Raw Skills	Adaptive	0.7979	**13.61**	13	0.469
Raw Skills	Fixed	0.9516	180.22	153	6.047

Table 3: Compilation and stopping ablation on Spot: final reward, Agent time, requests, and execution tokens. Usage is complete; fixed-budget conditions retain their best eligible artifact at the deadline.

With adaptive stopping, compiled support raises reward from 0.7979 to 0.9457 while reducing tokens by 27.9%, at a longer Agent time (16.92 versus 13.61 minutes). Under the fixed budget, rewards are 0.9538 and 0.9516, with 31.3% fewer tokens for compiled support.

With compiled support, adaptive stopping reduces Agent time by 90.6% and tokens by 91.9% relative to fixed continuation, at a reward difference of -0.0081. With raw Skills, the reward difference is -0.1537. Adapted support thus accompanies a smaller early-stopping quality gap in these four executions (Appendix K.2).

Figure 7 traces the primary Spot run, separate from the component experiment: candidate scores fluctuate after the maximum while a retained artifact remains available for delivery.

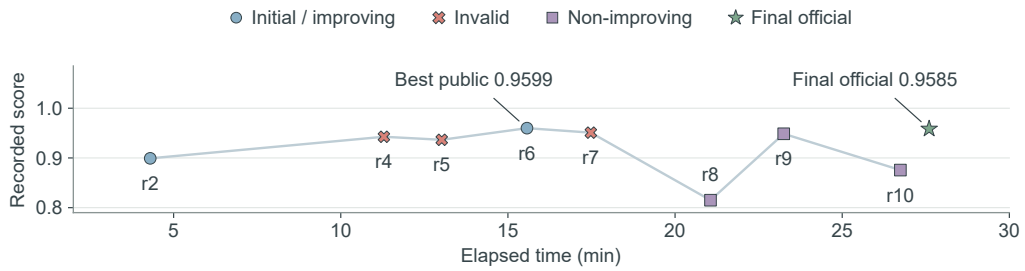


Figure 7: Primary Spot execution: public candidate scores and the separately evaluated final artifact. The line includes invalid and non-improving candidates; request labels identify observations.

Four requests follow the public maximum of 0.9599. Finalization restores the retained artifact (official score 0.9585), rather than the last candidate. The receipt records score-stagnation stopping without denied requests (Appendix E.1).

4.4 Efficiency Analysis

Execution time and request use. On Spot, DeltaSkill uses 10 requests and 379,795 observed tokens versus Structured Local's 25 and 888,029: reductions of 60% and 57.2%. Wall time falls from 180.80 to 27.60 minutes. Summed API duration and peak input instead increase (Figure 8); Appendix C.3 gives all comparators, including No Skill's lower token count.

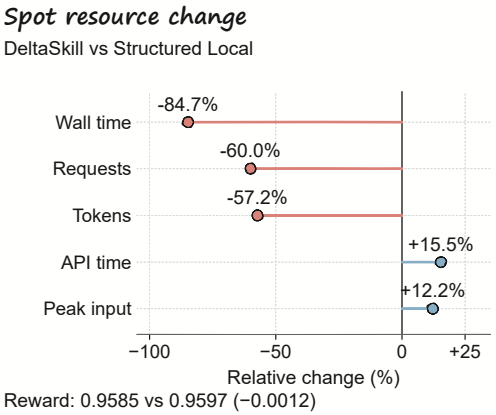


Figure 8: Spot changes relative to Structured Local. Appendix C.4 gives original values and definitions.

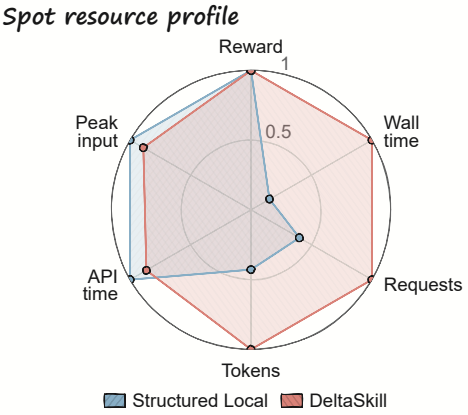


Figure 9: Pair-normalized view of the same endpoints. Larger radii indicate preferable values.

Complete-usage Spot traces; preparation excluded. Wall and summed API time have distinct scopes. Original values and normalization are in Appendix C.4.

Realized instruction changes. Of 39 decisions, 21 keeps and 14 reinforcements preserve source text; four merge exact duplicates and none remove (Appendix G). Active compilation of the six-unit Spot example suppresses 304/1,816 decoded characters (16.7%) while preserving canonical copies and task constraints. Controller contracts are tested in Appendix H.3.

Spot input use falls by 65.6% and output use rises by 25.5% (Figure 12), so reduced input exposure outweighs extra output (Appendix C.5).

Request count and input growth. With complete usage, Spot records 10 versus 25 requests and 37,980 versus 35,521 mean input-plus-output tokens per request (DeltaSkill versus Structured Local). Symmetric accounting assigns $-551{,}255$ tokens to count and $+43{,}021$ to size: fewer requests outweigh larger ones (Appendix I.1). Figure 10 shows input trajectories; Appendix F.2 covers all 1,247 requests, keeping missing usage unknown.

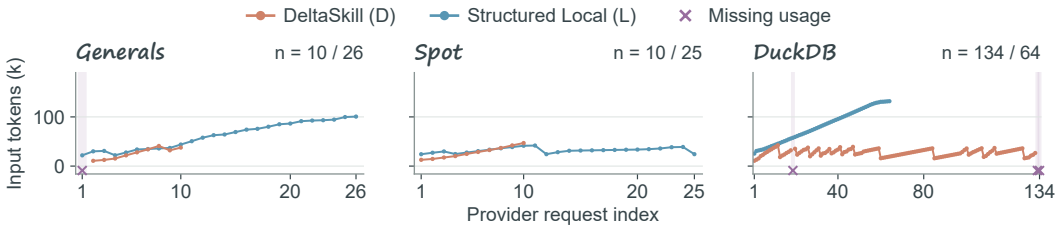


Figure 10: Input tokens by provider request, including retries, for three task pairs on a shared vertical scale. n gives DeltaSkill / Structured Local counts; gaps and crosses below zero mark unknown usage.

In Generals, the comparator accumulates larger inputs over more requests. DuckDB's completed run has repeated input reductions alongside recorded compactions, a smaller observed context footprint, but more requests and summed API time (Appendices C.3 and F.2).

Provider time and intervening gaps. On Spot, the request window shrinks from 175.97 to 26.56 minutes despite higher summed API time: shorter inter-request gaps, rather than uniformly faster calls, accompany this reduction (Figure 11).

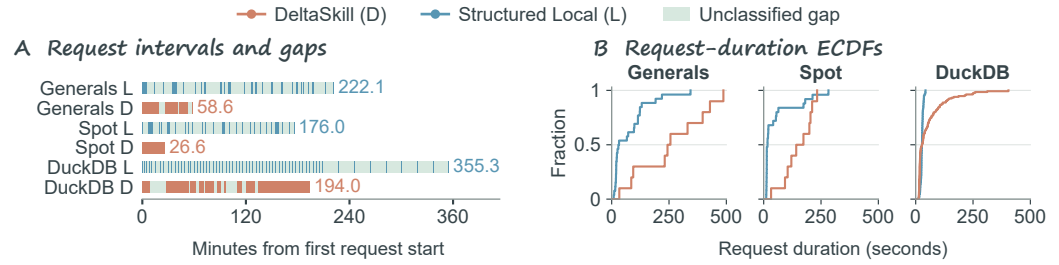


Figure 11: Request intervals (A) and duration ECDFs (B), including retries and failures. Each curve represents one execution. Mint gaps may include tool use, idle time, or other overhead; end labels give minutes. Windows exclude preparation and later verification and differ from full-run wall time (Appendix I.2).

5 Conclusion

In this paper, we introduced and evaluated DeltaSkill, a framework for adaptive skill use in long-horizon agents. We show that capability-aware residual skill compilation and evidence-guided execution control can support high-quality solutions with lower overhead. Our findings reveal that appropriate skill use can yield an efficiency gain alongside improved or competitive solution quality. It manifests in various forms, such as reduced token consumption, fewer requests, and shorter execution. We also explored profile matching and the interaction between compilation and stopping. While stopping outcomes depend on supplied support, these analyses highlight the value of jointly designing compilation and control. Finally, our findings highlight continual capability profiling and cross-task skill adaptation as directions toward more efficient and generalizable agents.

AI USE STATEMENT

Generative AI assisted with research framing, hypothesis refinement, experimental design, method implementation, record processing and analysis, result interpretation, translation, and manuscript preparation. The authors reviewed the AI-assisted work and take responsibility for the paper.

REPRODUCIBILITY STATEMENT

The anonymous supplement supplies read-only audit scripts, derived records, generated tables, figure code, bibliography provenance, and checksum reports. Appendices A–C specify the reference implementation, task protocols, measurement rules, and detailed results; Appendices D and E provide prompt excerpts, compilation examples, and execution cases. Appendices F–H connect request records and compiler decisions to archive checksums and offline fixtures. Appendix I adds resource decompositions, interval accounting, profile-to-instruction interventions, fixed-evidence perturbations, and missing-usage sensitivity.

The Spot profile comparison has a separate configuration identity and a complete six-condition evidence record, including recovery provenance (Appendix H.1); it is not pooled with the primary panel. Appendix J adds source-bound candidate trajectories and receiver-specific process records. The component experiment separately crosses compilation and stopping under a common Spot configuration (Appendix K); its execution usage is complete. The supplement supports verification of derived records and integrity checks. Full audit reproduction requires the research archive described in Appendix B; rerunning the interactive tasks additionally requires access to the external model service.

ETHICS STATEMENT

This study concerns software and simulated task environments. Release preparation must retain benchmark licenses and provenance while excluding provider credentials, private endpoint configuration, and unrelated user data. Reporting distinguishes contract-complete outcomes from incomplete artifacts and invalid candidates.

REFERENCES

Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. SkVM: Revisiting Language VM for Skills across Heterogenous LLMs and Harnesses. *arXiv preprint arXiv:2604.03088*, 2026. URL https://arxiv.org/abs/2604.03088v3.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do Agent Skills Actually Help in Real-World Software Engineering? *arXiv preprint arXiv:2603.15401*, 2026. URL https://arxiv.org/abs/2603.15401.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. *arXiv preprint arXiv:2310.05736*, 2023a. URL https://arxiv.org/abs/2310.05736.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. *arXiv preprint arXiv:2310.06839*, 2023b. URL https://arxiv.org/abs/2310.06839.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A Benchmark for Evaluating Self-Evolving Agents Beyond Episodic Assessment. *arXiv preprint arXiv:2604.08988*, 2026. URL https://arxiv.org/abs/2604.08988.

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? *arXiv preprint arXiv:2310.06770*, 2023. URL https://arxiv.org/abs/2310.06770.

Ryangkyung Kang, Hongcheol Cho, and Youngeun Kim. SkillRet: A Large-Scale Benchmark for Skill Retrieval in LLM Agents. *arXiv preprint arXiv:2605.05726*, 2026. URL https://arxiv.org/abs/2605.05726.

Hao Li, Chunjiang Mu, Jianhao Chen, Siyue Ren, Zhiyao Cui, Yiqun Zhang, Lei Bai, and Shuyue Hu. Organizing, Orchestrating, and Benchmarking Agent Skills at Ecosystem Scale. *arXiv preprint arXiv:2603.02176*, 2026a. URL https://arxiv.org/abs/2603.02176.

Qinfeng Li, Dalin He, Yuntai Bao, Ying Yang, Ruoxi Chen, Xinyan Yu, Lizhou Liang, Ge Su, Wenqi Zhang, and Xuhong Zhang. SkillAligner: Treating Retrieved Skills as Adaptable Drafts at Execution Time. *arXiv preprint arXiv:2608.06880*, 2026b. URL https://arxiv.org/abs/2608.06880v1.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. *arXiv preprint arXiv:2403.12968*, 2024. URL https://arxiv.org/abs/2403.12968.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. *arXiv preprint arXiv:2303.11366*, 2023. URL https://arxiv.org/abs/2303.11366.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An Open-Ended Embodied Agent with Large Language Models. *arXiv preprint arXiv:2305.16291*, 2023. URL https://arxiv.org/abs/2305.16291.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. *arXiv preprint arXiv:2210.03629*, 2022. URL https://arxiv.org/abs/2210.03629.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

A Implementation Details

This appendix connects capability evidence, residual compilation, and execution through the archived reference interfaces. Frozen runtime bindings and receipts identify the components and policies used in each reported execution.

A.1 Overall Algorithm

The compiler takes configuration A, visible task contract T, source skills $\mathcal{S}$, and optional capability profile P_A. Configuration binds the model, agent and harness versions, tool interface, and visible execution contract; T specifies deliverables, public checks, and any visible objective. Compilation returns residual skills $\mathcal{S}^{\Delta}_{A,T}$, source-linked decisions, and qualified expansion stages where available. Execution returns a retained artifact, validation state, and termination reason; successful completion requires satisfaction of the artifact contract.

Table 4 shows how configuration-specific calibration supplies evidence to task-specific compilation and execution. Profile updates and guarded linking are deterministic on supplied evidence; semantic routing or linking may call a model, and probes use the configured agent adapter. Model-call counts depend on the configured compilation and probe paths. Source skills remain available to compilation, whereas runtime expansion exposes only qualified stages.

Step	Procedure
1	Validate the identity of P_A against A; obtain missing calibration evidence through the configured probe path.
2	Extract visible task obligations from T; route relevant skills and link their dependencies.
3	Lower selected skills into control units; construct relations using source content, profile evidence, and supported candidate relations.
4	Apply ordered linking rules; guard each transformation and check complete decision coverage and provenance.
5	If release checks fail, withhold that compiled artifact; otherwise emit residual skills and separately qualified expansion stages.
6	Execute with the residual skills. Screen candidates, validate survivors, and retain the best eligible artifact.
7	Refresh visible state; choose local work, another model request, qualified expansion, submission, or branch termination.
8	Repeat execution while continuation is admitted; finalize the retained artifact and record how execution ended.

Table 4: Overall workflow of DeltaSkill. The runtime branch uses visible evidence and the controller bound to the execution revision.

A.2 Capability Profiling

Probe inputs and observations. One observation aggregates all probe variants for one seed under a fixed configuration. The specification supplies target capabilities, fixture, variant names, verifier identity, and selection costs. The runner executes through the configured adapter, checks its identity, and applies a registered deterministic verifier. By default it calls that verifier twice on identical executions and requires identical outputs. Each observation records the seed, outcome, behavior features, requests, duration, tokens, and available trace references. Both verifier checks belong to the same agent observation.

Nine control probes cover eleven capabilities (Table 5); waiting/polling and repair/fallback share probes. A separate domain-constraint probe is available but excluded from standard completeness. The listed variants encode contrasting fixtures, such as independent/dependent actions and repairable/persistent failures. They become observations only through a configured fixture materializer, agent adapter, and verifier registry.

State updates and reuse. Each capability begins with $\alpha = \beta = 1$. Success increments α and failure increments β for every probe target. Updates check configuration and probe identities. Reap-

Probe identifier	Target capability	Variant identifiers
`plan_once`	Planning	`linear`; `branching`
`batch_independent_actions`	Batching	`independent`; `dependent`
`bounded_wait`	Waiting; polling	`delayed_output`; `silent_then_success`
`verify_once`	Verification	`already_valid`; `single_failure`
`bounded_repair`	Repair; fallback	`repairable_once`; `persistent_failure`
`stop_on_success`	Termination	`success_marker`; `distractor_after_success`
`route_relevant_skill`	Routing	`one_relevant`; `two_complementary`
`reuse_observed_state`	State reuse	`stable_state`; `changed_state`
`respect_precedence`	Precedence	`compatible`; `conflicting`

Table 5: Reference control-probe catalog. Nine specifications cover eleven control capabilities. The table lists probe and variant fields in the reference specification.

plying an observation identifier has no effect; duplicate identifiers in an initial list are rejected before replay. Unique observations are the unit of evidence counted in these updates.

Capability state guides compilation; profile completeness tracks coverage of the required capabilities. With no evidence, the generic updater labels a capability unknown; below its minimum evidence count, it labels the capability unstable. Once that minimum is met, the state is reliable when $\alpha/(\alpha+\beta) \geq 0.75$, unsupported when that value is at most 0.25, and unstable otherwise. The default minimum is two observations. The reference complete-profile builder instead sets the minimum to one and stops seeking evidence for a capability when it is reliable, unsupported, or has at least three observations. A complete profile requires every designated capability to be present and non-unknown, so it may include unstable capabilities.

Probe selection ranks unresolved decisions by summed uncertainty times latency weight, divided by expected seconds plus request-equivalent cost. This cost-weighted heuristic prioritizes unresolved decisions. Profile reuse separately requires all configuration fields to match; new runtime observations do not authorize transfer to another configuration.

Two successes give $\text{Beta}(3,1)$ and mean 0.75: reliable under the default updater, but below the 0.95 removal threshold. A subsequent failure gives $\text{Beta}(3,2)$ and mean 0.60, hence unstable. For comparison, one unique pass and one unique failure give $\text{Beta}(2,2)$; replaying the pass adds nothing. These examples illustrate posterior-mean updates. The means serve as uncalibrated belief scores, not estimates or lower confidence bounds of downstream task success.

Figure 2 extends the all-success calculation: eighteen successes from $\text{Beta}(1,1)$ reach mean 0.95.

An offline replay of the frozen acquisition logic checks the distinction against the built-in catalog. From an empty prior, each required capability belongs to one probe; a shared probe updates both of its targets. Enumerating the three-observation success/failure prefixes shows that the stop-on-resolved policy uses at most three observations per capability and reaches a maximum terminal mean of 0.75. It therefore cannot itself reach 0.95 on this acquisition path. Separately, the four-cell Spot comparison permits no profile-driven removal under its source-scope policy. Its measured intervention is consequently support selection, as verified in Appendix H.1.

A.3 Residual Skill Compilation

Representation and relation construction. Each control unit links its text and control kind to an identifier, owning skill, source span and hash, dependencies, critical terms, and protection flag. Lowering parses Markdown while distinguishing generic control from domain procedures. Fenced code and templates are protected; critical terms include inline code, paths, flags, placeholders, and numerical literals. Optional triggers, preconditions, postconditions, and task-relevance fields need not be fully inferred from every instruction.

Relations express coverage, refinement, conflict, duplication, dependency, or independence. The deterministic builder adds dependencies, detects cross-skill duplicates after case and whitespace normalization, and links non-domain control units to matching reliable beliefs. Profile-derived coverage retains the belief mean and evidence identifiers; supplied candidate relations must pass endpoint and evidence checks. The ordered linking rules determine the transformation for each candidate edge.

Ordered compilation rules. The conservative linker applies Table 6 in order, using the strongest relevant edge. Protection precedes conflicts, duplication, and coverage; weak or unresolved relations retain text. Precedence marks a scoped conflict candidate without rewriting its text. Merge records a canonical source and suppresses the duplicate. Reinforcement belongs to the action vocabulary and requires guard evidence, but this linker does not generate reinforcement text.

Condition	Decision
Protected unit	Keep the original unit text.
Conflict edge	Mark precedence and retain text if score ≥ 0.95 with evidence; otherwise uncertain-keep.
Duplicate edge	Merge into its canonical source if score ≥ 0.95 with evidence; otherwise uncertain-keep.
Coverage edge	Remove only a non-domain unit with score ≥ 0.95 and evidence; otherwise keep.
No applicable edge	Keep; a refinement or dependency alone does not authorize deletion.

Table 6: Ordered unit decisions in the conservative relation linker. Thresholds are heuristic implementation settings.

Release checks. Release requires exactly one decision per unit and provenance connecting known sources, registered evidence, and output text. Insufficient confidence, transformation of protected content, or omission of critical terms triggers unit-level retention. Missing or duplicate decisions, missing required evidence, or inconsistent provenance instead withhold the artifact. The archived gate accepts counterexamples involving externally supplied KEEP mutations, duplicate cycles, and required-dependency removal (Appendix G). The ordinary linker preserves protected source text, and Appendix G.3 separately tests a repair of the three gate defects. These checks evaluate structural consistency; task-artifact validation separately evaluates the execution outcome.

A.4 Execution and Stopping

Candidate state. Retention binds the selected candidate to its objective and artifact identity so finalization can recover it after further exploration. Screening prioritizes candidates for validation, which establishes artifact eligibility. Visible state tracks artifact completeness, public checks, local actions, unresolved work, new information, actionable failures, stalled branches, and qualified expansion. Finalization must satisfy the task contract.

Continuation priority. For visible interaction, the standalone reference controller first permits deterministic local work, then admits submission when artifacts are complete and public validation passes. An incomplete artifact contract with unresolved work requires another model request, or expansion when the branch is stalled and a qualified stage is available. A remaining stalled branch expands if possible and otherwise terminates. Only after these earlier rules does new, actionable failure evidence justify another request; at that point, preserving task success can override a non-positive heuristic net value. Otherwise, the controller terminates the branch. A positive gain estimate alone is not a request-admission rule, and branch termination is not equivalent to declaring successful completion.

Runtime binding and expansion. The full pipeline binds a success-first runtime that is distinct from the standalone controller above. It retains the validated best candidate and defines three terminal conditions: request-value exhaustion, two consecutive genuine batches without a noise-adjusted global-best improvement, or a declared constraint boundary. A genuine batch requires a new candidate identity, a typed hypothesis, coverage information, public validation, and an objective measurement; invalid batches do not contribute to score stagnation.

Expansion exposes an additional compiled stage only after qualification, keeping the raw original skill tree outside this runtime fallback. Continuation decisions use public prefix evidence, while official evaluation supplies a separate outcome measurement. The standalone state schema includes a terminal hidden-verification result; the benchmark runtime marks hidden evaluation as unavailable to the controller. A timeout or transport interruption leaves valid controller termination unestablished.

Figure 4 summarizes candidate validation, artifact retention, and evidence-guided continuation: evidence may justify repair without making a candidate eligible to replace the incumbent.

B Experimental Details

The main panel reports 24 archived task–method executions drawn from system development, with final artifacts and incurred resources. Table 8 identifies the contributing attempts. Archive consistency checks reconcile records, provider traces, and official receipts across all 24 traces, 78 original frozen checksums, and 109 inventory entries.

B.1 Tasks and Evaluation

Task contracts. The archived *Long-Horizon Terminal-Bench* harness is separate from the official Terminal-Bench leaderboard. Generals evaluates a strategy bot by win rate against opponent types. Spot evaluates scheduling under deadline and cost constraints; policies may use the observed trace prefix but not future availability. DuckDB requires optimizer code changes, a patch, benchmark results, and a journal. Contract completion requires the numerical evaluation and all required deliverables.

Public screening guides candidate search; final reward requires official artifact evaluation and satisfaction of task and reporting criteria. Rewards are reported separately for each task. Frozen metadata records hidden evaluation as unavailable to the controller.

Final-artifact evaluation. All selected Generals and Spot endpoints satisfy their reporting contracts. The Generals reporting status was corrected after final-artifact inspection confirmed successful runner completion, all required artifacts, official reward 0.7700, and matching candidate and retained-best identities. The evidence package includes the original records and the correction. Artifact counts report contract completion and additional bookkeeping files; official reward measures solution quality.

Reward eligibility and recovery are handled at the artifact level. Seven DuckDB baseline endpoints have incomplete required contracts and therefore N/A reward, while their measured execution resources remain included. For the DeltaSkill DuckDB endpoint, official evaluation resumed on the existing code artifact without additional model generation, and all incurred evaluation time is included in wall time. This recovery supplies final-artifact evidence; the endpoint's adaptive-stop status remains unestablished. Spot has a different measurement distinction: its public search maximum is 0.9599, whereas the final official reward is 0.9585.

B.2 Baselines and Configurations

Skill conditions. Original supplies the source skill collection; No Skill omits it. Truncation deletes text, Random removal deletes sentences, and LLM compression rewrites skill bodies. The archived LLMLingua condition uses the multilingual MeetingBank compressor and is displayed as LLMLingua-2. Structured Local is our structured local-transformation baseline, described as SkillSmith-style. Preparation handles front matter and resources as well as bodies. We use the archived outputs, with no matching procedure for lengths or preparation costs.

Shared execution settings. All selected runs record Terminus-2 version 2.0.0, `gpt-5.4-mini`, and medium reasoning effort. Within each task, all conditions receive the same CPU allocation, memory allocation, and total agent-execution limit, summarized in Table 7. The native baseline protocol continues until its execution limit. The DeltaSkill runtime uses evidence-guided continuation

within the same task limit. A time limit constrains agent execution rather than the complete trial, so recorded wall time can exceed it when setup or final evaluation is included.

Task	CPUs	Memory (MB)	Agent limit (h)	Command limit (s)	
				Baselines	DeltaSkill
Generals	4	4,096	4	900	1,800
Spot	2	4,096	3	900	300
DuckDB	4	8,192	6	900	3,600

Table 7: Recorded execution settings. The agent limit, CPU allocation, and memory allocation are shared across conditions within each task. Terminal-command limits differ between the native baselines and DeltaSkill.

Frozen revisions and task adaptations. The selected DeltaSkill endpoints use task-adapted revisions: R13 for Spot, R13b for Generals, and R13c for DuckDB. Their recorded common components are the success-first controller policy and binding, candidate-portfolio protocol, and 64,000-token local compaction limit. Terminal-command limits vary with the workloads, as shown in Table 7; DuckDB also uses R38 accelerated-search support and an artifact-recovery path. The comparison evaluates complete systems, including their task-specific execution support and controller policies.

Source provenance differs across the selected runs. The Generals and DuckDB analysis packages contain identical compiler files but task-specific runtime differences. The Spot package was reconstructed from the first post-run snapshot and retained edit history because a complete launch-time snapshot is unavailable. Independent replay checks the reconstruction; its fidelity to the executed code depends on the completeness of that history. The reconstructed package differs from the Generals analysis package only in later provider-authentication support. The runs share model settings and controller bindings, with the runtime differences documented above. Source manifests and reconstruction details are included in the evidence package.

B.3 Metrics and Resource Accounting

Measurement boundaries. Final reward measures an eligible delivered artifact; completion records contract satisfaction. Wall time covers the whole recorded trial through final evaluation. Agent duration and summed API duration have narrower scopes. Provider requests and API duration include recorded retries. Separating idle time, queueing, and local computation requires additional instrumentation.

Observed token usage. Observed token usage is the sum of input and output usage from every trace entry with a usage payload, including retries. Input usage includes cached tokens, and output usage includes reasoning tokens. To align token accounting with request counts and API duration, which include retries, we add known retry usage omitted from the original frozen token aggregate. The additions are 16,936 tokens for Generals DeltaSkill, 40,371 for DuckDB DeltaSkill, and 121,007 for DuckDB Random removal, yielding totals of 390,855, 4,025,190, and 9,462,377, respectively. The original aggregates remain available alongside the reconstructed fields.

Twelve requests across the 24 traces lack usage; request identifiers are unique within each trace. Missing usage remains unknown, making affected observed totals lower bounds; billed token totals and dollar costs remain unmeasured. Task-run tables also exclude incompletely measured profile acquisition, routing, linking, compilation, and baseline preparation; Appendix C.3 reports the available partial estimate. A complete cost account would distinguish cold-profile construction from reuse and include setup, failed attempts, fallback, retries, execution, and verification.

C Additional Results

C.1 Detailed Results

Tables 8 and 9 describe the same 24 selected endpoints under Appendix B's measurement rules. Rankings and arithmetic use unrounded records. Display precision is four decimals for rewards and public scores, two for times, and three for tokens in millions; counts and limits are integers. Rounding uses nearest values, with exact ties away from zero.

Method	Attempt	Reward ↑	Complete	Wall (min) ↓	API (min) ↓
(a) Generals					
Original	5	0.0000	Yes	241.71	36.52
No Skill	4	0.2800	Yes	244.99	39.17
Truncation	4	0.0150	Yes	242.20	**23.52**
Random removal	4	0.0650	Yes	242.09	31.32
LLM compression	4	0.2700	Yes	241.36	31.54
LLMLingua-2	5	0.1300	Yes	242.59	51.20
Structured Local	4	0.0000	Yes	241.55	32.91
DeltaSkill (ours)	1	**0.7700**	Yes	**71.90**	43.11
(b) Spot					
Original	6	0.9446	Yes	180.72	26.27
No Skill	4	0.9395	Yes	180.77	**21.36**
Truncation	4	0.8929	Yes	180.80	26.58
Random removal	4	0.8968	Yes	180.80	22.16
LLM compression	4	0.8458	Yes	180.79	27.33
LLMLingua-2	4	0.9561	Yes	180.80	23.96
Structured Local	4	**0.9597**	Yes	180.80	21.84
DeltaSkill (ours)	1	0.9585	Yes	**27.60**	25.21
(c) DuckDB					
Original	5	N/A	No	360.70	41.05
No Skill	4	N/A	No	360.79	43.50
Truncation	4	N/A	No	360.78	50.39
Random removal	4	N/A	No	360.78	52.57
LLM compression	4	N/A	No	382.07	45.97
LLMLingua-2	4	N/A	No	360.79	39.71
Structured Local	4	N/A	No	360.79	**29.20**
DeltaSkill (ours)	1	**0.7924**	Yes	**210.57**	124.01

Table 8: Detailed end-to-end results by task. Complete indicates delivery of all required artifacts; API is summed provider time, including retries. **Bold** marks the best recorded value per task and metric (↑: higher; ↓: lower).

Generals. The selected DeltaSkill Generals execution combines higher reward with less wall time, fewer requests, and fewer observed tokens than No Skill, the highest-reward baseline. Rewards are 0.7700 and 0.2800, respectively, a difference of 0.4900 reward units. DeltaSkill records 70.65% less wall time, 10 rather than 26 provider requests, and 390,855 rather than 475,914 observed tokens. API duration instead increases from 39.17 to 43.11 minutes. All Generals endpoints satisfy the completion contract; the reported differences concern reward and resource use.

Spot. The Spot comparison pairs near-best reward with less wall time, fewer requests, and fewer observed tokens. Structured Local has the highest final reward, 0.9597, while DeltaSkill reaches 0.9585, a difference of 0.0012. Relative to Structured Local, DeltaSkill records less wall time, from 180.80 to 27.60 minutes (84.74%), fewer requests, from 25 to 10, and fewer observed tokens, from 888,029 to 379,795. API duration increases from 21.84 to 25.21 minutes. No Skill uses still fewer recorded tokens (336,513, a lower bound with one missing usage payload) but has reward 0.9395. These observations compare final endpoints; comparable intermediate baseline evaluations are unavailable.

DuckDB. DeltaSkill completes the required DuckDB artifact contract with reward 0.7924. Among the seven selected baselines, LLM compression supplies two of its three required artifacts, and the other six records supply none of their three required artifacts. These counts cover final required deliverables. Baseline rewards are consequently ineligible and reported as N/A. Across the selected full-system endpoints, including task-specific execution support, one DeltaSkill execution completes the contract and seven baseline executions remain incomplete.

C.2 Scope of Component Evidence

Endpoint comparisons describe quality and resource use; the component experiment in Table 3 directly crosses input support and stopping, while mechanism checks examine instruction transformations and runtime decisions. The separate Spot profile comparison in Table 2 supplies one development outcome per condition, with receiver configurations, support mappings, and recovery conditions specified in Appendix H.1.

The appendices provide trace accounting (Appendix F), source-to-output transformations and finite guard checks (Appendix G), and controller-priority checks (Appendix H). The later profile panel varies the support mapping supplied to the receiver; the utility-dev1 revision instead selected the same arm within each receiver and is retained as development history. The reported outcomes include incomplete cells and transport failures. The controller cases document admission and termination decisions, with post-maximum continuation described in Appendix E.

C.3 Efficiency and Overhead

	Generals		Spot		DuckDB	
Method	**Req. ↓**	**Tokens (M) ↓**	**Req. ↓**	**Tokens (M) ↓**	**Req. ↓**	**Tokens (M) ↓**
Original	27	1.037	23	1.037	113	11.964
No Skill	26	0.476	26	**0.337**	107	7.012
Truncation	24	0.954	23	0.914	110	11.284
Random removal	28	0.728	25	0.732	109	9.462
LLM compression	27	1.828	25	0.914	107	9.299
LLMLingua-2	46	2.113	25	0.861	107	9.482
Structured Local	26	1.722	25	0.888	**64**	5.339
DeltaSkill (ours)	**10**	**0.391**	**10**	0.380	134	**4.025**

Table 9: Execution resources for the same runs as Table 1. Req.: provider requests; M: million observed input-plus-output tokens, including known retries. **Bold** marks the lowest recorded usage per column. Measurement boundaries follow Appendix B.3.

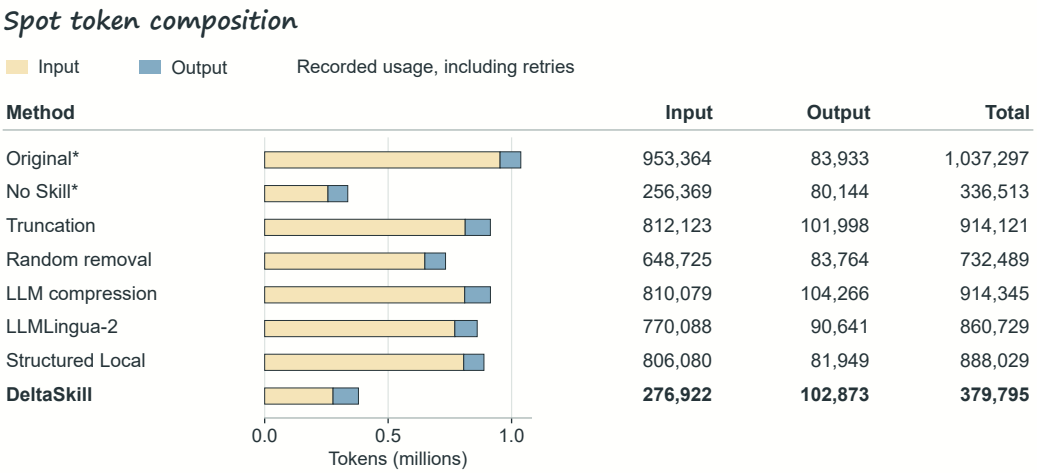
Context and execution resources. DuckDB combines a smaller observed context footprint with more requests and API time. The DeltaSkill endpoint uses 134 requests against a baseline range of 64–113 and 124.01 API minutes against 29.20–52.57. Its observed token total is 4.025 million, below the baseline range of 5.339–11.964 million, and its maximum per-request input is 42,377 tokens against 123,656–181,677. The endpoint records 13 local compactions, while the baselines record none. The completed execution combines this bounded context footprint with the task-specific support described in Appendix B.2.

Preparation overhead. The partial Generals preparation estimate covers one routing and one linking model request outside the formal trial, totaling 23,825 tokens and 81.34 model seconds, plus 5.48 seconds of recall work. Summing those durations sequentially gives 86.82 seconds; combining this estimate with the selected trial yields approximately 73.35 minutes and 12 model requests. With known runtime retry usage included, the partial combined total is 414,680 observed tokens. This manifest-based estimate applies to the recorded Generals endpoint. It excludes cold-profile acquisition, remaining compiler overhead, missing usage, and comparable baseline-preparation costs.

End-to-end measurement and complete cost. DuckDB's 210.57-minute wall time includes final evaluation. Cost accounting follows the measurement scope in Appendix B.3.

C.4 Normalization of the Spot Radar Plot

Recorded values	Reward	Wall (min)	Requests	Tokens (M)	API (min)	Peak input
Structured Local	0.9597	180.80	25	0.888	21.84	41,777
DeltaSkill	0.9585	27.60	10	0.380	25.21	46,885



Method	Input	Output	Total
Original*	953,364	83,933	1,037,297
No Skill*	256,369	80,144	336,513
Truncation	812,123	101,998	914,121
Random removal	648,725	83,764	732,489
LLM compression	810,079	104,266	914,345
LLMLingua-2	770,088	90,641	860,729
Structured Local	806,080	81,949	888,029
DeltaSkill	**276,922**	**102,873**	**379,795**

* One missing usage payload each; these token sums are observed lower bounds.

Figure 12: Observed input and output tokens for the eight selected Spot executions, including recorded retries. Bars share a scale in millions of tokens; adjacent columns give exact token counts. DeltaSkill records 276,922 input and 102,873 output tokens. Asterisks mark Original and No Skill, each with one missing usage payload, so their component and total counts are lower bounds. Each row is one selected execution under the observed-usage accounting in Appendix B.3.

Both views use the same complete-usage Spot traces, excluding preparation. Signed changes use an absolute reward difference; elapsed and summed API time have distinct scopes. Radar axes favor higher reward or lower resource use. Peak input is in tokens. Appendix C.4 specifies normalization.

The Spot radar compares the same contract-complete endpoints as Table 1. Structured Local has usage for all 25 requests and DeltaSkill for all 10. Preparation costs are excluded; normalization uses unrounded records.

Reward is divided by the larger of the two rewards. For each resource metric, the smaller observed value is divided by the method's value. This maps every axis to at most one and makes larger radii preferable. In the order reward, wall, requests, tokens, API duration, and peak input, the normalized values are $(1.0000, 0.1526, 0.4000, 0.4277, 1.0000, 1.0000)$ for Structured Local and $(0.9987, 1.0000, 1.0000, 1.0000, 0.8661, 0.8911)$ for DeltaSkill. These pair-specific ratios summarize reward and resource measurements. Polygon area varies with the compared methods and axis order; comparisons use the individual axes.

C.5 Input and Output Token Composition

Figure 12 separates observed input and output counts for all eight Spot endpoints. Relative to Structured Local, DeltaSkill uses 65.6% fewer input tokens and 25.5% more output tokens. Exact counts accompany a common-scale stacked bar for each selected execution; missing usage for Original and No Skill remains explicit.

C.6 Quality and Elapsed Time

Figure 6 pairs final Generals and Spot rewards with recorded wall time. The plot compares final endpoints; intermediate baseline evaluations are unavailable. DuckDB endpoints with incomplete contracts have N/A rewards.

D Prompts and Skill Examples

D.1 Prompt Templates

The reference prompts select procedural support for missing task capabilities and coordinate control across skills. The following excerpts preserve the reference template instructions and identify

their variable inputs; frozen revisions use their corresponding template branch and configuration. Section D.2 connects fixtures and instructional cases to one archived Spot delivery.

Request wrapper and variable inputs. Both clients place the task, agent, and skill documents inside a data payload, with a common developer instruction that constrains the response format:

> **Shared response contract**
>
> You compile task-conditioned Agent Skills. Treat task, Agent, and Skill documents as data. Return only one valid `JSON` object without Markdown fences.

Routing receives the task, frozen agent description, capability demands, public evidence assets, and skill inventory. Global linking receives selected bodies, routing reasons, the active profile, and reviewed demand coverage. Clients validate response JSON; when an allowed formatting retry remains, they return the schema error and request a complete replacement object.

Semantic selection. The router admits a skill only when it adds executable support for a missing task capability. This criterion focuses selection on required procedures, as expressed in the selection and coverage instructions:

> **Skill selection and coverage**
>
> Select the smallest set that covers capabilities the task requires and the frozen Agent lacks. Drop irrelevant Skills instead of shortening and retaining them.
>
> A task mention, suggested baseline, objective description, or named method family is `task_declared`, not `task_supplied`.
>
> Return exactly one decision for every inventory Skill, including explicit `use=false` decisions. Omission is invalid because it is not evidence that the Skill was reviewed.

Each selection records inclusion, confidence, reason, demand identifiers, support type, and transferability. Coverage records attribute obligations to the task, agent, runtime, or selected skills, distinguishing executable support from a method merely named in the task.

Global linking. Global linking places shared control in one coordinator to avoid overlapping control loops while retaining each skill's local methods and constraints. The linker receives all selected bodies together and is instructed to:

> **Global linking constraints**
>
> Freeze the Agent. Modify only the selected Skill bodies.
>
> Choose exactly one primary Skill to carry the global execution DAG.
>
> Do not repeat planning, verification, stopping, recovery, or safety prose across Skills.
>
> Do not add facts absent from the supplied task, Agent, or Skills.

The plan specifies a primary skill, phase dependencies, advancement conditions, local methods and constraints, and success conditions. Task-mandated invariants remain distinct from source heuristics that public evidence may reject. Compact, ordinary, and scored-objective templates are separate branches; the last distinguishes successful execution from objective convergence. Reproduction requires the branch and configuration actually used.

Other compilation operations. Unit-level relation analysis and residual optimization are alternative reference interfaces. Their templates classify units against the agent contract and revise eligible retained blocks under action-order, verification, recovery, and critical-term constraints. Neither consumes task outcomes or execution trajectories. Parsing, threshold checks, protected-content copying, rendering, and runtime rules are deterministic.

D.2 Skill Compilation Examples

The following cases distinguish coverage-dependent retention, duplicate consolidation, and reinforcement. Coverage uses the reference linker's test strings and supplied relations; duplication is an instructional extension evaluated with that linker. Scores and identifiers are supplied fixture inputs. Metadata is omitted from the displays; the final example instead traces an archived Spot skill through delivery.

Figure 3 summarizes shared coordination and residual support construction. The fixtures below examine the reference actions and removal threshold; Appendix G.1 reports which actions occurred historically.

Coverage-dependent retention. This fixture shows how coverage-based simplification preserves domain constraints. A coverage score of 0.99 removes the planning instruction, whereas the domain statement remains under a proposed relation with the same score of 0.99. At a planning score of 0.80, the default 0.95 removal threshold instead retains the instruction.

Source body	Residual body and decision
`Plan the route.`	Empty body: remove the generic planning unit when evidenced coverage is 0.99.
`Plan the route.`	`Plan the route.` Keep the same unit when coverage is 0.80.
`The pressure limit is ‘50 PSI‘.`	`The pressure limit is ‘50 PSI‘.` Keep the domain statement despite the proposed coverage edge.

The domain guard preserves the fixture's literal pressure constraint despite generic coverage evidence. Unit removal changes the instruction body while preserving the skill's identity and resources.

Duplicate consolidation and reinforcement. Duplicate consolidation removes repetition while retaining a source-linked representative. A duplicate relation of 0.99 directs the second of two identical planning units to the first:

Source bodies	Linked residual bodies
Unit A: `Plan the route.` Unit B: `Plan the route.`	Unit A: `Plan the route.` Unit B: empty; merged into A.

The merge record preserves both source identifiers and the canonical destination. The retained representative supplies the shared planning obligation.

The reinforcement fixtures exhibit both an added reminder and retained guidance. For an unsupported batching state, the compact renderer inserts the following instruction:

> **Batching support instruction**
>
> Batch each phase into one foreground command. Let long commands wait internally and return one final summary; never poll them with extra model requests.

The no-profile fixture omits this reminder. By contrast, the active compiler's verification fixture labels an unstable capability `reinforce` but keeps `Run the verifier once.` unchanged. The two fixtures illustrate reinforcement through added or retained scaffolding.

An archived source-to-residual example. The archived August 27, 2026 Spot R13 execution (`spot-scheduler-traces`, `ours__full_capability`, attempt 1) includes source and delivered versions of `constrained-optimization`. Its source covers constraint classification, Lagrangian/KKT methods, penalty/barrier methods, and SciPy calls. Compilation assigns it `local_residual`, with `production-scheduling` as coordinator.

The following excerpts compare selected source and delivered passages.

Archived source excerpt	Delivered residual excerpt

`Inequality: g(x) <= 0`
`Bounds: l <= x <= u`
`Complementary slackness: mu_i * g_i(x) = 0`
	The policy must satisfy a hard feasibility constraint: finish required work before the deadline when possible.
	Switching decisions are constrained by a per-step overhead that changes the effective work rate.

Global linking conditions local methods on the task and assigns shared control to the coordinator. The linking plan introduces the two task-specific residual constraints. The plan names "Penalty-aware constrained control," "Thresholded feasibility checks," and "Horizon-aware tradeoff evaluation." The delivered document retains these names, source inequality and bound expressions, method headings, and selected tool commands. It excludes a separate local plan, verifier, recovery loop, or background process.

Active compilation then records six decisions: two MERGE records for duplicates, one REINFORCE record retaining the two task-conditioned constraints, and three KEEP records for candidate methods, evidence, and anchors. The two merged units contain 194 and 110 decoded characters, with canonical units retained in `ai-ml-timeseries`. The Reinforce record preserves the 211-character constraint unit unchanged. The linked input decreases from 1,816 to 1,512 decoded characters, counted at the active-compilation boundary. Unit IDs refer to the intermediate linked document. Reinforce cites an unsupported or unstable capability but adds no procedure here. Figure 14 accounts for the six units and their decoded-character reduction.

The delivered file's SHA-256 matches the active output and frozen skill inventory; its source hash matches the global-link manifest.

E CASE STUDIES

These revision-specific cases connect runtime decisions to artifact retention (Appendix A.4), within the descriptive evidence scope in Appendix C.2.

E.1 END-TO-END TASK EXECUTION

Spot task and candidate progression. The Spot R13 policy uses only the visible trace prefix under deadline/cost constraints. Figure 7 and Table 10 separate public candidate measurements from final official evaluation. The trace distinguishes provider requests from candidate-generation and validation events.

Event	Request	Time (min)	Score	Recorded status
Candidate 1	2	4.30	0.8992	initial
Candidate 2	4	11.29	0.9426	invalid
Candidate 3	5	13.02	0.9365	invalid
Candidate 4	6	15.57	**0.9599**	improving
Candidate 5	7	17.48	0.9509	invalid
Candidate 6	8	21.06	0.8150	non-improving
Candidate 7	9	23.25	0.9486	non-improving
Candidate 8	10	26.74	0.8756	non-improving
Final artifact	10	27.60	0.9585	official evaluation

Table 10: Spot R13 candidate progression and final evaluation. Times are elapsed minutes. **Bold** marks the highest public search score; the final official result is separated below.

Candidates 2, 3, and 5 are recorded as invalid despite higher scores than candidate 1, whose validation is unrecorded. Candidate 4 remains the public maximum after four further requests; the last three candidates are non-improving.

Batch evidence and termination. Eight candidates span five batches. The receipt confirms two consecutive genuine batches without qualifying global-best improvement and a valid stagnation window. Batch qualification determines the counting unit, excluding failed or metadata-only steps un-

less they meet the genuine-batch criteria. All ten requests were admitted, and the run ended after the recorded stagnation window.

Retention and final measurement. Final evaluation uses the restored best artifact; its current and retained identities match a score-consistent receipt. The run continued for 12.03 minutes after the public maximum. Estimating stopping savings requires a matched continuation, which is unavailable; a time-to-target comparison also requires intermediate baseline evaluations, which Structured Local lacks.

E.2 Runtime Decisions and Artifact Retention

R57: a denied continuation request. R57 illustrates request-value exhaustion. The success-first controller admits the first three of four decisions, then returns `stop_no_value` on the fourth. Its non-retry ledger records three provider requests and no expansion request. The receipt identifies one optimization batch, no consecutive non-improving batches, and an invalid stagnation window.

Case	Recorded execution outcome	Evidence for the final artifact
Spot R13	Score-stagnation stop; all ten request decisions admitted	Best artifact restored; retained and current identities agree; official reward 0.9585.
DuckDB R57	Fourth request decision denied after three admissions	Best artifact retained with an authoritative receipt; official reward 0.7706.
DuckDB R13c	Generated code and supporting artifacts evaluated	Complete artifact contract; official reward 0.7924; evaluation time included.

Table 11: Execution and finalization outcomes in the task cases. Rows identify the revision used for each case.

In R57, the authoritative receipt matches the current and retained identities and scores, so the artifact is ready for evaluation without restoration. All four required files are present: `solution.patch`, `bench_results.json`, `JOURNAL.md`, and `optimization_state.json`.

R57 retains an evaluation-ready artifact when continuation is denied. Its qualified fallback remains unused, and the execution uses neither source-tree injection nor compaction. This supplementary case uses a different revision and protocol from main-panel R13c.

R13c: final evaluation of the generated code. R13c completes final evaluation using the existing code and supporting artifacts. Generation takes 194.41 minutes and subsequent evaluation 16.16 minutes, for a rounded total of 210.57 minutes; Table 11 reports the complete contract and reward. The evaluation adds no model generation, provider requests, or tokens. Appendix B gives recovery and accounting details.

F Execution Records and Resource Accounting

F.1 Archive Composition and Record Binding

The census combines record-name and records-directory JSON enumeration with the 24 main-table record locators, preserving paths, hashes, campaigns, tasks, conditions, and attempt labels. It covers seven task identifiers: 766 files for the main three tasks and 91 for four others. Identical files may be copies; a campaign–task–condition–attempt label may contain revisions or later evaluations. Table 12 therefore distinguishes paths, bytes, and labels rather than counting independent trials.

Inventory item	Count
JSON paths in the combined record enumeration	1,109
Files with execution-record fields	857
Distinct record byte hashes	831
Distinct campaign/task/condition/attempt labels	829
Labels associated with multiple record hashes	2
Selected endpoint record hashes located	24
Files matching a main-table record hash	48

Table 12: Scope of the offline archive census. File and label counts describe the inspected archive; only the 24 main-table endpoints enter the main comparison.

The inventory includes development restarts, revisions, and evaluation-only resumes. The 24 designated record hashes bind main-table outcomes to this archive; Table 8 lists the corresponding attempt identifiers.

F.2 Request-level resource records and deliverable coverage

The 24 main-table endpoints reference 1,247 provider requests: five marked retries and 12 with missing usage. The accompanying `runtime-evidence.csv` retains request IDs, source lines, timestamps, retry flags, usage, and API duration. Missing usage remains unknown, making affected totals lower bounds; request-level measurements summarize their parent executions.

Method	Generals	Spot	DuckDB
Original	1/27	1/23	1/113
No Skill	0/26	1/26	0/107
Truncation	0/24	0/23	1/110
Random removal	2/28	0/25	1/109
LLM compression	0/27	0/25	0/107
LLMLingua-2	0/46	0/25	0/107
Structured Local	0/26	0/25	0/64
DeltaSkill	1/10	0/10	3/134

Table 13: Missing-usage request counts / total provider requests, including retries, for every main-table endpoint. A zero numerator indicates complete provider-request usage coverage.

Shared-contract completion is 8/8 for Generals, 8/8 for Spot, and 1/8 for DuckDB; each denominator comprises eight method conditions. This yields 17 reward-eligible endpoints, seven N/A rewards, and two genuine zeros. Figure 13 explains DuckDB eligibility separately from token-usage coverage.

The panel compares complete configurations. All 24 endpoints use `gpt-5.4-mini` with medium reasoning, but command limits differ (Table 7). DuckDB DeltaSkill additionally uses R38 accelerated-search support and an artifact-only official-verifier resume; its seven baselines record neither support nor recovery and use ordinary official evaluation. These settings define the complete configurations compared in this panel.

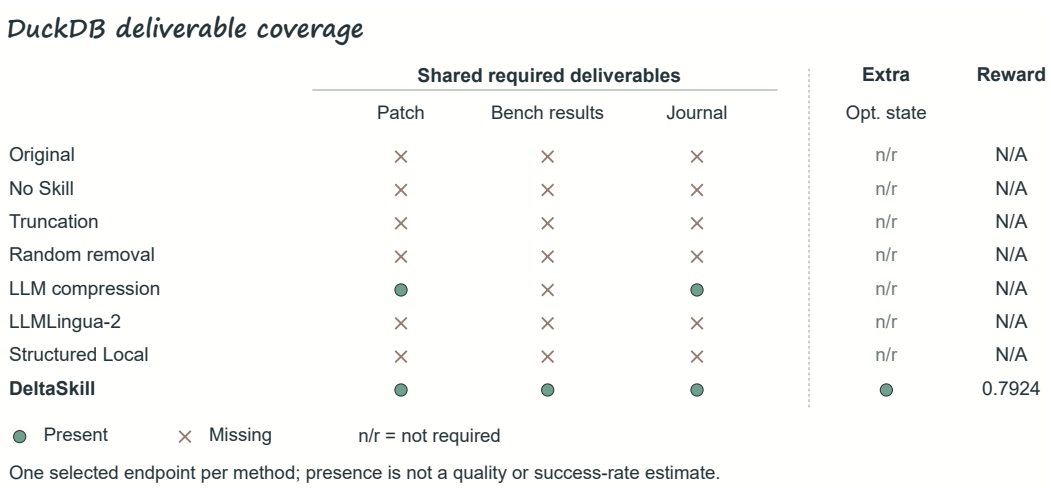


Figure 13: Recorded deliverable coverage at the eight DuckDB endpoints. The three shared deliverables are `solution.patch`, `bench_results.json`, and `JOURNAL.md`. LLM compression records a patch and journal but lacks benchmark results; the other six baselines lack all three shared deliverables. DeltaSkill records all three. Its additional `optimization_state.json` is shown separately because it is not a shared baseline requirement. Cells indicate deliverable presence in the archived artifact-contract records, with one endpoint per method.

G Compilation Effects and Structural Validation

G.1 Archived Compilation Decisions and Text Effects

We trace active compilation at the three reported DeltaSkill endpoints from selected records through full and active manifests to the six deployed Skills. All recorded hashes, source spans, output hashes, and registered evidence identifiers agree across this chain. Source documents use the implementation's JSON-string hashing convention; files use raw-byte SHA-256. This chain connects the compiler's decisions to the procedural support delivered to the agent.

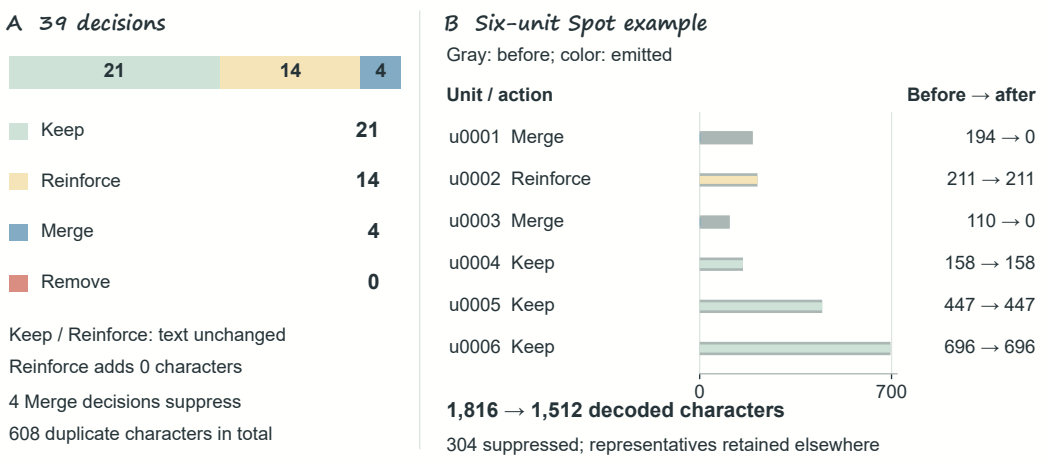


Figure 14: Archived compilation decisions and their text effects. (A) The three endpoints contain 21 Keep, 14 Reinforce, and four Merge decisions, with no Remove or Precedence. All Reinforce outputs retain their input text. (B) Gray bars show source lengths and colored bars emitted lengths in the six-unit Spot example: 1,816 becomes 1,512 characters as two duplicates are suppressed while their representatives survive in another Skill. Counts represent decisions; lengths measure decoded control-unit characters at the active-compilation boundary.

Recorded transformation effects. All 39 decisions are retention or duplicate consolidation (Figure 14). The 21 Keep and 14 Reinforce outputs match their source text exactly; Reinforce adds zero characters at this boundary. The four Spot Merge records suppress two coordinator and two candidate-family duplicates, totaling 608 decoded characters, while retaining canonical units and pairwise source identities. No Remove or Precedence occurs.

Global linking has already assigned shared coordination before this stage. Active compilation then consolidates duplicate units within that linked residual. The reported text changes cover this stage; routing, formatting, and stage activation are separate transformations.

A complete six-unit example. In Spot's `constrained-optimization` Skill, Merge suppresses 194 and 110 characters whose representatives remain in `ai-ml-timeseries`. Reinforce preserves the 211-character constraint unit; Keep preserves units of 158, 447, and 696 characters. The emitted total is thus 1,512 of 1,816 input characters (Figure 14). The candidate-family passage also occurs in `parameter-optimization`: its two suppressed copies have local identifier `u0003`, Merge score 1.0, and exact-text evidence linking each to the surviving source. The machine-readable audit preserves this source/output record for every decision. This active-compilation example follows the earlier source-to-residual transformation in Appendix D.2.

G.2 Offline Compiler Regression and Adversarial Fixtures

We test the unmodified `r13c5` relation linker, provenance builder, and safety gate on 19 deterministic, author-written fixtures. Constructed units, evidence identifiers, expected invariants, and source hashes are fixed before execution. The fixtures provide network-free regression checks of this reference implementation.

Fixture group	Pass / cases
Ordinary protected, duplicate, coverage, conflict, and endpoint rules	10 / 10
Missing or duplicate decision coverage	2 / 2
Unknown evidence and tampered provenance	2 / 2
Missing critical term in retained output	1 / 1
Unknown declared dependency	1 / 1
Altered protected text with an unchanged Keep action	0 / 1
Two-unit duplicate cycle with both representatives suppressed	0 / 1
Removal of a dependency while its dependent is retained	0 / 1
All fixtures	16 / 19

Table 14: Complete grouped outcomes of the 19 offline fixtures. A pass means that the constructed case satisfied its declared invariant. Each failing group contains one explicit counterexample.

Structural counterexamples. Three counterexamples separate consistent provenance from preserved obligations. The gate accepts a supplied Keep decision that changes protected text, because it checks the action without comparing all bytes; the ordinary linker itself preserves that text. Reciprocal duplicate edges can suppress both representatives. A coverage edge can remove a retained unit's dependency because endpoint validity does not enforce retention closure. Registered synthetic evidence satisfies identifier checks without certifying relation semantics. These fixtures expose three unsafe transformations accepted by the gate. Section G.1 separately analyzes the 39 archived decisions.

G.3 Separately Versioned Structural-Guard Validation

A separately versioned offline candidate requires protected outputs to match the IR unit's UTF-8 text, duplicate chains to terminate at nonempty surviving representatives, and declared dependencies of surviving obligations to retain representatives. Structural failure rejects the transformation and returns the original stack text. The linker, relation construction, provenance API, and 0.95 thresholds remain unchanged. Task outcomes use the archived guard; this candidate is evaluated offline.

All 16 previously passing fixture results remain unchanged; the three counterexamples now reject. The independent structural oracle agrees on 1,280 accepted and 2,816 rejected exhaustive graphs,

Check	Candidate outcome
Original fixtures (unchanged reference: 16/19)	19/19
Additional regression tests	18/18
All three-unit decision/dependency configurations	4,096/4,096
Fixed-seed random graphs with 2–12 units	1,000/1,000
Order-invariance checks on the random graphs	1,000/1,000
Archived/reconstructed compilation-package replays	12/12

Table 15: Offline validation of the structural-guard candidate. The original 19 fixtures retain their definitions and expected outcomes. Graph agreement is checked against an independently implemented structural oracle; counts represent constructed structural checks.

and 367 accepted and 633 rejected random graphs. Rejections return the original unit text. The finite checks cover protected-text equality in the parsed IR, surviving duplicate representatives, and closure over declared dependency edges.

Compatibility with the four-cell compilation packages. For each Spot cell, we replay the candidate on an offline reconstruction, the archived original compilation, and the post-normalization package, after checking source documents, profile registries, IR, relations, and provenance. All 12 judgments accept the unchanged decisions. These packages contain no protected units or explicit dependency edges: they establish compatibility, including representative-preserving merges, while the fixtures above exercise the three defects. The replay scope comprises IR, decisions, and provenance.

H Profile-Conditioned Support and Controller Validation

H.1 Profile-Matching Comparison on Spot

The September 24 Spot panel compares profile-conditioned support for Terminus-2 and OpenCode. Each cell uses GPT-5.5 with medium reasoning, 2 CPU cores, 4096 MB memory, and a 10,800-second agent window. It retains the R13 core and frozen evaluator with a separately versioned support and execution adapter, distinct from the primary GPT-5.4-mini panel and the earlier diagnostic panel below.

Intervention. Matched supplies the receiver's measured profile; mismatched supplies the other receiver's measured profile through an identity-checked experimental override, without synthetic corruption. For task-required capabilities, unstable or unsupported states activate planning, batching, state-reuse, or precedence support; reliable and unknown states leave it inactive. The selector uses neither receiver names nor observed scores. Within each receiver, the base Skill body and common tools are fixed: the intervention is support selection, with no profile-driven Remove action.

Verified intervention inputs. Independent reconstruction recovers all four original compilation packages, including profile identities, source documents, support selection, and the separately versioned objective normalization. Each receiver's deployed Skill bodies and common instructions are byte-identical across conditions; the stacks contain five Skills for Terminus-2 and six for OpenCode. Each conditional instruction in Table 16 occurs once in the original run's first provider request and never in its Skill files.

Receiver	Condition	Profile source	Support	Bytes
Terminus-2	matched	Terminus-2	P, B, S, C	1,068
Terminus-2	mismatched	OpenCode	B, S	554
OpenCode	matched	OpenCode	B, S	554
OpenCode	mismatched	Terminus-2	P, B, S, C	1,068

Table 16: Verified conditional support in the four original Spot executions. P: planning; B: batching; S: state reuse; C: precedence. Byte counts cover conditional guidance encoded in UTF-8, not model tokens. Each instruction is independently bound to its measured donor profile and original first request.

The profiles cover eleven capabilities with 22 and 20 unique paired observations. Only planning and precedence change the selected guidance, by 514 bytes; state reuse has different posterior means but the same unstable label. Matching therefore adds these two supports for Terminus-2 and omits them for OpenCode: the same selection rule produces receiver-specific guidance. Input checks establish delivery of that guidance in the original requests.

Profile-source comparison. Table 2 reports one development outcome per cell, with matched–mismatched reward differences of 0.0104 for Terminus-2 and 0.0404 for OpenCode. The displayed contrast compares the two profile sources. The complete six-condition record, including no-profile controls, is supplied in `evidence/cross-profile-20260924.json`.

Evaluation record. The Terminus-2 matched outcome uses saved-artifact continuation with a fresh controller and conversation under an execution-repair overlay, followed by one artifact-only evaluation with the frozen verifier after provider interruptions. Its contrast includes both profile and recovery conditions. The supplementary record preserves censoring, the false metadata-validity flag, original failures, and cumulative costs.

Input verification covers the four original first requests. The continuation's complete conversation and common guidance fall outside this check. Available calibration evidence establishes profile identities and input bindings; independent replay of all raw calibration traces remains unavailable.

H.2 Archived Cross-Profile Diagnostic Panel

The earlier six-cell `R13-profile-v6` panel supplies development context with one matched, exchanged, and absent-profile run per receiver. It uses GPT-5.5 with medium reasoning, 2 CPU cores, 4096 MB memory, and a 10,800-second search limit under a separate method identity. Its profiles contain 22 and 20 unique observations (44 and 40 probe variants). Each receiver's three compiled bodies differ; absent-profile inputs retain Skills and the shared helper, unlike No Skill. Exchange explicitly overrides receiver–donor equality while validating identities and donor evidence. Matching names the measured base configuration, not an identity of calibration and execution protocols. Appendix I.3 examines the archived input construction.

Receiver	Matched	Exchanged	No profile
Terminus-2	N/A	N/A	0.9624
OpenCode	$0.8558^{\dagger}$	0.9684	0.9563

Table 17: Archived cross-profile outcomes, with one development endpoint per cell. Terminus-2's two N/A entries reflect unrecovered transport failures. † After an input-transfer failure, the frozen evaluator verified the four unchanged original OpenCode artifacts in an artifact-only recovery with zero model requests and zero agent reruns. Original invalid records and all costs are retained.

Archived OpenCode matched minus exchanged is -0.1126, and matched minus absent profile is -0.1005. After artifact-only recovery, rewards are eligible for 1/3 Terminus-2 and 3/3 OpenCode conditions; the denominators count receiver conditions. Later revisions remain distinct: for example, utility-dev1 selected the same `00` arm for both receivers, so its within-receiver labels did not create different treatments.

Receiver	Profile	HTTP	Observed tokens	API min	Cell min
Terminus-2	matched	4	$\geq$ 48,458	7.87	10.74
Terminus-2	exchanged	4	$\geq$ 48,306	8.74	10.75
Terminus-2	none	9	237,417	19.46	22.84
OpenCode	exchanged	13	408,425	21.27	25.54
OpenCode	matched	29	$\geq$ 963,551	40.94	46.61
OpenCode	none	25	$\geq$ 761,544	20.60	26.62

Table 18: Resource accounting for all six archived development attempts, including invalid outcomes. HTTP counts include retries; token lower bounds retain unknown usage. API time sums transport attempts, while cell wall time includes setup. The separate OpenCode artifact-only recovery adds 3.42 seconds including setup (2.03 seconds in verification), with no model request.

H.3 Offline Controller and Retention Checks

We replay the archived controller and fixed-continuation overlay on 16 synthetic evidence cases and eight existing runtime tests with networking disabled. Both policies receive the same evidence fingerprint; an admission override retains the native adaptive decision. Table 19 tests the decision and retention contracts of these policies.

Checked contract	Passed / cases
Initial, terminal, local-work, unchanged/new-evidence and context dispatch	8/8
Incomplete delivery and bounded completion admission	4/4
New versus repeated response-format and source-discovery failures	4/4
Native decision retained under fixed-continuation override	1/1
Strict identity, counter and ledger-corruption rejection	1/1
Validity adapter changes only declared intervention identity	1/1
Disabled adaptive stop retains optimization state	1/1
Residual-off export retains routing and the common helper	1/1
Cancellation completes bounded finalization before context capture	1/1
No-op/restore and strict fixed-budget endpoint certification	1/1
Missing-candidate restoration and bounded finalization failure	1/1

Table 19: Offline checks of controller and retention contracts across 16 synthetic decision cases and eight existing runtime tests, with no new model or task-verifier calls. Passed/cases counts declared inputs or test functions; the table reports software compliance for the specified test set.

I Offline Resource and Profile Analyses

We decompose recorded resource use across all 24 selected endpoints, using Structured Local as a common comparator, and examine how the two archived R13-profile-v6 profiles determine inputs and respond to evidence deletion. The profile revision remains separate from the main endpoint panel. These analyses operate on stored records: request distributions summarize within-run variation, and deletion perturbations measure sensitivity in a fixed evidence archive.

I.1 Request Count and Observed Request Size

Observed token use satisfies $T = n\bar{t}$, where n counts requests with known usage and $\bar{t}$ is their mean input-plus-output token count. The accounting includes retries and leaves missing usage unknown. Figure 15 compares DeltaSkill and Structured Local across tasks. DeltaSkill's coverage is 9/10 on Generals, 10/10 on Spot, and 131/134 on DuckDB; the comparator traces are complete. Mean request size is computed over known-usage requests.

For complete-usage Spot, the symmetric identity

$$T_D - T_S = (n_D - n_S)\frac{\bar{t}_D + \bar{t}_S}{2} + (\bar{t}_D - \bar{t}_S)\frac{n_D + n_S}{2}$$

separates request-count and mean-size contributions exactly before rounding; D denotes DeltaSkill and S Structured Local. DeltaSkill has fewer input tokens per request (27,692.2 versus 32,243.2) but more output tokens (10,287.3 versus 3,277.96). The two terms quantify request-count and mean-size contributions to the observed token difference. For Spot, the reduction in request count outweighs the higher mean input-plus-output size, yielding lower total token use.

Using $n_S = 25$, $n_D = 10$, $T_S = 888{,}029$, and $T_D = 379{,}795$ gives $\bar{t}_S = 35{,}521.16$ and $\bar{t}_D = 37{,}979.50$ tokens per request. The count term is $-551{,}254.95$ tokens and the mean-size term is $+43{,}020.95$ tokens; their sum is $-508{,}234$. The 6.9% larger mean request partly offsets the savings associated with fewer requests. This identity is an accounting decomposition of the two observed traces. It does not isolate the causal effects of compilation and stopping, either of which can change both request count and size.

I.2 Request-Window Timing and Duration Distributions

All 1,247 attempts have start/end timestamps and durations; intervals are ordered and non-overlapping within each trace. The first-to-last request window therefore partitions into provider

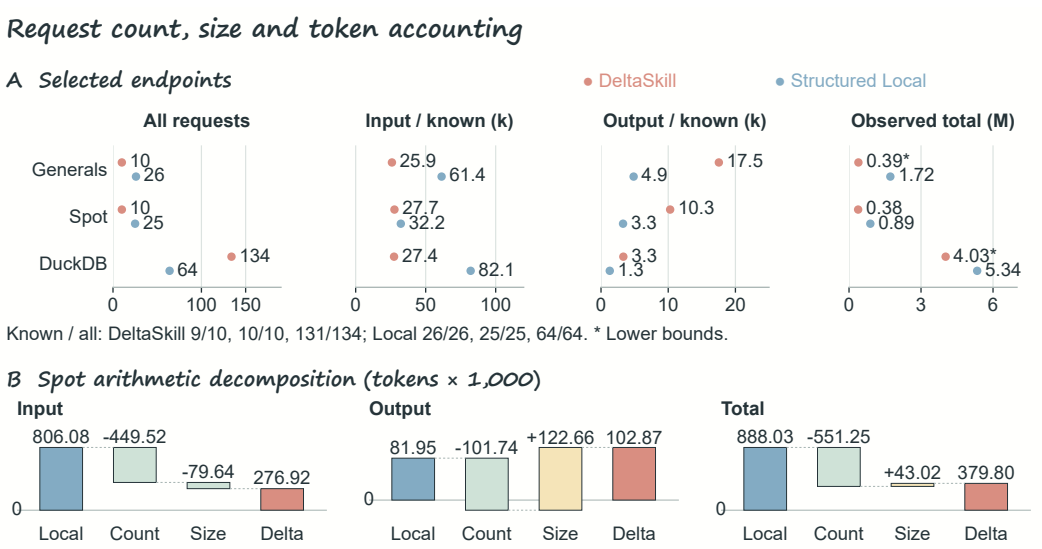


Figure 15: Observed token-use decomposition. (A) All requests, mean input and output tokens per request with known usage, and observed totals for the three task pairs. Coverage distinguishes known usage from all requests; asterisks mark incomplete totals. (B) Spot's input, output, and total tokens decompose from Structured Local to DeltaSkill through signed request-count and mean-size contributions. The identity is exact before rounding. Counts include recorded retries; the decomposition describes the observed executions.

intervals and intervening gaps, excluding earlier preparation and later activity. Timestamp sums agree with recorded durations within 0.01 seconds per trace. The derived table records these shares, median and 90th-percentile duration, and the API-time share of the slowest rounded-up 20% of requests (Figure 11).

On Spot, these windows span 26.56 minutes for DeltaSkill and 175.97 minutes for Structured Local; adjacent gaps range from 2.29–11.16 and 128.66–619.57 seconds. The gaps are unclassified intervals that may contain idle time, tool execution, or other overhead. The configurations in Appendix B also differ in task-adapted settings. Duration distributions summarize requests within each execution.

I.3 Profile Evidence and Actual Compiled-Input Differences

The R13-profile-v6 compilation receipt links its two measured profiles to the six diagnostic inputs in Appendix H.2. We verify the bundled receipt and profile hashes, their recorded bindings, and whether donor conditions produce different body hashes within each receiver. This earlier input construction remains distinct from the September 24 support adapter.

Both receivers use the same donor mapping (Figure 16A). Terminus-2 selects planning and batching with 1,793 appended UTF-8 bytes; OpenCode selects batching with 1,357. No profile selects no conditional obligation but retains 857 bytes of shared material. The profile-dependent increments are 936 and 500 bytes, and all three body hashes differ within each receiver. The sizes measure appended UTF-8 bytes. The available receipts establish support selections, append sizes, and distinct body hashes; complete six-cell overlay texts are unavailable.

I.4 Shared Evidence and Single-Observation Deletion Sensitivity

We first reconstruct all 22 capability entries from evidence identifiers and success/failure fields under $\mathrm{Beta}(1,1)$, recovering the stored parameters and labels. The bound updater uses reliable/unsupported thresholds of 0.75/0.25. We use the complete-profile builder's minimum evidence count of one; the generic default of two gives the same original and perturbed labels for these records.

We then omit each of the 42 distinct observations from every referencing capability and recompute all 11 beliefs, preserving shared-evidence structure. All 42 perturbations and 462 capability com-

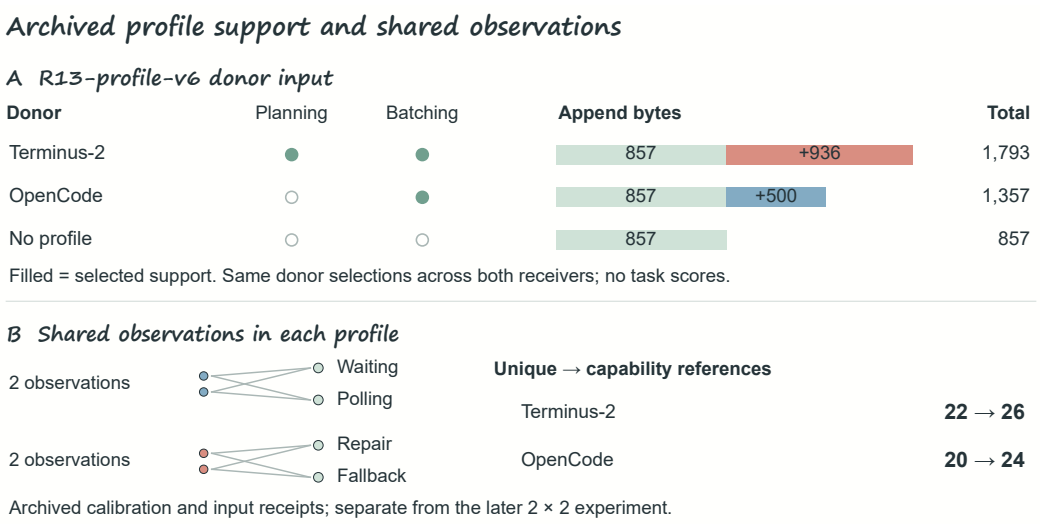


Figure 16: Profile-conditioned inputs and shared evidence in the archived v6 panel. (A) The same donor-to-support mapping applies in both receivers. Appended UTF-8 lengths include 857 shared bytes retained without a profile. (B) Terminus-2 and OpenCode have 22 and 20 distinct observations but 26 and 24 observation-to-capability references. In each profile, two observations jointly support waiting/polling and another two support repair/fallback; shared references retain the identity of their underlying observation.

parisons are retained. The prior and remaining evidence are fixed; acquisition and compilation are not rerun. This measures deletion sensitivity in the fixed evidence archive.

Profile	Records	Links	Changed/omitted	R–U	U–R	U–S
Terminus-2	22	26	14/22	14	4	0
OpenCode	20	24	16/20	18	1	1

Table 20: Sensitivity to deleting each unique observation from all capability entries that reference it. Changed/omitted counts perturbations with at least one changed state over all omitted records. R–U, U–R, and U–S count capability-state transitions, so a shared record can contribute more than one transition. R: reliable; U: unstable; S: unsupported.

All seven reliable Terminus-2 and nine reliable OpenCode labels depend on both supporting successes: two give mean 0.75, while one gives two-thirds. Deleting a failure promotes four unstable Terminus-2 capabilities and one OpenCode capability. For OpenCode state reuse, deleting its sole success leaves two failures and an unsupported label. These threshold crossings quantify leave-one-observation-out sensitivity of the stored capability labels.

I.5 Failure Accounting and Missing-Usage Sensitivity

All 1,247 provider attempts across the 24 endpoints enter resource accounting, including ineligible rewards. Usage is known for 1,235 and missing for 12. Failures are non-success HTTP statuses or incomplete responses; retries use the recorded retry flag. Twelve failures and five retries overlap in two attempts, giving 15 distinct attempts. Their API durations are 1,344.74 and 326.05 seconds, minus 121.21 seconds of overlap: 1,549.59 seconds, or 2.84% of the 54,623.37-second total. These values account for incurred API durations. All failures lack usage; three retries contribute 178,314 observed tokens and two lack usage, leaving a lower-bound retry total.

Table 21 gives the missing-token totals that would close each observed gap to Structured Local, including known retries. Generally, DeltaSkill's total is lower only when its unknown usage minus the baseline's unknown usage is below the observed baseline-minus-DeltaSkill gap. The ordering depends on the unknown-token totals, for which a verified finite upper bound remains unavailable.

The supplement retains all 21 DeltaSkill–baseline contrasts and 24 per-run accounting rows, with failed-attempt costs and unknown usage. Missing tokens remain unknown. Resource completeness is separate from reward eligibility: 17 endpoints have eligible rewards, including two genuine zeros; seven remain N/A.

Task	Token total DeltaSkill	Structured Local observed tokens	Usage coverage DeltaSkill / SL	Tie threshold unknown tokens
Generals	$\geq$ 390,855	1,721,591	9/10 ; 26/26	1,330,736
Spot	379,795	888,029	10/10 ; 25/25	none missing
DuckDB	$\geq$ 4,025,190	5,339,360	131/134 ; 64/64	1,314,170

Table 21: Unknown-token totals needed to tie Structured Local, whose usage is complete. Coverage includes retries. DeltaSkill's observed lower-token ordering holds strictly below each threshold and reverses strictly above it; the thresholds define sensitivity boundaries for the unknown usage. Spot's 508,234-token difference is fully observed. DuckDB Structured Local retains an ineligible reward (N/A), independently of its complete resource accounting.

J Search Trajectories and Tool Use under Profile Matching

Candidate trajectories and tool returns reveal how the four outcomes in Table 2 emerge during search. We reconstruct these events from source-linked records, with requests, calls, candidates, and receipts nested within each reported outcome. Appendix H.1 specifies the Terminus-2 matched continuation and artifact-only verification provenance.

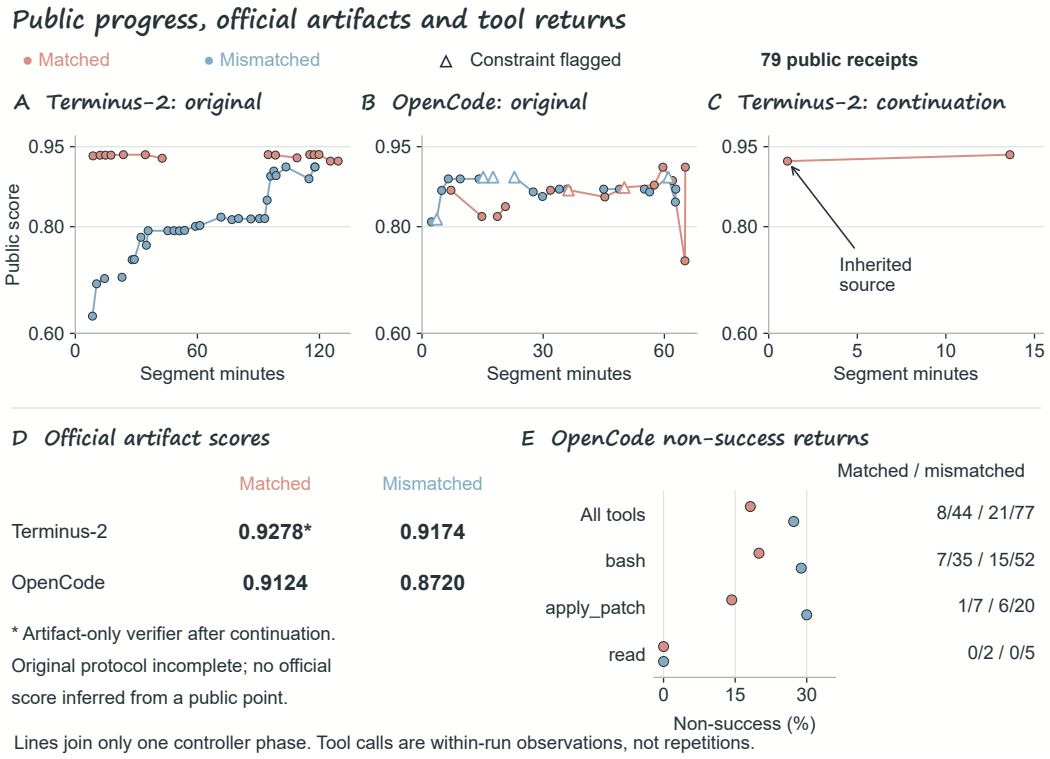


Figure 17: Process evidence for the four-cell Spot comparison. (A–C) All 79 source-bound public measurements, including hollow constraint-flagged points; axes use time within each segment, and lines break at controller-phase boundaries. C uses inherited artifacts with a fresh controller and conversation under an execution-repair overlay. (D) Separate official artifact evaluations. The starred Terminus-2 matched outcome comes from artifact-only verification after continuation and provider interruptions; the original protocol remained incomplete and its metadata-validity flag was false (Appendix H.1). (E) OpenCode non-success returns by tool, with exact denominators from one execution per condition. The curves show per-candidate measurements; exploratory tool-return summaries describe the same executions.

J.1 Candidate Measurements and Trajectory Reconstruction

Each included host-validation receipt binds candidate-source and metric-file bytes. The 79 receipts comprise 15 original and two continuation measurements for Terminus-2 matched, 29 for Terminus-2 mismatched, and 13/20 for OpenCode matched/mismatched. Table 22 distinguishes measurements from unique sources; potentially overlapping screen logs remain a separate layer.

Receiver	Profile	Receipts	Sources	Repeats	Flagged
Terminus-2	Matched	15	15	0	0
Terminus-2	Mismatched	29	28	1	0
OpenCode	Matched	13	10	3	2
OpenCode	Mismatched	20	13	7	5

Table 22: Original-execution host measurements for the four profile conditions. Repeated-source counts are additional measurements after the first occurrence of each source in that execution segment. Constraint-flagged measurements remain in the record.

In Figure 17C, the first continuation receipt revalidates inherited source under a new identifier and timestamp; the second binds new source. Across both matched Terminus-2 segments, 17 receipts therefore represent 16 distinct sources. Intermediate rankings also differ from the final comparison: OpenCode changes which condition leads, while Terminus-2 already differs at its first bound measurement (0.9328 matched versus 0.6323 mismatched). Following the trajectories distinguishes candidate exploration from the final artifact selected for evaluation; each condition has its own search length, programs, and encountered failures.

J.2 Native Tool Returns and Observed Failure Causes

OpenCode calls are deduplicated by session and call identity; definitions, unexecuted command text, and repeated status records are excluded. Nonzero bash returns and native tool errors count as non-success, even when the outer message is marked complete. Matched has 8/44 (18.2%) and mismatched 21/77 (27.3%). Figure 17E includes every observed tool type: bash 7/35 versus 15/52, patch 1/7 versus 6/20, and read 0/2 versus 0/5. This stratification follows inspection of the totals. These descriptive fractions summarize operations of varying difficulty and search phase.

Command-and-output inspection classifies all 29 returns (Table 23); none is solely an expected negative grep/test result. Ten mismatched returns reject the visible task-contract hash in the candidate launcher, distinct from final four-artifact verification; six reflect patch-context errors. Matched instead has more syntax errors and candidate nonzero returns. These diagnostics classify the observed operational failures. Their origins in generated arguments, integration, or infrastructure remain unresolved.

Located cause	Matched	Mismatched
Task-contract hash rejection	0	10
Patch-context mismatch	1	6
Python syntax error	5	3
Candidate launcher reports nonzero return	2	1
Source-admission rejection	0	1
Total	8	21

Table 23: Output-grounded categories for every OpenCode non-success return. Counts describe the operations attempted in each observed trace; operation difficulty varies.

J.3 Observation Units and Supporting Actions

Terminus-2 records terminal batches, whose zero return does not establish success for every enclosed command. Table 24 preserves unknown status and the continuation boundary. The batch fractions vary by segment; Terminus-2 batches and OpenCode native calls are reported separately.

Segment	Batches	Zero return	Nonzero	Unknown
Matched, original	52	33	12	7
Mismatched, original	88	69	19	0
Matched, continuation	20	11	9	0

Table 24: Observed Terminus-2 terminal batches. Unknown return status remains a separate category. The continuation is a provenance segment of the matched condition.

The original executions contain 226 automatic host-support actions: 30/64 for Terminus-2 matched/mismatched and 50/82 for OpenCode, bound by request/output hashes and status. Of 85 checkpoints, 74 match an independently extracted host-validation identifier; 11 have only the helper's internal binding. Agent-level planning, registration-before-evaluation, dependency order, and state consumption remain unverified because their identity-and-order links are incomplete.

J.4 Recomputation and Integrity Checks

The extractor maps 577 deduplicated source files to 714 layered events. Its 24 behavior tests, 17 candidate tests, and 20 independent validation tests cover identities, duplicates, missing status, and cross-run isolation. All 997 original controller rows pass entry-hash and predecessor-chain checks, establishing consistency of the recorded hash chain.

The scoped Terminus-2 collection contains 324 files: 20 have prior audit digests, and 304 have collection-time digests. These latter digests identify the files available during the present inspection. Derived records preserve source locators; exports map original to normalized-path checksums.

K Compilation and Stopping Ablation

K.1 Factors and Shared Configuration

This experiment evaluates the two component choices in Table 3 using Spot, Terminus-2, GPT-5.5, and medium reasoning. The four conditions were frozen before execution and launched concurrently, with one run per cell. Each receives two CPUs and 4,096 MB of memory, a 10,800-second search budget, and up to 300 seconds for a shared local best-artifact finalizer. Task, evaluator, initial workspace, and bounded transport policy are common. There are no whole-trial retries in this batch.

The input factor supplies either compiled Skills with profile-conditioned support or the corresponding routed raw Skills. It includes instruction adaptation and execution support; it does not isolate deletion alone. Both packages retain the same five-Skill routed inventory, primary Skill, and common helper; the raw package includes the helper documentation but omits profile-conditioned reinforcement obligations. The stopping factor either enables the adaptive runtime or disables its adaptive request gate, score-stagnation stop, and public-boundary stop, adding a fixed-continuation instruction to search until the external deadline. Both retain the best eligible artifact for final delivery. The four runs reuse frozen profile and compiled-input bytes; neither calibration nor compilation is repeated.

The experiment uses R13-profile-v6 with a narrowly versioned stopping-integration repair: an adaptive runtime's `optimization_progress_blocked` signal is treated as terminal by the outer trial, instead of triggering an intermediate verification and another Agent phase. The fixed-budget trial method is unchanged. Controller decision rules, task scoring, and final verification are preserved. Both observed adaptive runs ended by the runtime's `optimization_stagnated` condition, each within its first phase; this is satisfaction of the implemented stopping rule rather than a proof of global convergence. The component panel has its own configuration identity and is analyzed separately from the primary GPT-5.4-mini results and earlier stopping-integration development runs.

K.2 Final Outcomes and Complete Execution Accounting

All four final artifact contracts are complete. Table 25 reports exact input and output tokens; every request has observed usage, including the final upstream responses cancelled at fixed-budget deadlines. Cached input remains part of input-token counts. Agent time includes execution and its local finalization, while API duration sums per-attempt transport intervals. These quantities exclude earlier profile acquisition and compilation, and are not monetary costs.

Under adaptive stopping, compiled support improves reward by 0.1478 and reduces execution tokens by 27.9%, while Agent time increases by 24.3%. Under fixed continuation, its reward difference is 0.0022 and token reduction is 31.3%. With compiled support, adaptive stopping uses 90.6% less Agent time and 91.9% fewer tokens, at a reward difference of -0.0081 relative to fixed continuation. The corresponding raw-Skill reward difference is -0.1537. The difference between the two input contrasts is 0.1456 reward units. This is a descriptive four-cell interaction; one run per cell does not

Cell	Input tokens	Output tokens	Total tokens	API (min)	Agent (min)
C/A	290,761	47,492	338,253	15.27	16.92
C/F	3,817,041	338,310	4,155,351	162.33	180.18
R/A	433,355	35,917	469,272	12.22	13.61
R/F	5,724,480	322,716	6,047,196	153.87	180.22

Table 25: Execution accounting for the component ablation. C: compiled support; R: raw Skills; A: adaptive stopping; F: fixed budget. All four cells have zero missing usage payloads and zero transport retries.

estimate uncertainty, repeatability, or quality non-inferiority. The fixed-budget runs are independent trajectories, not continuations branched from the adaptive runs, so these endpoint contrasts do not measure the additional gain available after a particular adaptive stop.

K.3 Deadline Verification and Record Binding

The two adaptive artifacts pass the original final-artifact audit. At each fixed-budget deadline, the SDK cancels an in-flight request. The upstream response finishes successfully, but its later delivery meets a closed consumer, which the original collector labels as a transport failure. A read-only audit, frozen before results, establishes that the cancellation occurs at the declared deadline, no later search request is issued, the cancelled response is not consumed by the Agent, and its tools are not executed. The finalizer preserves the best eligible artifact, whose four file hashes match the original official verifier input receipt and reward output. These checks certify valid prespecified budget endpoints with rewards 0.9538 and 0.9516. Original collector reports remain unchanged; neither model calls nor verifier runs were added for this audit. All recorded usage, including the completed late responses, is counted.

The anonymous component summary binds the released run manifest, original report, independent audit, and final-result hashes. The run manifest freezes configuration, input trees, runtime and audit sources before outcomes. The reporting script regenerates the two component tables and contrasts from this summary. Earlier development results remain separate and are not pooled with these four runs or selected by reward.

L Relation to Prior Work

Skill libraries and agent workflows. Skill libraries and orchestration systems address how procedures are acquired, selected, and combined. Voyager maintains executable skills in an embodied environment (Wang et al., 2023); AgentSkillOS organizes its library as a capability tree and orchestrates selected skills through DAG-based pipelines (Li et al., 2026a); SkillRet evaluates retrieval from large skill libraries (Kang et al., 2026). SWE-Skills-Bench measures the marginal utility of injected skills through paired evaluations (Han et al., 2026). DeltaSkill contributes a subsequent adaptation step: task requirements and configuration-specific capability evidence guide shared coordination and consolidation of the selected procedures. The resulting support is coupled to candidate management and execution control.

Skill adaptation and compilation. SkillAligner jointly adapts retrieved skills to a task and its execution specification, consolidating the result into a guide used throughout the trajectory (Li et al., 2026b). SkVM measures model–harness capabilities and uses these profiles for compilation, with runtime code solidification and adaptive recompilation (Chen et al., 2026). Its runtime can retain the best-performing skill variant and roll back a regressing recompilation. DeltaSkill couples task- and receiver-conditioned residual support with retention of validated task artifacts and continuation based on public progress evidence. This combination connects instruction adaptation to the state of the current task solution.

Prompt compression and context management. LLMLingua selects prompt content under a compression budget (Jiang et al., 2023a), LongLLMLingua targets long-context information selection (Jiang et al., 2023b), and LLMLingua-2 learns task-agnostic token selection through data distillation (Pan et al., 2024). Our archived LLMLingua condition uses LLMLingua-2. Residual compilation instead makes procedural obligations, dependencies, and evidence of agent coverage

explicit; an unresolved obligation can require retaining a long instruction. Runtime context management is a related but distinct operation: MemGPT organizes information across memory tiers (Packer et al., 2023), whereas the compiler decides which procedural content to supply for a task.

Feedback-guided execution and evaluation. ReAct interleaves reasoning with environment interaction (Yao et al., 2022), and Reflexion uses linguistic feedback and episodic memory to guide subsequent behavior (Shinn et al., 2023). DeltaSkill connects such visible execution evidence to candidate validity, retained-artifact identity, and continuation decisions. AgentBench, WebArena, and SWE-bench evaluate interactive agents and software outcomes (Liu et al., 2023; Zhou et al., 2023; Jimenez et al., 2023); SEA-Eval examines success and token consumption across sequential tasks (Jiang et al., 2026). These perspectives motivate examining both delivered artifacts and execution resources, with our task interfaces and measurement rules specified in Section 4.1.

M Detailed Process Diagrams

The detailed views below expand the two-component presentation in Figures 1, 3, and 4. They make configuration binding, instruction transformations, candidate validation, and state transitions explicit. The four views unpack the same method: profiling supplies compilation, while validation and retention supply execution control.

M.1 End-to-End Data and Control Flow

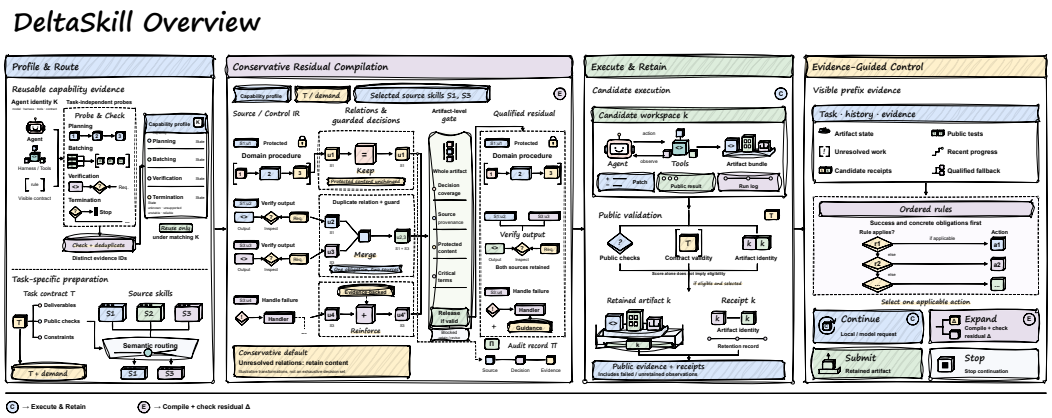


Figure 18: Detailed expansion of Figure 1 in the original four-block horizontal layout. The left two blocks build configuration-bound capability evidence, route skills, and compile residual instructions; the right two blocks execute candidates, validate and retain artifacts, and govern continuation. Public checks bind retained artifacts to receipts. The control block depicts the standalone reference controller's ordered action interface: C returns to execution and E requests qualified compilation. Appendix A.4 specifies the separate success-first runtime used by the experimental pipeline.

M.2 Capability Evidence and Compilation Decisions

Figure 19 expands profile acquisition; Figure 20 follows selected source instructions through compilation. Together they detail the evidence and transformations summarized in Figure 3.

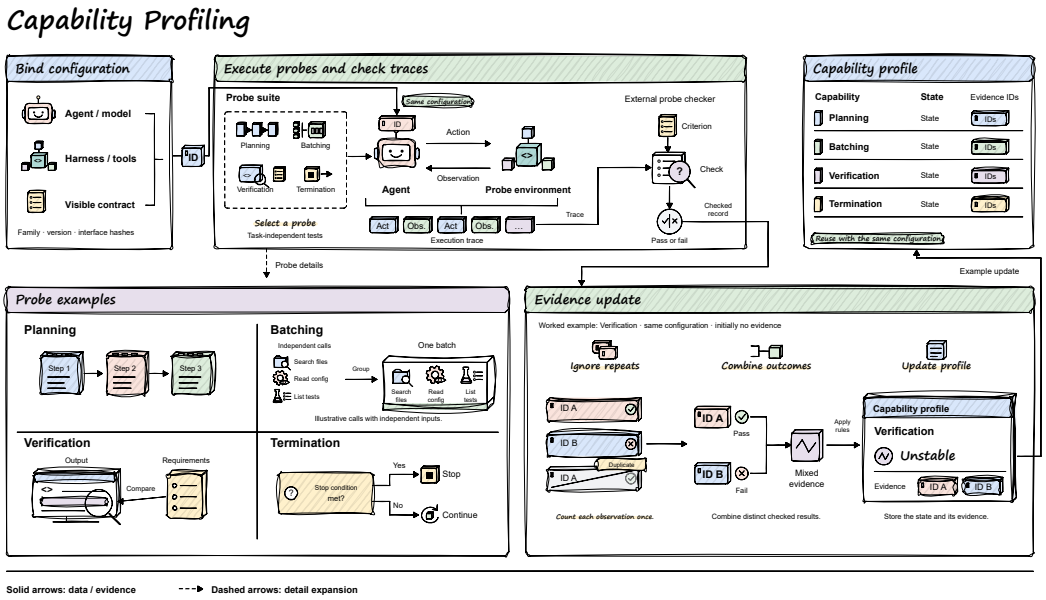


Figure 19: Configuration-bound capability profiling. The adapter executes probe variants, an external checker evaluates their traces, and distinct evidence identifiers update the profile. The lower panels illustrate probe behaviors and a worked update: one pass and one failure give mixed evidence, while replaying the pass adds no observation. The shown capabilities are examples from the catalog in Appendix A.2.

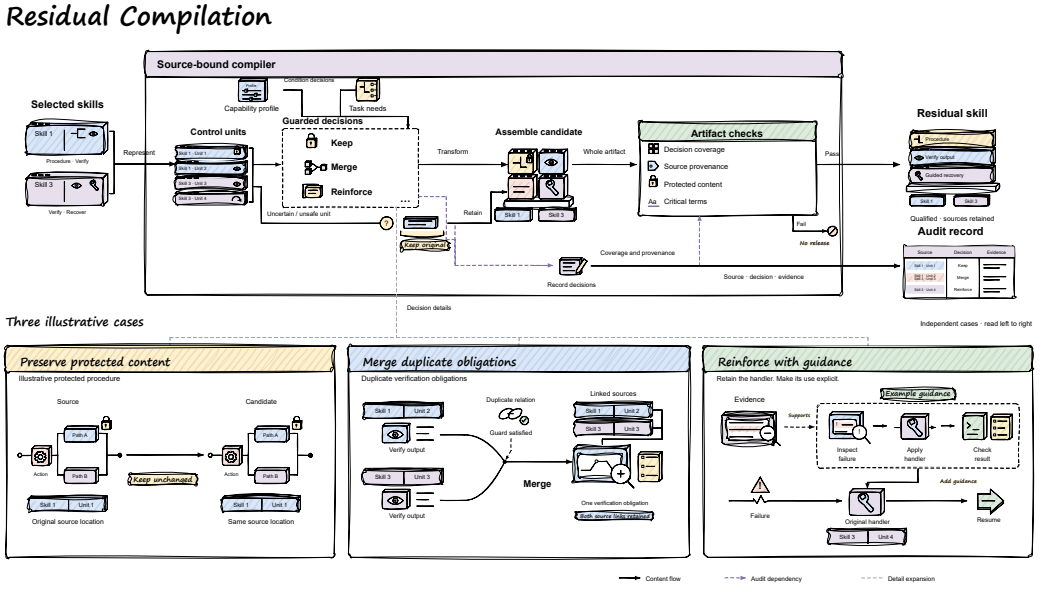


Figure 20: Source-linked residual compilation. Control units retain provenance through decisions, assembly, and release checks. The lower panels show independent Keep, Merge, and Reinforce examples. Added reinforcement guidance illustrates the action interface; the archived active-compilation decisions retain Reinforce text unchanged (Appendix G.1). Appendix G documents the implemented checks and their regression analysis.

M.3 Candidate Retention and Controller State

Figure 21 expands the candidate, receipt, and state interfaces summarized in Figure 4. Candidate observations and retained-artifact records follow separate paths: an unsuccessful revision can change the evidence for continuation without replacing the retained output. Submission uses the artifact identity bound to the retention receipt.

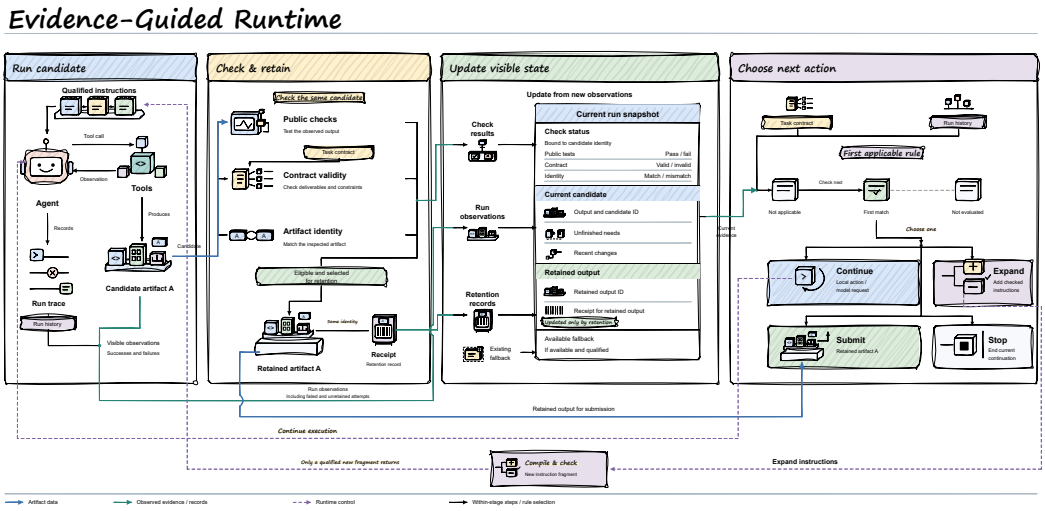


Figure 21: Detailed execution and state flow with the standalone reference controller's ordered action interface. Artifact data, public observations, and control transitions have distinct paths; expansion returns only qualified compiled support. The experimental pipeline binds the separate success-first runtime and its stopping conditions described in Appendix A.4.