Task-Aware Self-Evolution for Resource-Constrained AI Appliances

Anonymous authors
Paper under double-blind review

Abstract

How should a resource-constrained AI appliance learn to serve its tasks better? For multi-step agents, the work to be served is itself an outcome of execution: model responses trigger tools, change context, and determine subsequent requests. Optimizing isolated inference requests therefore leaves a gap between system performance and useful task delivery. We present a task-aware self-evolution framework that addresses this gap through a persistent cycle of execution, outcome modeling, and resource decisions. The framework couples a conditional three-level action space—instance structure, inference configuration, and serving policy—with task-level probabilistic feedback and cost-aware experiment selection. A cohort model separates task heterogeneity, configuration effects, and batch variation, allowing incomplete executions to inform later decisions. We instantiate the framework on a three-node AI appliance and study 54 experimental attempts on 50 Office tasks. The study reveals quality–time trade-offs, substantial queueing exposure, and changing execution work even under the same nominal configuration. Together, the system design and empirical findings establish a task-centered approach to adaptation: learn from what the appliance delivers, reason about what another experiment would teach, and adjust how resources serve the next task cohort.

1 Introduction

An AI appliance brings models, inference infrastructure, and execution tools onto a fixed resource budget. Its value is ultimately realized in completed work: a reconciled spreadsheet, a correct document, or a successful workflow. A faster inference engine is useful insofar as it helps deliver that work. Yet the decisions that determine delivery—how to organize model instances, configure inference, and assign requests—are usually expressed in system-level terms, while success is judged at the task level.

This gap becomes pronounced for multi-step agents. A single task can alternate between generation and tools, accumulate long context, revise earlier outputs, and repeatedly verify an artifact. The assigned task is known in advance, but the sequence of requests needed to finish it is not. *Execution changes the workload that the system must subsequently serve.* As a result, fewer model calls may reflect either an efficient solution or premature termination; high throughput may sustain either productive work or repeated correction. Neither quantity, by itself, determines whether the user received a better result.

Application-aware serving systems already exploit structure across requests. SGLang exposes structured language-model programs, Parrot uses application dataflow, and Autellix schedules agents as programs (Zheng et al., 2024; Lin et al., 2024; Luo et al., 2025). Configuration optimization addresses another part of the problem: selecting efficient operating points from a large system space. The question we pursue connects these two perspectives: *how can an appliance use the outcomes of executing tasks to learn better deployment and serving decisions, while accounting for the cost of learning?*

We call this process *task-aware self-evolution*. What evolves is the system's knowledge and its resource decisions; model weights need not change. An execution supplies evidence about task difficulty, configuration suitability, delivery risk, and resource expenditure. The learner uses that evidence to update its predictions, select an informative next experiment, and recommend a configura-

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.

tion for future execution. The practical objective has two parts: improve useful task delivery, and make the process of finding improvements worthwhile.

Our central design choice is to make the *task cohort the unit of predicted value* and the *deployment batch the unit of configuration evidence*. This supports three complementary mechanisms. A conditional action space connects instance structure, inference settings, and serving choices without proposing infeasible combinations. A hierarchical outcome model distinguishes persistent configuration effects from task heterogeneity and batch variation, retaining partial observations rather than waiting for an all-or-nothing batch score. A cost-aware decision rule asks how an additional experiment could change the next recommendation. These mechanisms form an outer learning loop around fast request-level serving and task progress control.

Our empirical study makes the motivation concrete. Across 54 attempts on a fixed 50-task Office cohort, quality and completion time form distinct trade-offs. Two executions of the same nominal candidate produce 1,175 and 1,043 model calls, with faster completion accompanying lower aggregate quality. In two mixed-serving batches, waiting accounts for about 70% of completed-request time exposure. These observations suggest that useful adaptation requires both task outcomes and an explanation of the work and waiting that produced them.

We make three contributions. First, we formulate task-aware self-evolution as a feedback loop over a conditional deployment–inference–serving space, with separate delivery and exploration objectives. Second, we build a system that connects partial task observations, cohort prediction, cost-aware selection, and physical execution. Third, we provide an empirical study of task delivery, endogenous execution work, and resource bottlenecks that grounds the design. The contribution is the task-centered organization of this loop and its operational realization; established serving and Bayesian decision primitives provide its building blocks.

2 Task-Aware Self-Evolution

2.1 From request performance to task delivery

Let B denote the available hardware budget and $\mathcal{T}$ the assigned task cohort. Each task has a family $g(i)$ and an execution history of model calls, tool actions, and artifact updates. Decision-time task information may include family, input size, prior observations, and current progress. Future calls, final output lengths, and final scores are outcomes to predict rather than features available in advance.

For a cohort with families $\mathcal{F}$, we use fixed weights $w_i = 1/(|\mathcal{F}|n_{g(i)})$ so that each family contributes equally. Let $q_i \in [0,1]$ be terminal artifact quality, T_i^{del} source-to-valid-delivery time, and $\delta_i(h)$ indicate delivery by deadline h. We evaluate three complementary outcomes:

$$\begin{aligned} C_h &= \sum_i w_i \delta_i(h) q_i, \qquad S_h = \prod_i \delta_i(h), \\ L_h &= \sum_i w_i \left[\delta_i(h) T_i^{\text{del}} + (1-\delta_i(h))h\right]. \end{aligned} \tag{1}$$

Delivered quality credit C_h measures useful output within the deadline, L_h penalizes unfinished work at the horizon, and S_h records whether the entire cohort is delivered. We also report delivery count, complete terminal quality $Q = \sum_i w_i q_i$, and, when every task is delivered, mean delivery time $M_{\text{del}} = \sum_i w_i T_i^{\text{del}}$. Keeping these quantities separate makes the quality, speed, and reliability trade-off visible.

Learning consumes resources too. For experiment e, the desired cost is attributable node-time $c(e) = \int n_e(t)\, dt$, including deployment, execution, evaluation, and recovery under a consistent accounting rule. Delivery metrics describe what an operating point offers; exploration cost describes what it takes to find that point. This distinction prevents an expensive search from appearing efficient solely because its best observed configuration is good.

Table 1: Three levels of adaptation. Each level changes a different part of the path from resources to delivered tasks.

Level	Decisions	Task-level consequence
Structure H	Replicas, parallelism, ranks, placement	Available execution paths and resource sharing.
Inference d	Concurrency, memory, prefill, cache, speculation	Service capacity and the cost of each request.
Serving ϕ	Admission, compatible pools, presets, routing	Which tasks wait, retain locality, or access a worker.

2.2 A conditional three-level action space

A system action $x = (H, d, \phi)$ specifies how resources are organized, how inference runs, and how tasks use the resulting instances:

$$\mathcal{X}_{\text{adm}}(B, \eta) = \{(H, d, \phi) : H \in \mathcal{H}(B),\ d \in \mathcal{D}_{\text{adm}}(H; \eta),\ \phi \in \Phi_{\text{adm}}(H, d; \eta)\}. \tag{2}$$

The execution conditions η include model and backend compatibility. Conditionality is essential: a two-node pipeline is one logical worker, its inference settings differ from those of independent replicas, and routing can only choose among compatible deployed workers. The three levels describe one coupled decision space.

2.3 A slow learning loop around fast serving

Figure 1 shows the system organization. The outer loop maintains execution evidence, predicts cohort outcomes, selects an experiment, and updates the deployment at a batch boundary. The inner loop admits requests, routes them to compatible workers, and monitors task progress. A thin ingress identifies the model pool, worker class, and request preset; native routing selects the worker, whose capacity gate controls admission. This separation lets serving react quickly while configuration changes use accumulated evidence.

Task awareness appears at multiple points: the objective values task artifacts and delivery, the predictor represents differences among tasks and families, and execution retains task identity and progress across requests. This does not require a semantic classifier as a prerequisite. In our instantiation, the learner adapts to a fixed cohort, while the serving layer uses compatible request classes and native locality information.

3 Learning from Execution Experience

3.1 An observation model for unfinished work

An expensive batch rarely produces a single, immediately reliable label. Some tasks finish early, others remain active at a deadline, and a failed execution may still leave a scorable artifact. Our evidence representation therefore retains task-level delivery, failure, quality, and timing rather than reducing every batch to one success flag. Administrative censoring contributes the time observed at risk; it is not recorded as completion or as a zero-quality task.

Observations from the same deployment share execution conditions. We model that dependence explicitly: 50 task outcomes enrich knowledge of the cohort, but they are not 50 independent repetitions of a configuration. Similarly, a later observation extends an earlier deadline view of the same task; the common prefix enters the likelihood once. Compatibility is target-specific, allowing reusable evidence to persist while changes in timing or execution conditions remain distinguishable.

3.2 Predicting the next cohort

For batch b, task i, and outcome head k, the predictor combines configuration effects with task and batch structure:

$$\ell_{bi}^{(k)}(x) = f_{\text{shared}}^{(k)}(x) + f_{\text{domain(b)}}^{(k)}(x) + a_{g(i)}^{(k)} + t_i^{(k)} + u_b^{(k)}. \tag{3}$$

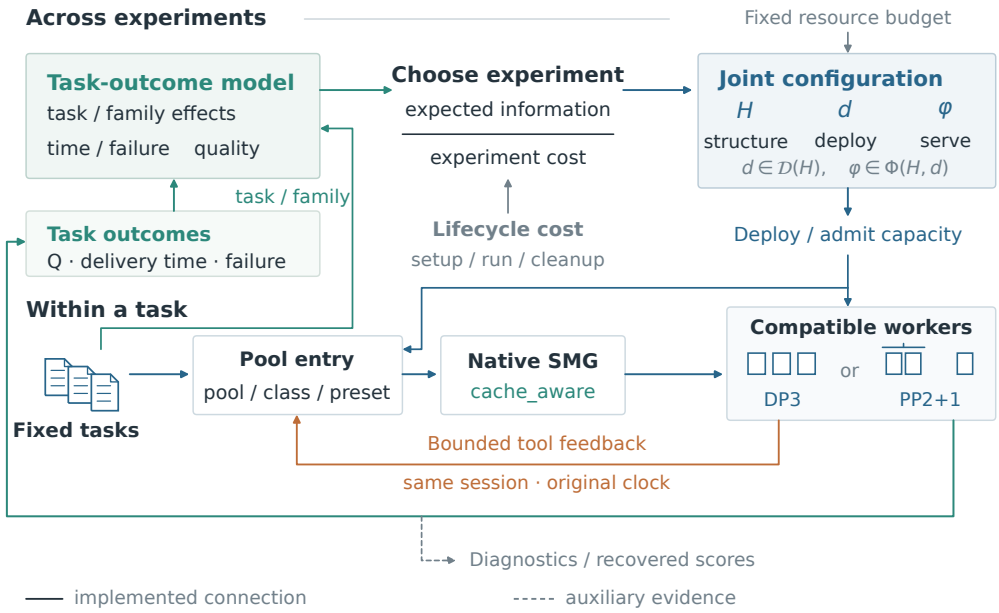

Figure 1: From execution experience to resource decisions. Task outcomes update the outer learner; its selected deployment defines the conditions under which the inner serving loop operates. Auxiliary predictors expose execution behavior alongside the active cohort decision path. SMG is the native model gateway; DP3 denotes three replicas and PP2+1 a two-node pipeline plus one replica.

The shared component transfers configuration information; the domain component accounts for compatible execution regimes; family and task offsets represent heterogeneous difficulty; and $u_b^{(k)}$ captures a common batch disturbance. Predicting a new batch integrates its disturbance instead of treating a favorable realization as a permanent configuration advantage.

Timing uses piecewise competing risks for delivery and failure, allowing incomplete executions to inform the posterior through survival exposure. Quality uses a bounded score model on delivered tasks; recovered artifact scores remain available for descriptive analysis. The implemented quality model clips a latent Gaussian score to $[0, 1]$, preserving probability at both endpoints. Appendix D specifies the likelihoods, Gaussian priors, and Laplace approximation.

Four configuration models—a global GP, a regularized linear model, a topology-local GP, and an RF-leaf similarity model—share this observation structure. They differ in how configurations generalize to one another, rather than in what counts as a task outcome. Forecasts are issued before deployment and scored against subsequent observations; GP initially controls selection, with challenger control conditioned on prospective predictive evidence. The present task/family terms model offsets, and timing and quality use separate coefficients. This provides a tractable fixed-cohort model without assuming that arbitrary semantic task–configuration interactions are already learned.

3.3 Choosing what to learn next

The recommendation rule translates outcome predictions into an explicit operating preference. Our current instantiation favors reliable delivery of the full cohort:

$$U_h(x) = \mathbb{E}\left[\mathbf{1}\{S_h = 1\}\frac{Q}{1 + M_{\text{del}}/h}\,\middle|\, x, \mathcal{D}\right]. \tag{4}$$

Here $\mathcal{D}$ is the accumulated evidence. Quality and time matter within the complete-delivery event, while incomplete batches receive zero utility. This is a full-cohort reliability preference; applications that value partial delivery can instead formulate preferences over C_h and L_h. We report these component metrics independently so the operating preference remains visible.

The next experiment need not be the current best recommendation. Its purpose may be to resolve uncertainty that changes a later decision. With a fixed recommendation set $\mathcal{R}$, define

$$V(\mathcal{D}) = \max_{x \in \mathcal{R}} U_h(x; \mathcal{D}),$$
$$I(e) = \mathbb{E}_{O_e|\mathcal{D}}[V(\mathcal{D} \oplus O_e)] - V(\mathcal{D}), \qquad e^* = \arg\max_e \frac{I(e)}{\mathbb{E}[c(e) \mid \mathcal{D}]}. \tag{5}$$

This knowledge-gradient-style rule values an observation by its effect on the recommendation (Frazier et al., 2009). It gives costly deployment experiments a common currency: expected decision improvement per resource-time. Hypothetical observations use the same task identities and observation contract as actual execution, and conditional posterior updates account for what each observation would reveal. Finite numerical integration makes the calculation practical. The rule is a one-step cost-aware policy, not a global finite-budget optimality claim.

3.4 Closing the physical loop

The loop proceeds through five operations: *observe* task outcomes and costs; *update* compatible outcome models; *select* an informative admissible experiment; *execute* its deployment and serving contract; and *incorporate* the resulting evidence into the next decision. The evidence prefix, forecast, and chosen candidate are bound before deployment, making it possible to examine how new information changes behavior.

Execution controls support this learning process. Capacity checks prevent unsupported layouts from reaching task intake. Bounded progress feedback distinguishes a reported tool operation from a verified artifact change and limits intervention in unproductive paths. Continuation uses the task's original clock and remaining budget. Unsuccessful attempts and recovery costs return to the evidence record, so the learner sees the consequences of a choice rather than only its successful outputs. Initialization is study-wide; changing an execution protocol does not restart the exploration allowance.

4 Empirical Study

4.1 Platform, workload, and study design

We use three DGX Spark nodes to execute 50 WorkBuddy Office tasks through the CodeBuddy Code harness. The cohort contains 17 document-operation, nine automation, and 24 data-processing tasks, scored by their native normalized rubrics. We keep the 27B model weights fixed and use mixed NVFP4/FP8 quantization with a BF16 output head. The physical instantiation supports three independent replicas or a two-node pipeline plus one replica. Its native router uses `cache_aware`; the three-level framework is thus instantiated with a restricted serving-policy slice. Appendix C lists the search ranges and routing contract.

The empirical corpus contains 54 attempts from system development, with 1,980 task-score observations, including 246 recovered from retained artifacts. It covers both complete and partial executions. The corpus precedes the final integrated cohort-controller release: it supports the following characterization of task delivery and the design requirements for adaptation. Final-controller comparative learning gains are outside this study. We retain all 54 attempts in Appendix E, with analysis-specific inclusion rules rather than a single filtered success set.

We use terminal quality for artifact analysis and valid-delivery time for delivery analysis. Earlier records support solver/artifact seal time M_{seal}, a distinct endpoint before verification. Fourteen earlier complete batches provide comparable measurement definitions for the seal-time panel; four later complete deliveries provide the delivery-time panel. Across all 12 post-repair batches, a two-hour deadline and a three-hour full-path limit support partial-delivery analysis. Software and protection conditions are retained when interpreting comparisons.

4.2 Task value has more than one performance axis

Figure 2 shows the quality–time trade-off. Among the 14 earlier complete batches, four observations are nondominated: a faster operating point provides $Q = .8021$ at 30.43 minutes, while a

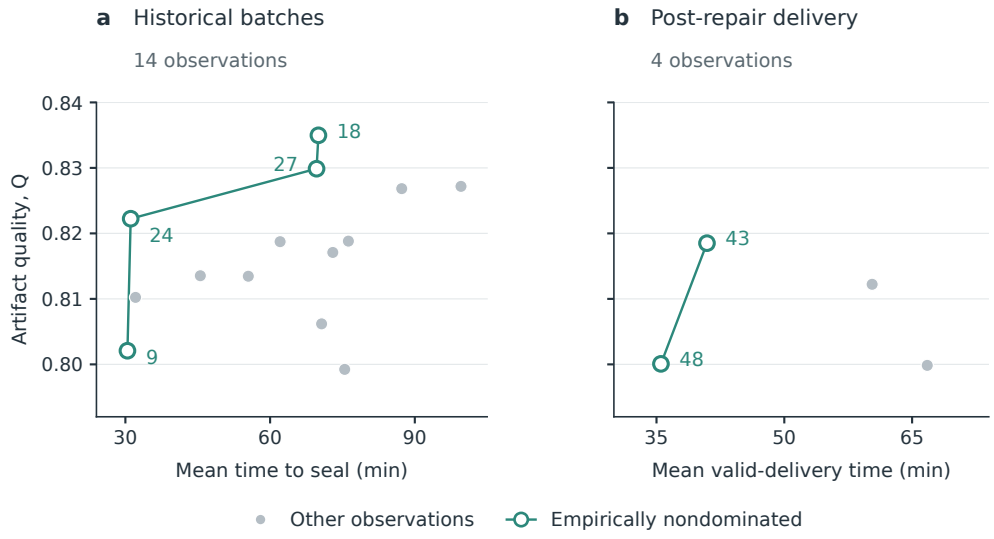

Figure 2: Task quality and completion time offer complementary views of appliance performance. Each panel preserves a common measurement endpoint; highlighted observations are nondominated within that panel. Lines guide the eye. These empirical trade-offs motivate outcome-aware selection rather than optimizing response speed alone.

higher-quality point reaches $Q = .8350$ at 70.02 minutes. The later delivery panel likewise has two nondominated points: $Q = .8185$ at 40.92 minutes and $Q = .8001$ at 35.52 minutes. A single speed ranking would conceal these differences in delivered quality.

Completion also differs from artifact quality. Two later batches have scores for every task after artifact recovery, but only 49 valid deliveries at the deadline. Another delivers 40 tasks, with delivered quality credit $C_h = .6470$ and deadline-penalized time $L_h = 85.24$ minutes. Such outcomes matter to an appliance user and to its learner: unfinished work supplies information about delivery risk even when an eventual score is available. The cohort representation preserves this distinction without discarding the useful observations.

4.3 The task trajectory changes the work to be served

Figure 3 compares two executions of the same nominal candidate. Their model-call counts are 1,175 and 1,043; input tokens are 57.94M and 49.86M, and output tokens are .644M and .550M. The second has 13.21% lower mean delivery time and 1.84 percentage points lower aggregate quality. Across tasks, 20 scores increase, 24 decrease, and six stay unchanged. The batches differ in code revision, so the comparison describes execution variability rather than isolating a configuration effect.

The implication is that the work required by an agent task cannot generally be treated as a fixed numerator divided by inference throughput. Subsequent requests depend on earlier responses and tool outcomes. A task-aware learner must therefore connect resource decisions to the joint outcome of execution—work generated, time spent, and artifact delivered. This motivates predicting task outcomes directly while using request traces to explain their variation.

4.4 Waiting and capacity connect the three action levels

Task completion depends on access to service as well as service speed. For each request, let W be arrival-to-dispatch waiting and S dispatch-to-terminal duration. In two mixed-serving batches, $\sum W / \sum (W + S)$ is 70.67% and 69.79% over completed requests. Including all timed terminal records gives 69.68% and 68.04%, respectively. These request-time exposures show that substantial delay can accumulate before dispatch; they are not GPU idle fractions or fractions of batch wall time.

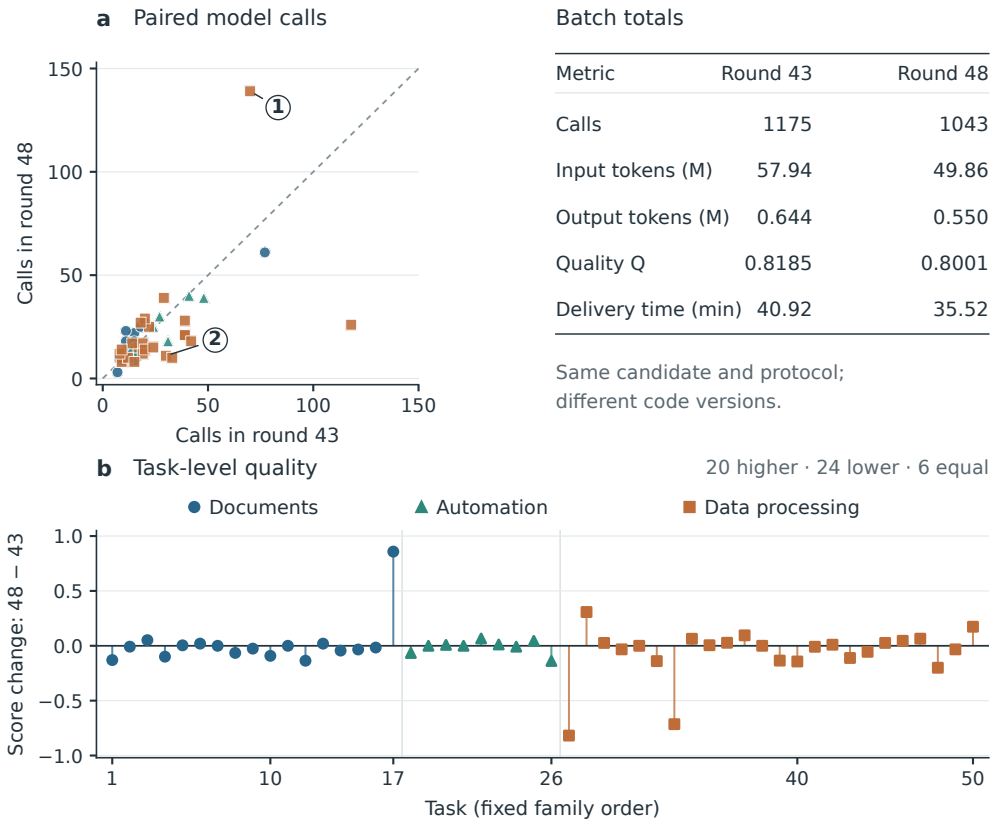

Figure 3: Task trajectories reshape the serving workload. Paired task calls and scores vary across two executions of the same nominal candidate. Faster aggregate completion accompanies lower quality; the batches also differ in code revision. The callouts show a contract-extraction task (1) and a health-quote task (2).

Configured concurrency alone does not determine effective capacity. One three-worker deployment assigns 16 slots per worker, while startup token pools are approximately .686M, .718M, and .719M. Sixteen independent requests at the allowed 100,000-input plus 16,384-output limit would require 1.862M tokens. This worst-case comparison illustrates a structural mismatch between slot counts and token residency; actual pressure also depends on realized lengths, state sharing, and cache behavior.

These observations explain why the action levels are coupled. Instance layout determines the worker pool; inference configuration determines each worker's effective capacity; serving decisions distribute the realized task demand across that capacity. Locality can reduce repeated work while concentrating queues, and admission can protect capacity while increasing waiting. The evidence motivates evaluating these choices by their task consequences rather than assuming that any individual system metric is sufficient.

4.5 What execution experience contributes

The study identifies complementary learning signals: artifact scores describe delivered value, partial timing constrains delivery risk, repeated executions expose workload variability, and request traces reveal where delay accumulates. Historical training prefixes show that earlier outcomes were reused in later decisions. The integrated controller turns this evidence flow into an explicit prediction-and-selection procedure, separating the recommendation from the experiment chosen to improve it.

Costs make the distinction consequential. All 54 attempts have recorded wall cost, but allocation and planning detail vary across the historical record. A uniform reserved-three-node proxy totals 248.25

node-hours, including unsuccessful attempts. This quantifies a substantial experimental investment; it is not an algorithmic learning-efficiency curve. A comparison of learning policies must evaluate independently tested recommendations at matched cumulative cost, including initialization and failures. The principal next empirical question is how much the structured feedback reduces the cost of finding reliable, high-value operating points.

5 Related Work

Application- and agent-aware serving. Serving systems increasingly exploit the structure of applications spanning multiple model calls. SGLang provides programming primitives and runtime optimizations for structured language-model programs, including prefix reuse (Zheng et al., 2024). Parrot exposes application dataflow through semantic variables to improve end-to-end execution (Lin et al., 2024). Autellix incorporates program context into scheduling and prioritizes calls using the execution history of their parent programs (Luo et al., 2025). These systems establish the value of looking beyond isolated requests. Our work connects this perspective to adaptation across deployments: completed tasks provide evidence about the quality, timeliness, and reliability of delivery, which guides subsequent resource and serving decisions.

Deployment and inference configuration. AlpaServe studies placement and model parallelism under bursty demand (Li et al., 2023). Vidur combines profiling and simulation to explore inference configurations spanning parallelism, batching, and scheduling (Agrawal et al., 2024). SCOOT uses Bayesian optimization to tune inference engines for service-level objectives, incorporating known constraints and learned feasibility (Cheng et al., 2025). These approaches make configuration search and its evaluation cost central systems concerns. We study the additional difficulty posed by agent execution: future requests emerge through model–tool interactions, and configuration choices must ultimately be assessed through task artifacts and valid completion. Our conditional action space connects these outcomes to instance structure, inference settings, and serving contracts.

Learning from costly and partial observations. Algorithm configuration already addresses heterogeneous problem instances and mixed parameter spaces. SMAC models configuration performance across instances (Hutter et al., 2011b; Lindauer et al., 2022), while censored-response Bayesian optimization retains information from interrupted evaluations (Hutter et al., 2011a). Cost-aware Bayesian optimization accounts for variable evaluation duration (Snoek et al., 2012); knowledge gradient evaluates measurements through their influence on subsequent recommendations (Frazier et al., 2009). Budgeted optimization further distinguishes such local criteria from planning over a finite remaining budget (Astudillo et al., 2021). Building on this foundation, our formulation preserves task-level delivery, failure, and quality observations within a shared deployment batch. It separates persistent configuration effects from batch variation and evaluates experiments by their expected contribution to future task delivery relative to their cost.

Self-adaptation and the object of improvement. Experience-driven configuration has a substantial systems lineage: OtterTune reuses prior tuning experience to recommend database settings (Van Aken et al., 2017). At the agent level, Reflexion improves decisions through feedback and memory, while automated agent design searches programs and workflows (Shinn et al., 2023; Hu et al., 2025). Our object of adaptation is the appliance's deployment and serving behavior. With the underlying task program and model held fixed, the research question is how task outcomes can teach the system to use its resources more effectively. The resulting task-delivery–evidence–decision loop links systems execution with sequential learning.

6 Discussion and Conclusion

Task-aware self-evolution treats an AI appliance as a system that can learn from the work it performs. Our framework connects conditional resource choices, partial task-outcome modeling, and cost-aware experimentation in one persistent loop. The empirical study shows why these components belong together: speed and quality trade off, agent trajectories change the workload, and waiting links task demand to effective capacity. These findings support a broader design principle: system adaptation should be organized around delivered task value and the evidence needed to improve it.

The present study concerns a fixed Office cohort and observational operating points. It does not establish held-out task generalization, final-controller superiority over matched search baselines, or the independent benefit of every component. The current native-routing slice and approximate outcome model bound that evaluation. These limits identify focused extensions—prospective learning curves, task-structure and cost ablations, and matched policy interactions—within the same research direction. The central system objective remains to turn accumulated execution experience into progressively better resource decisions for useful task delivery.

AI Use Statement

Generative AI assisted with conceptual organization, mathematical formulation and review, implementation and analysis code, interpretation of experimental records, manuscript drafting, translation, and figure preparation. Reported quantitative observations are computed from preserved experimental records; synthetic AI-generated experiment outcomes are not used. The authors are responsible for the final claims, references, and artifacts.

Ethics Statement

The evaluation concerns benchmark Office workflows. Tool actions can modify files or duplicate side effects, motivating bounded recovery and preserved provenance. Trace and artifact release requires review of content and permissions. Task rubric scores do not establish general autonomous-agent safety or comprehensive deployment fairness.

Reproducibility Statement

Appendices specify measurement boundaries, search support, evidence eligibility, and complete historical results. The author-side source bundle retains bilingual LaTeX, vector figures, frozen analysis inputs, recomputation code, and provenance hashes. It is distinct from a public anonymous release of raw task artifacts or runtime code.

References

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for LLM inference. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/b74a8de47d2b3c928360e0a011f48351-Abstract-Conference.html.

Raul Astudillo, Daniel R. Jiang, Maximilian Balandat, Eytan Bakshy, and Peter I. Frazier. Multi-step budgeted Bayesian optimization with unknown evaluation costs. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a8ecbabae151abacba7dbde04f761c37-Abstract.html.

Ke Cheng, Zhi Wang, Wen Hu, Tiannuo Yang, Jianguo Li, and Sheng Zhang. SCOOT: SLO-oriented performance tuning for LLM inference engines. In *Proceedings of the ACM Web Conference 2025*, 2025. URL https://arxiv.org/abs/2408.04323v2.

Peter I. Frazier, Warren B. Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613, 2009. doi: 10.1287/ijoc.1080.0314. URL https://people.orie.cornell.edu/pfrazier/pub/CorrelatedKG.pdf.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/36b7acf6f6010652b3f2a433774a66fe-Abstract-Conference.html.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Bayesian optimization with censored response data. In *NIPS Workshop on Bayesian Optimization, Experimental Design, and Bandits*, 2011a. URL https://ml.informatik.uni-freiburg.de/wp-content/uploads/papers/11-NIPS-workshop-BO-with-censoring.pdf.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, pp. 507–523, 2011b. URL https://www.cs.ubc.ca/labs/algorithms/Projects/SMAC/papers/11-LION5-SMAC.pdf.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679. USENIX Association, 2023. URL https://www.usenix.org/conference/osdi23/presentation/li-zhouhan.

Chaofan Lin, Zhenhua Han, Chengruidong Zhang, Yuqing Yang, Fan Yang, Chen Chen, and Lili Qiu. Parrot: Efficient serving of LLM-based applications with semantic variable. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 929–945. USENIX Association, 2024. URL https://www.usenix.org/conference/osdi24/presentation/lin-chaofan.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile Bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54): 1–9, 2022. URL https://www.jmlr.org/papers/v23/21-0888.html.

Michael Luo, Xiaoxiang Shi, Colin Cai, Tianjun Zhang, Justin Wong, Yichuan Wang, Chi Wang, Yanping Huang, Zhifeng Chen, Joseph E. Gonzalez, and Ion Stoica. Autellix: An efficient serving engine for LLM agents as general programs. *arXiv preprint arXiv:2502.13965*, 2025. URL https://arxiv.org/abs/2502.13965.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012. URL https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pp. 1009–1024, 2017. doi: 10.1145/3035918.3064029. URL https://db.cs.cmu.edu/papers/2017/tuning-sigmod2017.pdf.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

Table 2: Configurations of selected observations. NT denotes non-thinking generation; T denotes thinking. Capacities are per logical worker. These descriptors identify the plotted measurements rather than establish matched experimental treatments.

ID	Instance layout	Capacities	Generation	Routing
9	Three replicas	16/16/16	NT, $T_{\text{samp}} = 0$	Load-only
24	Three replicas	16/16/16	NT, $T_{\text{samp}} = 0$	Load-only
27	PP2 + replica	15/8	NT, $T_{\text{samp}} = 0$	Load-only
18	PP2 + replica	16/16	NT, $T_{\text{samp}} = 0$	Load-only
43	Three replicas	16/16/16	NT, $T_{\text{samp}} = .7$	Load-only
48	Three replicas	16/16/16	NT, $T_{\text{samp}} = .7$	Load-only
45	Three replicas	4/4/4	NT/NT/T	Task affinity
52	Three replicas	4/4/4	NT/NT/T	Load-only

A Measurement and Observation Contract

Task identity and aggregation. The requested cohort always contains 50 task identities. Document operations has 17 tasks, automation 9, and data processing 24. Scores lie in $[0, 1]$ and retain their native rubric meaning. Family-macro weights are fixed before execution; partial results do not change denominators. For observed-score set $\mathcal{O}$,

$$Q_{\text{lo}} = \sum_{i \in \mathcal{O}} w_i q_i, \qquad Q_{\text{hi}} = Q_{\text{lo}} + \sum_{i \notin \mathcal{O}} w_i. \tag{6}$$

These are identification bounds under bounded missing scores, not confidence intervals. An observed zero is different from a missing score. Recovery of a score does not establish that the corresponding task completed successfully or that the score was available at an earlier deadline.

Time boundaries. Seal-time reconstruction subtracts source submission from each task's solver/artifact seal on matching monotonic clocks and boot identities. The seal is the verification-start boundary after artifact collection. We aggregate individual times using the fixed family weights. Batch makespan and agent time after subtracting request waiting are different quantities and cannot fill a missing M_{seal}. Valid-delivery time additionally requires the delivery contract. A terminal seal event alone does not establish complete valid delivery.

Deadline and final observations. The post-repair horizon is two hours, with a three-hour full-path bound in the current contract. A task can contribute deadline credit, later provide a terminal artifact score, or remain censored. Later observations extend the existing history; they do not create an independent copy of the prefix. The controller distinguishes delivery, failure, and censored timing evidence. Tables preserve unknown values as dashes.

Request exposure. Let a_r, d_r, e_r be compatible arrival, dispatch, and terminal timestamps. Waiting is $W_r = d_r - a_r$ and observed post-dispatch duration $S_r = e_r - d_r$. We report both completed and all-timed terminal populations, retaining missing timing as missing. Since concurrent request intervals overlap across tasks, summing them gives exposure rather than wall time. Even serial task paths can include admission, tools, approval, and artifact work not captured by these request intervals. An unmeasured residual must not be labeled entirely as tool time.

B Experimental Configuration and Scoring

All listed observations use CodeBuddy Code 2.109.3 and the local checkpoint served as `Qwen3.8-27B-NVFP4-BF16-LMHead`, with FP8 E4M3 KV cache. The checkpoint combines NVFP4 and FP8 quantization with a BF16 LM head. The pinned SGLang build reports revision `78da625`; backend context capacity is 131,072 tokens, with a 16,384-token generation limit. Here T_{samp} denotes sampling temperature, not task completion time. The NT preset in observations 43/48/45/52 uses top-$p = .8$, top-$k = 20$, and presence penalty 1.5; the thinking worker in 45/52 uses temperature 1, top-$p = .95$, top-$k = 20$, and presence penalty 0. Software revisions and other

execution conditions differ across some observations; configuration labels alone do not imply exchangeability.

Task scoring. The fixed cohort contains 17 document-operation, nine automation, and 24 data/file-operation tasks. Native task-specific verifiers evaluate generated artifacts and task states; the LLM judge is disabled. Scores are normalized by each task's native maximum and averaged first within families, then equally across the three families. For example, the health-quote task checks spreadsheet validity, module and price correctness, complete details, and summary consistency. These scores measure satisfaction of the specified rule-based checks.

C Executable Search and Decision Details

Physical support. The final release admits three independent 27B replicas and a two-node pipeline with one independent replica. A pipeline is one logical worker with multiple ranks. The ordinary candidate generator uses the ranges in Table 3. Physical-layout registration can express concurrency 1–16 but requires separate exact admission; this does not expand every ordinary branch to that range. Model availability belongs to deployment, while request selection among existing compatible classes/presets belongs to serving. A mode cannot be rewarded twice as two independent axes.

Table 3: Ordinary 27B deployment axes. Legal candidates still undergo binding and physical-admission checks.

Parameter	Values	Condition
Logical concurrency	8–16, step 1	Per worker
Static memory fraction	.60–.80, step .01	Not actual token capacity
Chunked prefill	1024–8192, step 256	Per instance
Scheduling conservativeness	.8–1.3, step .05	Backend setting
Draft tokens	2–12, step 1	Non-pipeline branch
Draft window	256, 512, 1024, 2048	Non-pipeline branch
Mamba full-memory ratio	.6–3.0, step .1	Backend-specific
Mamba cache strategy	Extra buffer / lazy	Pipeline fixes extra buffer
Decode Graph bound	Derived from concurrency	Not an independent axis

Serving contract. The native router uses version 0.3.2 with `cache_aware`, cache threshold 0.5, absolute and relative balance thresholds 4 and 1.5, eviction interval 120 seconds, and maximum tree size 67,108,864. Class admission derives from effective worker capacities; it is not an independently searched K. Worker attribution uses actual routing evidence rather than inferred selection. The available cache signal is approximate prefix affinity, not measured resident-KV hits. The thin ingress has no separate request-total limit, while native routing retains a 10,920-second timeout. Hence removal of the older truncation rule does not mean the entire stack is timeout-free.

Model and control semantics. The common design contains configuration basis functions, domain-specific configuration effects, family/task offsets, and a shared batch term. Timing represents delivery and failure risks across observation intervals; quality uses its own coefficients and score likelihood. Fresh-batch recommendation integrates batch nuisance. Model alternatives use global, regularized linear, topology-local, or RF-leaf similarity structure. Local GP requires adequate topology-specific batches and otherwise falls back globally. Control begins with GP; model competition uses forecasts issued before deployment. Descriptive request and artifact models are recorded separately and do not automatically gain control through better in-sample fit.

Experiment loop. At each boundary the controller (1) publishes the fixed-cohort evidence index, (2) proposes admissible candidates, including eligible historical configurations as seeds, (3) assembles target-compatible observations, (4) fits and compares outcome models, (5) evaluates conditional information value per expected node-second, (6) signs the forecast and executes the selected deployment, and (7) seals incremental outcomes and updates evidence. The recommendation and hypothetical observation sets remain consistent during a comparison. Numerical integration and rank

refinement are bounded; no calibration or exact-optimality certificate is inferred from completion of this calculation.

Progress protection. At most two feedback interventions are allowed by the bounded progress mechanism. A tool's claim that it modified a file is not equivalent to verified artifact change. Repeated reads alone do not justify stopping a task. Same-session continuation is owned, bounded by a checkpoint/child-exit barrier, and uses the remaining turn budget and original source clock. Failed continuation terminates and is recorded. These controls protect the execution contract; their independent causal benefit has not been measured by the present observational comparisons.

D Probabilistic Model and Experiment Selection

Task outcomes and observation horizons. An experiment selects a configuration x and observation horizon $h \in \{H, G\}$, where $H = 7200$ and $G = 10800$ seconds. A task ends in valid delivery D, protocol-terminal failure F, or administrative censoring. We use piecewise-constant cause-specific hazards on $(0, H]$ and $(H, G]$. For task i, interval j, and cause k, let $\lambda_{ijk} = \exp(z_i^\top \beta_{jk})$, where z_i encodes configuration, domain, family, task, and batch. Its survival function is

$$\begin{aligned} \mathcal{S}_i(t) &= \exp\left[-\sum_{j,k} E_{ij}(t)\lambda_{ijk}\right], \\ E_{ij}(t) &= \max\{0, \min(t, b_j) - b_{j-1}\}. \end{aligned} \tag{7}$$

with boundaries $(b_0, b_1, b_2) = (0, H, G)$. An exact event (t, k) contributes $\mathcal{S}_i(t)\lambda_{ij(t)k}$; censoring at c contributes $\mathcal{S}_i(c)$. Thus unfinished tasks supply exposure without invented event counts. The active fitter requires exact event or censoring records; unknown failure clocks are not replaced by completion times.

Bounded artifact quality. Quality follows a clipped Gaussian working model: $Z_i \sim \mathcal{N}(\mu_i, \sigma_q^2)$, $q_i = \min(1, \max(0, Z_i))$, and $\mu_i = z_i^\top \gamma$. With standard-normal density φ and distribution Φ, its likelihood is

$$p(q_i \mid \mu_i) = \begin{cases} \Phi(-\mu_i/\sigma_q), & q_i = 0, \\ \varphi((q_i - \mu_i)/\sigma_q)/\sigma_q, & 0 < q_i < 1, \\ \Phi((\mu_i - 1)/\sigma_q), & q_i = 1. \end{cases} \tag{8}$$

This retains genuine probability mass at both endpoints. Only valid-delivery scores enter this likelihood. Recovered scores without delivery remain available for descriptive analysis. Timing and quality use separate coefficients and are conditionally independent; conditional independence across tasks holds given their coefficients and shared batch effects.

Hierarchical configuration priors. The predictor adds shared and domain-specific configuration functions, family and task offsets, and a batch term. Gaussian priors regularize these components; their variances are respectively $.01, .02, .1, .1, .1$. Domain-specific functions have independent coefficient blocks: compatible historical evidence updates the shared function while preserving uncertainty in an unobserved target domain. Configuration functions use a common finite candidate basis. Normalize each configuration feature by the maximum of one and its absolute maximum over that basis, obtaining $\widetilde{x}$. The global kernel is $k(x, x') = \exp[-\|\widetilde{x} - \widetilde{x}'\|^2/(.12p)]$ for p features. Alternatives use a normalized linear kernel, a topology-local/global mixture, or RF leaf proximity. The local component activates after three qualified topology-specific batches. RF proximity mixes with the global kernel at weights $.75/.25$; its forest has 32 trees, maximum depth four, bootstrap sampling, and seed 4317. It fits batch-level mean observed delivery quality, making this an empirical-Bayes alternative. Same-cause hazard coefficients have correlation $.5$ between intervals. Prior log-rate intercepts are -8.188689 for delivery and -11.366743 for failure in both intervals; the quality intercept is $.5$ with $\sigma_q = .25$.

Inference and incremental evidence. For Gaussian prior $\theta \sim \mathcal{N}(m, K)$, minimize the negative log posterior

$$\begin{aligned} \mathcal{L}(\theta) &= -\log p(\mathcal{D} \mid \theta) + \tfrac{1}{2}(\theta - m)^\top K^{-1}(\theta - m), \\ p(\theta \mid \mathcal{D}) &\approx \mathcal{N}(\widehat{\theta}, A^{-1}), \end{aligned} \tag{9}$$

where $A = \nabla^2 \mathcal{L}(\widehat{\theta})$. The timing likelihood uses event counts and at-risk exposure; the quality likelihood uses the delivered scores above. Trust-region Newton optimization uses analytic derivatives. Hypothetical observations condition this Gaussian approximation by refitting; model weights use Laplace marginal likelihoods. Both horizons project the same latent task outcomes. Their incremental log likelihood is $\log p(O_G \mid \theta) - \log p(O_H \mid \theta)$. Cumulative G evidence replaces its H training prefix, so an extended experiment remains one batch. Conditional evidence calculations start from the same pre-experiment posterior.

Fresh-batch recommendation. Recommendation evaluates delivery at H, including when an experiment observes to G. Experimental and recommendation batches have distinct nuisance variables. For each posterior draw, we integrate an independent fresh batch effect. Given first-interval hazards, task delivery probability is $p_i = \lambda_{iD}(1 - e^{-r_i H})/r_i$, with $r_i = \lambda_{iD} + \lambda_{iF}$. Complete-cohort probability is $\prod_i p_i$. Conditional delivery times follow an exponential distribution truncated at H. We combine sampled times with the analytical clipped-Gaussian mean and fixed family weights to integrate the complete-delivery utility. This prevents learning one favorable batch realization from being counted as persistent configuration improvement.

Finite information-value computation. Let $\mu_{\mathcal{D}}(x)$ denote posterior mean recommendation utility. On a frozen recommendation set, choose incumbent $x^* = \arg\max_x \mu_{\mathcal{D}}(x)$ and estimate

$$I(e) = \mathbb{E}_{O_e}\left[\max_x \mu_{\mathcal{D},O_e}(x) - \mu_{\mathcal{D},O_e}(x^*)\right]. \tag{10}$$

Under exact conditioning this equals the knowledge-gradient expression in the main text. Shared random numbers couple candidate comparisons and paired horizons. The deployed numerical contract evaluates up to 12 candidates using 32 hypothetical outcomes, 128 posterior draws, 64 inner draws, and four independent repetitions; ambiguous rankings trigger up to eight repetitions. Selection uses mean information value divided by mean node-seconds. Expected cost integrates time until all tasks reach either terminal cause, capped at h, adds overhead, and multiplies by three nodes. Survival integration uses 96-point quadrature, checked against 48 points; overhead starts from a 600-second prior with weight two and incorporates compatible observations. These approximations provide executable decisions.

E Complete Historical Accounting

Tables 4 and 5 retain all 54 settled attempts. Complete terminal quality includes attempts 50 and 53 after recovery; complete seal-time reporting still requires the original full-batch boundary. The first three attempts contain requested task slots but no admitted task execution. Counts of tasks, scores, requests, batches, and independent repetitions are kept distinct.

Historical software and protection conditions affect transfer. The older bounded-window segment included a request-truncation mechanism and partial delivery, so it is not a pure hardware-performance treatment. Later repair changes the observation process. Within a fixed measurement target, topology is an intended action variable rather than a reason to discard comparisons automatically; relevant execution differences still preclude causal attribution without a matched design. The tables are an evidence inventory, not an exchangeability certificate.

Table 4: Historical attempts 1–27. N is observed scores; R is recovered scores. Quality is the complete terminal value; time is in minutes. C: complete, W: closed window, F: failed, I: interrupted. Dashes retain unavailable values.

ID	State	N	R	Q	M_{seal}	M_{del}
1	F	0	0	–	–	–
2	F	0	0	–	–	–
3	F	0	0	–	–	–
4	C	50	0	0.8187	62.07	–
5	C	50	0	0.8102	32.11	–
6	C	50	0	0.8135	45.54	–
7	I	1	1	–	–	–
8	I	1	1	–	–	–
9	C	50	0	0.8021	30.43	–
10	I	20	0	–	–	–
11	C	50	0	0.8188	76.23	–
12	F	0	0	–	–	–
13	F	8	2	–	–	–
14	C	50	0	0.7992	75.46	–
15	I	21	14	–	–	–
16	I	14	1	–	–	–
17	F	2	1	–	–	–
18	C	50	0	0.8350	70.02	–
19	C	50	0	0.8272	99.57	–
20	F	18	0	–	–	–
21	C	50	0	0.8135	55.48	–
22	C	50	0	0.8268	87.29	–
23	C	50	0	0.8171	72.99	–
24	C	50	0	0.8222	31.10	–
25	I	49	0	–	–	–
26	C	50	0	0.8062	70.68	–
27	C	50	0	0.8299	69.63	–

Table 5: Historical attempts 28–54. N is observed scores; R is recovered scores. Quality is the complete terminal value; time is in minutes. C: complete, W: closed window, F: failed, I: interrupted. Dashes retain unavailable values.

ID	State	N	R	Q	M_{seal}	M_{del}
28	I	40	0	–	–	–
29	W	38	15	–	–	–
30	W	37	22	–	–	–
31	W	39	17	–	–	–
32	W	39	15	–	–	–
33	W	40	15	–	–	–
34	W	44	12	–	–	–
35	W	38	17	–	–	–
36	W	41	12	–	–	–
37	W	41	17	–	–	–
38	W	39	15	–	–	–
39	W	40	17	–	–	–
40	W	42	16	–	–	–
41	W	41	15	–	–	–
42	W	41	18	–	–	–
43	C	50	0	0.8185	40.68	40.92
44	C	50	0	0.8122	60.08	60.31
45	W	32	1	–	–	–
46	C	50	0	0.7998	66.54	66.78
47	W	49	0	–	–	–
48	C	50	0	0.8001	35.28	35.52
49	W	39	0	–	–	–
50	W	50	1	0.8085	–	–
51	W	49	0	–	–	–
52	W	37	0	–	–	–
53	W	50	1	0.8057	–	–
54	W	40	0	–	–	–

F Extended Task and Request Results

Table 6: All post-repair deadline observations. N is terminal score coverage, D is valid deadline deliveries, and L_h is minutes. Complete artifact quality and deadline quality credit are distinct.

ID	N	D	Q	C_h	L_h
43	50	50	0.8185	0.8185	40.92
44	50	50	0.8122	0.8122	60.31
45	32	31	–	0.4437	98.47
46	50	50	0.7998	0.7998	66.78
47	49	49	–	0.8014	52.11
48	50	50	0.8001	0.8001	35.52
49	39	39	–	0.6039	90.46
50	50	49	0.8085	0.8085	51.85
51	49	49	–	0.7681	53.02
52	37	37	–	0.6197	93.02
53	50	49	0.8057	0.8024	42.02
54	40	40	–	0.6470	85.24

Table 7: Request populations and completed-request means. $N_c/N_t/N_a$: completed/timed/all records. W and S are seconds; E is completed-request queue exposure in percent.

ID	N_c	N_t	N_a	W	S	E (%)
43	1175	1175	1175	1.02	97.79	1.03
44	1170	1170	1170	44.94	101.82	30.62
45	777	790	797	250.45	103.93	70.67
46	1086	1086	1086	41.31	134.61	23.48
47	963	963	964	30.46	122.33	19.94
48	1043	1043	1043	0.03	92.18	0.03
49	879	889	889	127.76	167.50	43.27
50	941	941	942	23.43	134.13	14.87
51	1143	1143	1143	20.26	108.04	15.79
52	805	817	819	226.11	97.88	69.79
53	1015	1015	1016	2.79	113.60	2.40
54	974	983	986	161.80	80.97	66.65

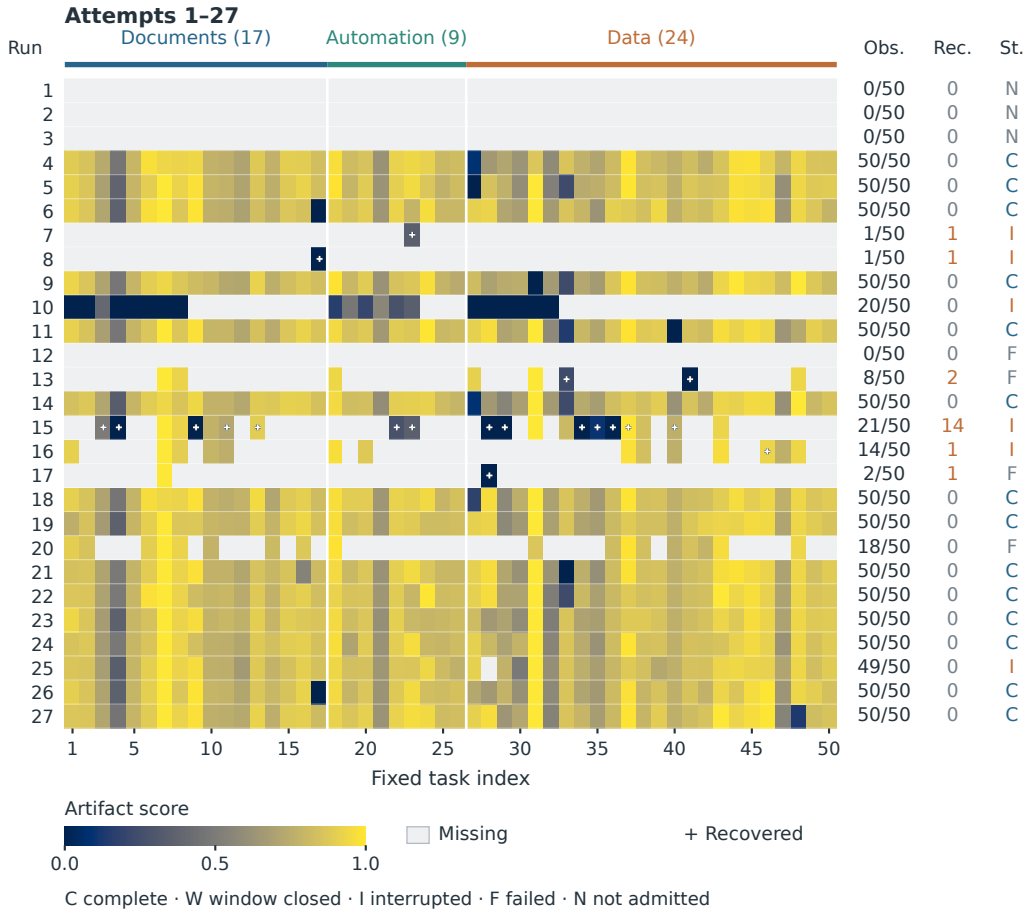

Figure 4: Terminal artifact scores for attempts 1–27. Columns retain task identity and family order; missing scores are distinct from zero. Plus signs mark recovered scores. Row annotations preserve observation and recovery counts. The first three attempts admitted no tasks.

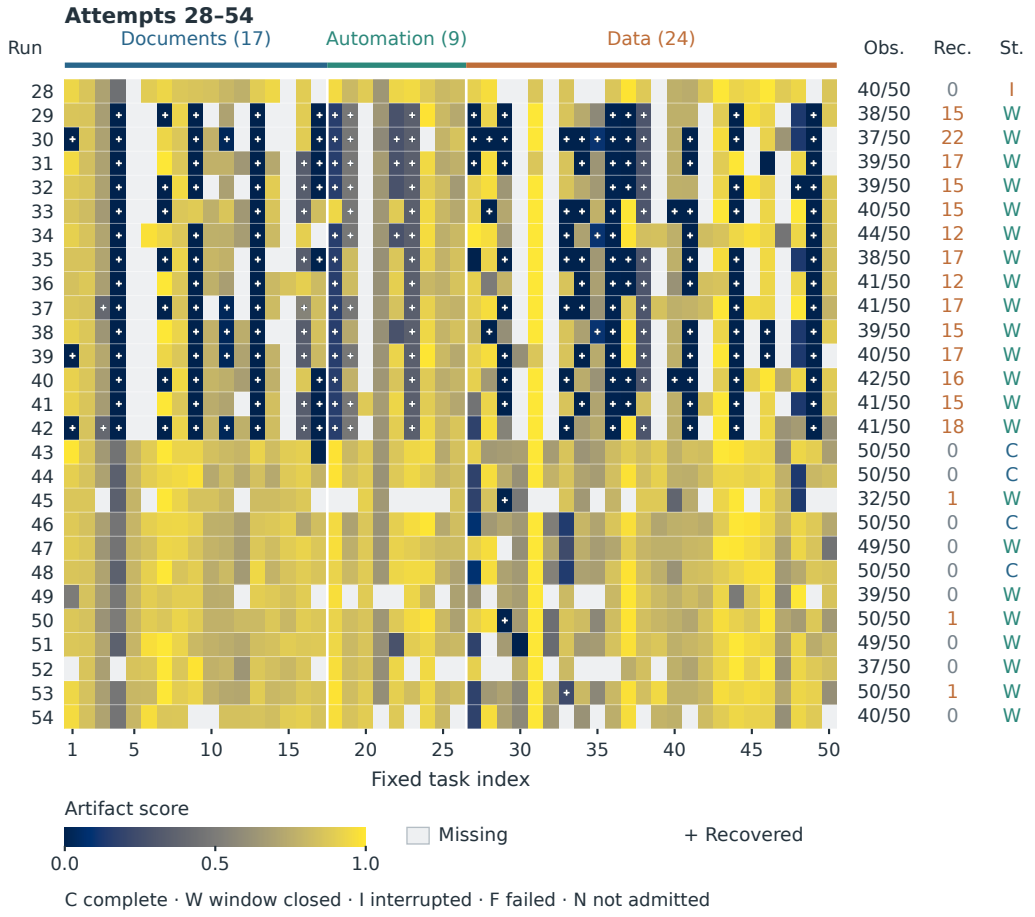

Figure 5: Terminal artifact scores for attempts 28–54, with the same ordering and scale as the preceding panel. Recovered scores do not establish successful deadline delivery.

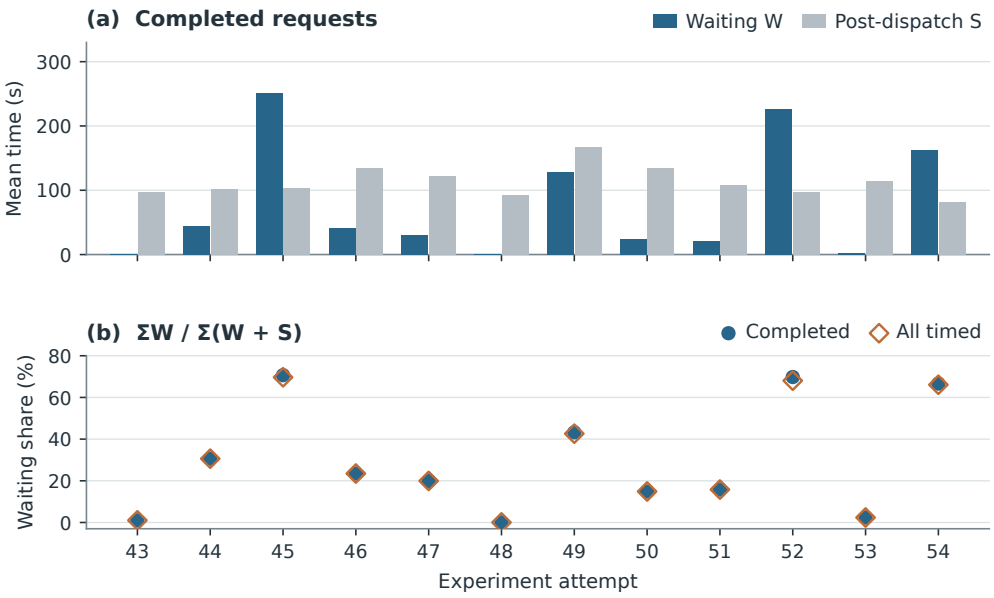

Figure 6: Completed-request waiting and post-dispatch means, with completed versus all-timed exposure ratios. Coverage counts retain terminal records lacking timing. The ratios are neither wall-time fractions nor GPU idle rates.

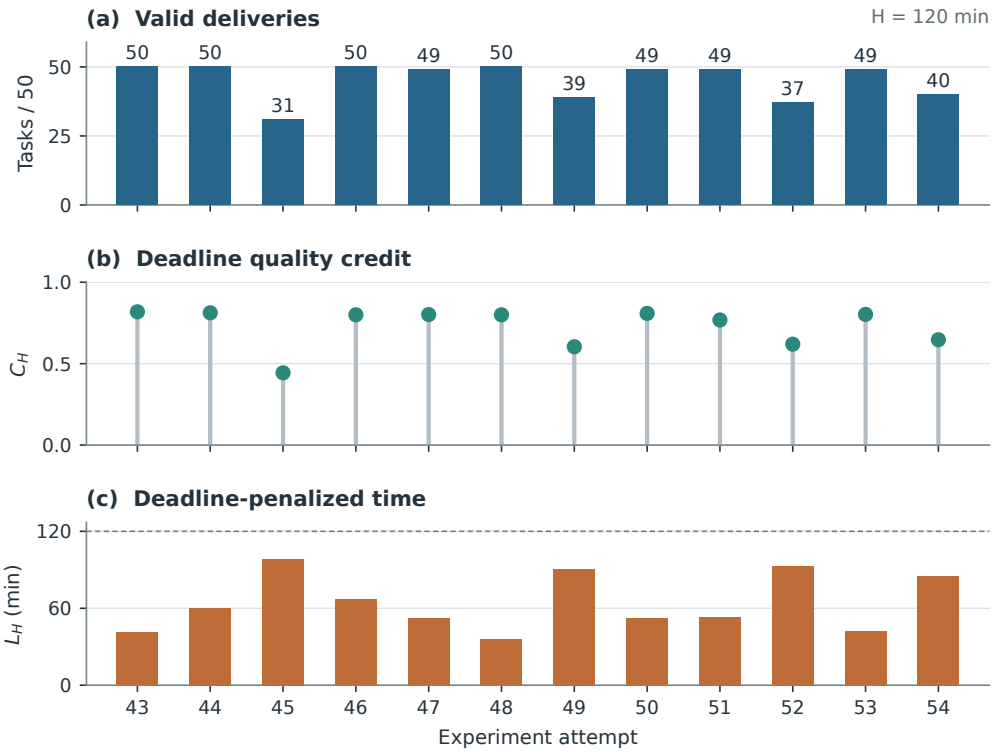

Figure 7: Deadline delivery, quality credit and penalized mean time across all twelve post-repair historical batches. Terminal recovered quality is not substituted into the original deadline observation.

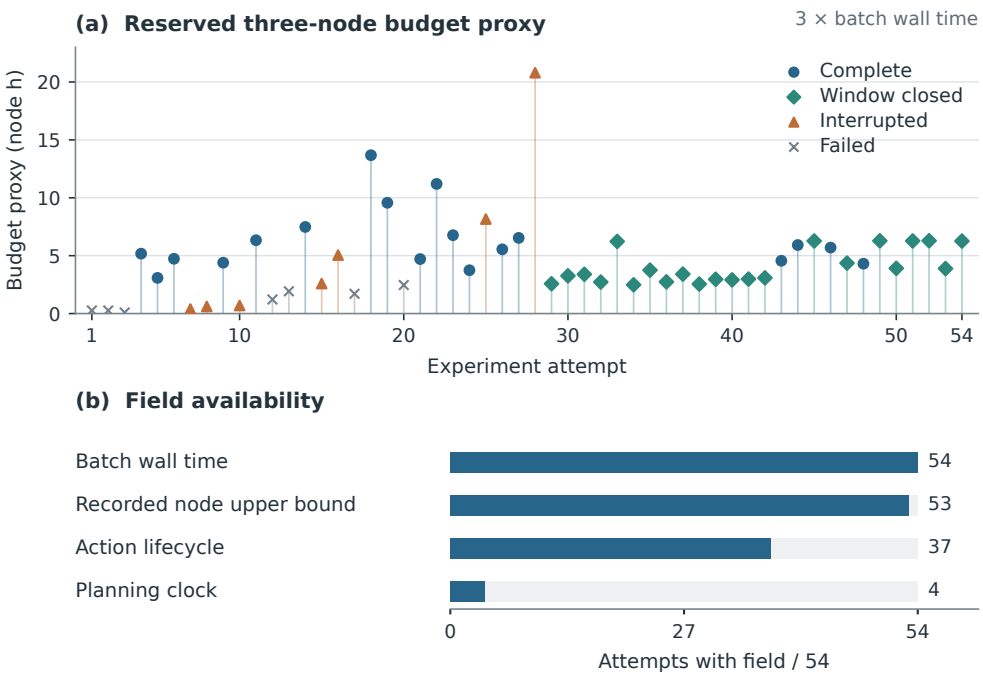

Figure 8: Historical reserved-budget proxy and cost-field coverage. All 54 settled attempts contribute; wall, recorded node-upper-bound, lifecycle, and planning fields have distinct availability. This is descriptive expenditure, not a final-controller learning curve.

G Cost and Prospective Evaluation

Using three times each recorded wall cost yields a uniform reserved-three-node proxy of 248.25 node-hours across the 54 attempts. This figure includes unsuccessful attempts but is not an allocation-duration integral, energy measurement, or complete accounting of out-of-batch probes and historical planning. The newer lifecycle contract explicitly includes planning, whereas older wall fields do not consistently do so. We therefore do not join these fields into an unlabeled algorithm-efficiency curve.

The primary prospective comparison should hold initial evidence, tasks, hardware budget, feasible actions, and scoring fixed. Random admissible search and a batch-summary-only surrogate are informative baselines. Evaluate recommendations independently at common cumulative-cost checkpoints, retaining failures and setup costs. Task/family-structure and joint-selection ablations should change the specified component alone. Native-routing ablations require additional admitted policies; the single-policy release cannot supply them. Repetitions must be at the deployment level for claims about configuration uncertainty, rather than treating requests within one batch as independent replications.

A decision-case record should link the evidence cutoff, pre-deployment predictive distribution, value and cost estimates, selected candidate, actual execution, measured outcome, and next decision. If an action is chosen for coverage or repair, preserve that reason. A planning-only record must never be paired with an unrelated historical task result to simulate a closed loop. This evaluation separates whether service improves from whether the process of learning is cost-effective.