CREATIVE DESIGN OF AGENTIC SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Frontier models are increasingly contributing to mathematical problem solving and scientific discovery, making creativity a central capability to evaluate beyond established task performance. Existing agent benchmarks keep the agent architecture fixed and measure execution ability, while creativity studies assess generated content rather than the executable systems agents build. Therefore, we present Agents Creating Agents (ACA), **the first benchmark for evaluating the creativity of agents from an architectural perspective**: an architect transforms real development tasks into an executable architecture satisfying utility, novelty, and cost awareness, which together form the Creativity score. We further propose **Evidence-Grounded Mechanism Search (EGMS)**, guides architecture search using the causal structure of failures rather than a scalar development score. EGMS improves mean utility by **6.4 points** over the strongest of five baselines and raises the Creativity score to **0.272** ↑. Further analysis shows that the harness influences architectural creativity as much as the backbone model, and a small local model with a compiled domain architecture outperforms `GPT-5.5` on the Codex harness with runtime skill injection, suggesting that architecture can substitute for model scale at execution time. Our code is open at `https://anonymous.4open.science/r/ACA-BENCHMARK-2775/`.

1 INTRODUCTION

Frontier models from OpenAI and Anthropic are increasingly contributing to mathematical problem solving and scientific discovery, making creativity an important capability beyond performance on established tasks. OpenAI reports GPT-assisted scientific results and a proposed solution to the Navier–Stokes Millennium Prize Problem from an internal model,[1] while Anthropic reports `Claude Fable 5`'s resolution of the Jacobian conjecture.[2] These advances sharpen an evaluation challenge: a creative output must be original relative to existing knowledge and useful under independent verification. Creativity studies have primarily assessed stories, ideas, and diverse answers (Chakrabarty et al., 2024; Si et al., 2025; Zhang et al., 2025c), but agents can also express creativity through executable systems (Hu et al., 2025; Ma et al., 2026). Strong reasoning and tool use make the harness that organizes these capabilities consequential (Young, 2025; Lopopolo, 2026), yet harnesses remain largely handcrafted (Hu et al., 2025; Kapoor et al., 2026). Reusable skills offer another route, but selecting and loading them at runtime incurs recurring costs, with benefits that vary across tasks (Li et al., 2026b; Hajimiri et al., 2026). Can an agent instead turn domain knowledge into a reusable architecture that is effective, novel, and cost-aware?

Agent evaluation has matured around execution benchmarks, spanning repository repair (Jimenez et al., 2024), web and desktop environments (Zhou et al., 2024; Drouin et al., 2024; Xie et al., 2024), and interactive application and enterprise settings (Trivedi et al., 2024; Yao et al., 2024; Xu et al., 2026; Microsoft, 2026). These benchmarks provide realistic environments and programmatic evaluation, but keep the agent architecture fixed: they measure execution ability rather than architectural creativity. A parallel line of work automates agent design, searching over code, workflows, modular components, and harnesses (Hu et al., 2025; Zhang et al., 2025b; Shang et al., 2025; Liu et al., 2024; Zhou et al., 2026; Lee et al., 2026; Yang et al., 2026), but typically optimizes a scalar development score and neither freezes candidates for evaluation under a shared protocol nor provides a reference library for judging whether a discovered architecture is structurally novel. A third line distills reusable knowledge as prompts, memory, or skills (Wang et al., 2024; Ouyang et al., 2026;

[1] `https://openai.com/index/accelerating-science-gpt-5/`; `https://openai.com/index/navier-stokes-solution/`

[2] `https://www.anthropic.com/research/discovering-cryptographic-weaknesses`

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

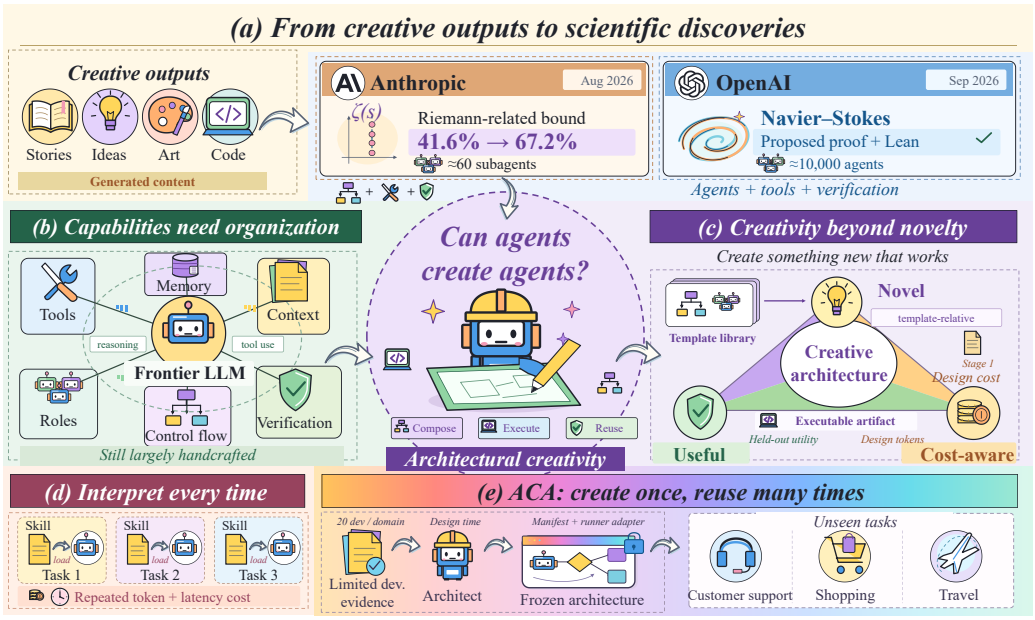


Figure 1: From creative outputs to architectural creativity. (a) Frontier models already generate creative content and scientific results. (b) These capabilities still depend on an organizing harness that remains largely handcrafted. (c) We treat the architecture itself as the creative artifact. (d) Interpreting skill documents at runtime repeats token and latency cost on every task. (e) ACA instead creates once and reuses many times: an Architect turns limited development evidence into a manifest and runner adapter, which is frozen and then executed on unseen tasks.

Agrawal et al., 2026; Li et al., 2026b; Gautam et al., 2026); these artifacts are repeatedly loaded at inference time, and runtime injection may not justify its token cost (Hajimiri et al., 2026). Finally, work on LLM creativity (Zhang et al., 2025c; Chakrabarty et al., 2024; Lu et al., 2025; Cao et al., 2026) evaluates originality in generated content rather than executable systems.

These limitations expose three gaps in measuring agent creativity: there is no established operational definition that asks an architecture to be useful, novel, and cost-aware at the same time; structural novelty has no widely used frozen reference against which it can be assessed; and created architectures are seldom frozen and then evaluated on hidden tasks. We therefore define the agent architecture itself as the unit of creativity: created from development evidence, frozen, and then evaluated for utility, novelty, and cost on held-out tasks, as illustrated in Figure 1c.

We introduce Agents Creating Agents (ACA), a benchmark that operationalizes this definition. Given a task-family specification, public tool schemas, and limited development tasks, an Architect produces an executable architecture consisting of a manifest and runner adapter. The architecture is then frozen and evaluated once per held-out task in a live environment with an inner tool-feedback loop. We score each architecture on held-out utility, novelty judged by an agent against a frozen library, and normalized token cost, and combine them into a Creativity score. On the same benchmark, we propose Evidence-Grounded Mechanism Search (EGMS), which guides architecture search using the causal structure of failures rather than a scalar development score. Failures are grouped by causal decision, repairs target shared mechanisms with falsification rules, and candidates are accepted only when they fix the underlying cause, survive generated counterexamples, and preserve previously successful capabilities.

Our experiments demonstrate that current model-harness combinations show limited architectural creativity: even the strongest combination reaches a Creativity score of only 0.33, indicating that the harness shapes architectural creativity as much as the model itself. Building on this observation, we evaluate EGMS against five architecture-search baselines under the same combination and budget; by supervising search with the causal structure of failures rather than a scalar development score, EGMS improves mean held-out utility by 6.4 ↑ points over the strongest baseline and raises the Creativity score 0.153 ∼ 0.272 ↑. Finally, once a domain-adapted architecture is created, a small local model `Qwen3.6-27B` executing it solves 49.6% of held-out tasks, versus 36.2% for `GPT-5.5`

driving the Codex harness with runtime skill injection, suggesting that architectural quality can partially substitute for model scale at execution time.

① ***The ACA benchmark***: a benchmark for evaluating the creativity of agents from an architectural perspective, with a two-stage create–freeze–evaluate protocol and a Creativity score combining held-out utility, agent-judged novelty, and design cost. To our knowledge, ACA is the first benchmark of this kind.

② ***A compiled view of domain skills***: rather than repeatedly injecting skill documents at runtime, ACA compiles domain workflow knowledge into deterministic subagent architectures at design time. Several generated adapters require zero runtime model calls, and the resulting architectures achieve roughly $2.5\times\downarrow$ lower token cost per solved task than a monolithic harness.

③ ***Evidence-Grounded Mechanism Search (EGMS)***: a search method that uses causal failure structure, mechanism-level repair, falsification, and counterexample-gated acceptance instead of a scalar development score. EGMS improves mean held-out utility by $6.4\uparrow$ points over the strongest of five baselines and increases the Creativity score $0.153 \sim 0.272\uparrow$. Component ablations further show that each mechanism contributes to the overall gain.

④ ***Evidence that architecture can substitute for model scale***: a small local runtime model executing a domain-adapted architecture solves 49.6% of held-out tasks, outperforming `GPT-5.5` operating Codex harness with runtime skill injection while using about $2.4\times\downarrow$ fewer tokens per solved task. This indicates that the benefit of a well-designed architecture persists even when the executing model is substantially smaller.

2 Related Work

Creativity in LLMs and agents. *(i) Creative writing and diverse generation.* Art or Artifice introduces expert assessments of creative writing, while NoveltyBench evaluates the diversity and quality of model generations (Chakrabarty et al., 2024; Zhang et al., 2025c); CreataSet and CrEval extend textual creativity evaluation across domains (Cao et al., 2026). These studies make originality measurable beyond answer correctness, but their evaluated objects are textual responses. *(ii) Research ideation and executable creativity.* MetaMuse goes beyond text by guiding creative algorithm generation with diversity and usefulness in performance space (Ma et al., 2026). Our target is instead the reusable agent architecture, whose utility, novelty, and design cost are evaluated together.

Automated agent design and reusable knowledge. *(i) Architecture and workflow search.* ADAS programs agents in code, AFlow searches workflow code with Monte Carlo tree search, AgentSquare searches modular components, DyLAN adjusts agent teams for individual tasks, and MASS optimizes prompts and topologies (Hu et al., 2025; Zhang et al., 2025b; Shang et al., 2025; Liu et al., 2024; Zhou et al., 2026). These methods optimize coordination for task performance; ACA supplies a common protocol for assessing the resulting architecture as a created artifact. *(ii) Harness adaptation.* Meta-Harness searches the full scaffold, and Better Harnesses studies automated adaptation for lower execution cost (Lee et al., 2026; Yang et al., 2026). EGMS instead focuses on the development signal used to search that space: causal failure clusters, mechanism-level repairs, and counterexample-gated acceptance. *(iii) Prompts, memory, and skills.* ReasoningBank and GEPA distill reasoning memory and evolve prompts through reflective feedback (Ouyang et al., 2026; Agrawal et al., 2026); SkillsBench and SkillAxe evaluate and refine skill documents, while SkillFlow, SkillLearnBench, and SWE-Skills-Bench examine skill discovery, continual learning, and software-engineering transfer (Li et al., 2026b; Gautam et al., 2026; Zhang et al., 2026; Zhong et al., 2026; Han et al., 2026). These approaches primarily improve knowledge consumed by an agent loop; ACA instead embeds domain knowledge in executable control flow and tool policies, and evaluates the resulting execution cost.

Agent execution and evaluation benchmarks. *(i) Software, web, and desktop tasks.* SWE-bench uses tests to evaluate repository repair, GAIA evaluates general assistance, and WebArena, WorkArena, and OSWorld provide realistic web, knowledge-work, and desktop environments (Jimenez et al., 2024; Mialon et al., 2023; Zhou et al., 2024; Drouin et al., 2024; Xie et al., 2024). These environments support reproducible execution comparisons, but execution success does not itself measure the creativity of an architecture's design. *(ii) Stateful and interactive workflows.* AppWorld exposes executable application APIs, τ-bench includes tool–agent–user interaction, and TheAgentCompany evaluates enterprise work through environment states (Trivedi et al., 2024; Yao et al., 2024; Xu et al., 2026). *(iii) Evaluation infrastructure, cost, and reliability.* HAL standardizes evaluations across models, scaffolds, and benchmarks with cost accounting and log inspec-

tion (Kapoor et al., 2026), and the Science of AI Agent Reliability complements success rates with consistency, robustness, predictability, and safety measurements (Rabanser et al., 2026), together showing why a single success rate is insufficient.

Nevertheless, performance-guided search and content-oriented creativity assessment do not by themselves establish whether an architecture is both structurally novel and useful on unseen executions. In contrast, ACA freezes the generated manifest and adapter before hidden-task evaluation and measures utility, novelty, and design cost together, and its EGMS search accepts mechanism-level repairs only after counterexample testing and preservation of successful capabilities.

3 METHODOLOGY

ACA evaluates architectures created from limited development evidence and reused on hidden tasks. We first define this protocol and its task corpus, then introduce Evidence-Grounded Mechanism Search (EGMS) as a harness-agnostic method for architecture creation.

3.1 TASK DEFINITION

An Architect A receives a task family $\mathcal{F}$, public tool schemas $\mathcal{T}$, and development tasks $\mathcal{D}_{\text{dev}}$. It produces an executable architecture $\pi = (m, a)$: the manifest m declares roles, control flow, communication, memory, and tool policies, while the adapter a implements the runtime policy. The adapter may call a shared model ϕ, identical across compared systems. The design objective is

$$\pi^* = \arg\max_{\pi} \ \mathbb{E}_{x \sim \mathcal{D}_{\text{hid}}}[U(\pi, x)], \tag{1}$$

where $U(\pi, x) \in [0, 1]$ measures task utility and $\mathcal{D}_{\text{hid}}$ is hidden from the Architect. Search receives only development traces and pass/fail evidence; A is a guidance-conditioned black-box generator.

Task inputs and information boundary. Each task is $x = (r_x, s_x^0, \mathcal{Q}_x)$, comprising a public request, initial environment state, and private state requirements. Tasks are partitioned before creation as $\mathcal{D} = \mathcal{D}_{\text{dev}} \dot{\cup} \mathcal{D}_{\text{hid}}$, where $\dot{\cup}$ denotes disjoint union. Development packages and their feedback are available during design. On a hidden task, the executor receives a sanitized request and public tool observations; the full state and requirements remain private to the evaluator. Define the public design input as $\mathcal{I}_{\text{dev}} = (\mathcal{F}, \mathcal{T}, \mathcal{D}_{\text{dev}})$.

Creation and reuse. The Architect creates an artifact within budget K, then freezes it for all hidden executions:

$$\pi = A(\mathcal{I}_{\text{dev}}; K), \qquad \bar{\pi} = \text{Freeze}(\pi), \qquad \pi_x = \bar{\pi} \quad (x \in \mathcal{D}_{\text{hid}}). \tag{2}$$

Freezing fixes the manifest and adapter, while allowing task-local memory and responses to tool feedback. Hidden outcomes cannot redesign the artifact.

Interactive execution. For public history $h_t = (r_x, u_{<t}, o_{\le t})$, tool calls alternate with environment feedback:

$$u_t = a(h_t; \phi), \qquad (s_{t+1}, o_{t+1}) = \mathcal{E}_x(s_t, u_t), \qquad s_0 = s_x^0. \tag{3}$$

Here o_0 is the initial public input. The runtime model is optional: the adapter may implement a decision in code. Each task receives one attempt, which ends at step T_x when the adapter terminates or the task budget is exhausted. Only the evaluator checks $\mathcal{Q}_x$ on the final state; private requirements never serve as the executor's stopping oracle. Section 3.4 defines the utility, novelty, and cost measures. Algorithm 1 summarizes their information boundary and execution protocol.

3.2 BENCHMARK CONSTRUCTION

Figure 2 left summarizes the ACA corpus and its domain and task-family coverage. We construct the benchmark by separating the information available during architecture creation. ACA adapts STATE-Bench (Microsoft, 2026) into 212 tasks across customer support, shopping, and travel. Each domain supplies 20 development tasks; the remaining tasks are held out for single-attempt, private state-based evaluation. A frozen library of 100 architecture templates provides the novelty reference. Task filtering, input sanitization, split construction, and library coverage are detailed in Appendix A.

3.3 EGMS: EVIDENCE-GROUNDED MECHANISM SEARCH

As shown in Figure 2 right, EGMS is a fixed-point iteration over five operators. Among them, Eval and the acceptance and checkpoint checks are executed deterministically by the harness, while Δ, $\mathcal{R}$, and CE are performed by the Architect model guided by the method's instructions.

The search runs for a budget of K repair iterations, each costing one development evaluation over $\mathcal{D}_{\text{dev}}$. The first one or two iterations compete genuinely different design directions, and the remaining iterations improve the development leader, an explicit explore-then-exploit allocation that

Algorithm 1 ACA creation and frozen evaluation

Input: Architect A; public $\mathcal{I}_{\text{dev}}$; private $\mathcal{D}_{\text{hid}}$; budget K; library L; cost reference C_{ref}
Output: Frozen $\bar{\pi}$; $\widehat{U}$, N, C_{design}, C_{exec}, and Creativity S

// Create using development evidence only
$\pi \leftarrow A(\mathcal{I}_{\text{dev}}; K)$; $C_{\text{design}} \leftarrow$ Tokens(Stage 1)
$\bar{\pi} \leftarrow \text{Freeze}(\pi)$ ▷ Eq. (2)
for $x \in \mathcal{D}_{\text{hid}}$ **do**
  $(s_{T_x}, \tau_x, C_x) \leftarrow \text{RunOnce}(\bar{\pi}, r_x, \mathcal{E}_x, s_x^0)$ ▷ Eq. (3)
  $y_x \leftarrow \mathbf{1}[\bigwedge_{q \in \mathcal{Q}_x} q(s_{T_x})]$ ▷ Eq. (8)
// Score privately; never update the frozen artifact
$\widehat{U} \leftarrow |\mathcal{D}_{\text{hid}}|^{-1} \sum_x y_x$; $C_{\text{exec}} \leftarrow \sum_x C_x$
$N \leftarrow \text{Novelty}(\bar{\pi}, L, \{\tau_x\})$ ▷ Eq. (9)
$S \leftarrow \widehat{U} N[1 - \min(1, C_{\text{design}}/C_{\text{ref}})]$ ▷ Eq. (10)
return $(\bar{\pi}, \widehat{U}, N, C_{\text{design}}, C_{\text{exec}}, S)$

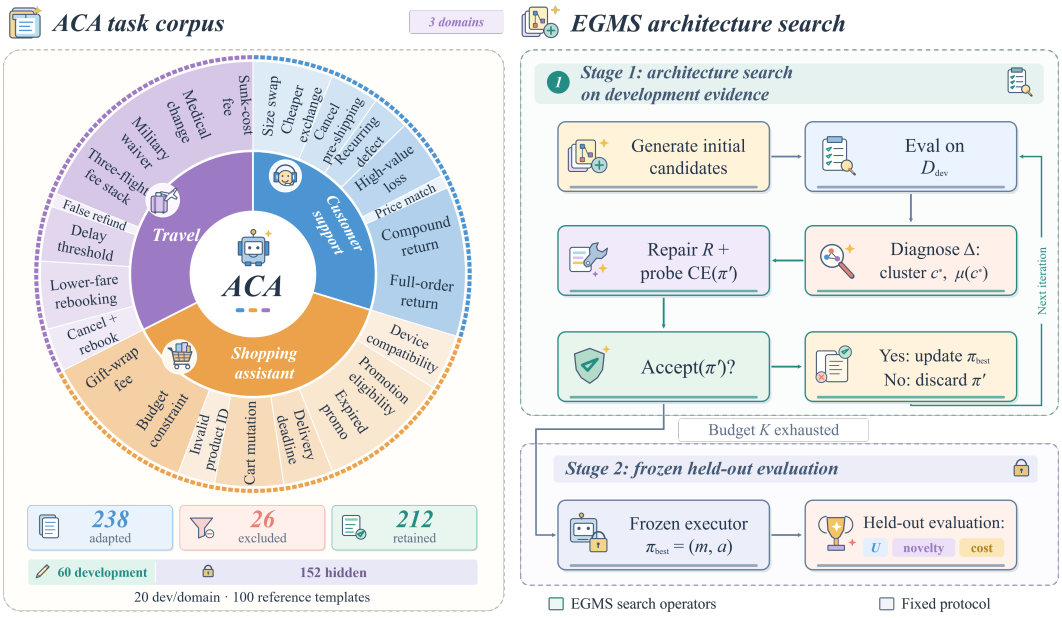


Figure 2: Overview of the ACA task corpus and the EGMS search procedure. Left: ACA task corpus and split protocol. The inner ring partitions 212 adapted tasks across the three domains, the middle ring shows task families from the frozen taxonomy, and the outer labels name representative held-out tasks. Right: EGMS uses two stages. Stage 1 iterates evaluation, causal diagnosis, mechanism-level repair, and counterexample-gated acceptance on development evidence, updating π_{best} only with accepted repairs. Stage 2 freezes and evaluates the best checkpoint on held-out tasks.

Section 4 relates to token cost. $\text{Eval}(\pi, \mathcal{D}_{\text{dev}})$ executes the candidate on the development set and returns execution traces and the failure set $\mathcal{F}$.

Causal diagnosis. $\Delta : (\pi, \mathcal{F}) \mapsto \{c_1, \ldots, c_k\}$ partitions failures by the first causal decision at which a trace diverges from valid behavior. For a failing trace τ with decisions $a_1, a_2, \ldots$, the divergence point is

$$i^*(\tau) = \min\{\, i : a_i \notin \text{Valid}(\mathcal{T} \mid o_{<i}) \,\}, \tag{4}$$

the earliest decision that is invalid given the evidence available when it was made. Two failures belong to the same cluster if they share the divergence mechanism at i^*, not if they share surface symptoms. The repair target is the dominant cluster c^*.

Mechanism-level repair and falsification. $\mathcal{R} : (\pi, \mu(c^*)) \mapsto \pi'$ modifies the shared mechanism $\mu(c^*)$ responsible for the cluster, using the cluster's traces as evidence. If the same root cause survives two mechanism-level repairs, the design direction is treated as falsified and the candidate

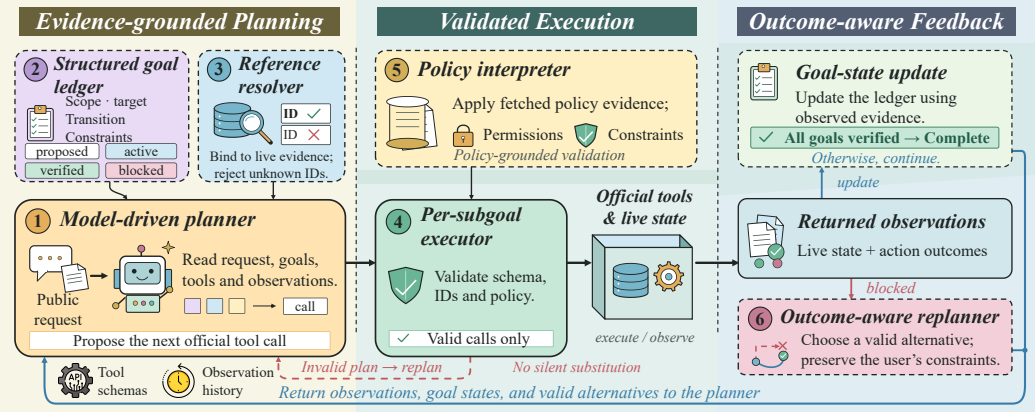


Figure 3: Six design-space priors connect evidence-grounded planning, validated execution, and outcome-aware feedback.

is rebuilt along a different direction instead of being extended with another branch:

$$\pi' = \mathcal{R}(\pi, \mu(c^*)), \qquad \text{survive}(\mu(c^*), 2) \Rightarrow \text{rebuild}(\pi'). \tag{5}$$

Counterexample probing. $\text{CE} : \pi' \mapsto \{e_1, \ldots, e_m\}$ generates probes by varying the surface form, logical structure, references, scale, information availability, observation order, and environment responses of development tasks. Probing is a falsification device: it tests whether a repair addresses the mechanism or merely the observed instance. The acceptance gate thus protects utility directly through the regression conjunct, while the probing supplies behavioral evidence that an accepted repair changes the mechanism, which is what the novelty term later measures.

Acceptance and selection. A repaired candidate is accepted only if it fixes the target cluster, passes all probes, and preserves every previously passing capability cluster:

$$\text{Accept}(\pi') \iff \text{Fixed}(\pi', c^*) \wedge \big[\forall e \in \text{CE}(\pi') : \text{Pass}(\pi', e)\big] \wedge \big[\forall c \in \mathcal{C}_{\text{pass}} : \neg\text{Regress}(\pi', c)\big]. \tag{6}$$

Let $v_{\text{dev}}(\pi) = (\text{passed}, -\text{errors}, -\text{failed})_\pi$. Accepted candidates update the best checkpoint only under a strict lexicographic improvement:

$$\pi_{\text{best}} \leftarrow \pi' \iff v_{\text{dev}}(\pi') \succ_{\text{lex}} v_{\text{dev}}(\pi_{\text{best}}). \tag{7}$$

The loop terminates when the budget K is exhausted, after which π_{best} is frozen. By construction, the resulting search satisfies two useful guarantees: (i) *monotone checkpointing*: the development score of π_{best} under $\succ_{\text{lex}}$ is non-decreasing across iterations, since Eq. 7 updates the checkpoint only when a strictly better candidate is found; and (ii) *regression-free acceptance*: by the third conjunct of Eq. 6, every accepted candidate preserves all capability clusters that are passing at acceptance time. Thus, a regressive candidate may be explored, but it cannot replace the current checkpoint.

The search procedure above is representation-agnostic, but unconstrained search over π is sample-inefficient. EGMS therefore restricts the search to a prior subspace $\Pi_{\text{prior}} \subseteq \Pi$, where every architecture instantiates the six mechanisms summarized in Figure 3.

In particular, execution is model-driven: at round t, the runtime model selects the next valid tool call as $a_t = \phi(r, G_t, \mathcal{T}, o_{\le t})$, with $a_t \in \text{Valid}(\mathcal{T})$, while deterministic code is used only for schema, policy, safety, and completion checks; invalid actions are returned to the model for replanning rather than silently replaced. Execution is further grounded by a structured goal ledger $G = \{g_j\}$, where each atomic goal $g_j = (\text{scope}_j, \text{target}_j, \text{transition}_j, \text{constraints}_j)$ is tracked through $\ell_j \in \{\text{proposed}, \text{active}, \text{verified}, \text{blocked}\}$, and execution proceeds until all $\ell_j = \text{verified}$. The execution trajectories additionally obey the validity constraints detailed in Appendix C.1.

3.4 Benchmark Evaluation

ACA evaluates the artifact on hidden utility, template-relative novelty, and design cost. The execution cost is measured separately to assess reuse.

Utility. For $n = |\mathcal{D}_{\text{hid}}|$ hidden tasks, a single execution succeeds only if its final state satisfies all private requirements:

$$U(\pi, x) = \mathbf{1}\left[\bigwedge_{q \in \mathcal{Q}_x} q(s_{T_x})\right], \qquad \widehat{U}(\pi) = \frac{1}{n} \sum_{x \in \mathcal{D}_{\text{hid}}} U(\pi, x). \tag{8}$$

Novelty. Let $R_{10}(\pi, L)$ retrieve ten structurally similar templates from the frozen library L. A locked agent-as-judge assigns $d_j(\pi, T) \in \{0, 1, 2\}$ for each dimension $j \in \mathcal{J} = \{\text{roles}, \text{control}, \text{communication}, \text{tools}, \text{memory}\}$ using manifest, adapter, and public trace evidence. The nearest-template score is

$$N(\pi) = \min_{T \in R_{10}(\pi, L)} \frac{1}{10} \sum_{j \in \mathcal{J}} d_j(\pi, T) \in [0, 1]. \tag{9}$$

The dimension rubric and evidence rules are specified in Appendix B.1.

Design cost and composite. Let C_{design} count Stage 1 input and output tokens across creation, development evaluation, and repair. With a common fixed reference C_{ref}, normalized cost and CreativityScore S are

$$\widetilde{C} = \min\left(1, \frac{C_{\text{design}}}{C_{\text{ref}}}\right), \qquad S(\pi) = \widehat{U}(\pi)\, N(\pi)\, (1 - \widetilde{C}). \tag{10}$$

The reference is shared across tables. Stage 2 tokens are excluded from C_{design} and reported per task and per solved task. Appendix B.2 gives the cost reference, boundaries, and conventions.

4 Experiments

We organize the experiments around four research questions (RQs).

RQ1: How do different Architect systems (harness $\times$ model) compare in the architectural creativity?
RQ2: Does our search method produce better architectures than established search baselines?
RQ3: Which components of the method are responsible for its effectiveness?
RQ4: Does creating a reusable agent architecture reduce execution cost and improve utility?

4.1 Experimental Setup

Harnesses and models. For RQ1, we compare `GPT-5.5`, `DeepSeek-V4-Flash`, and `Qwen3.8-Max` with Codex, ClaudeCode, and OpenClaw, plus ClaudeCode (`Opus 4.8`), yielding ten model–harness combinations. RQ2 and RQ3 fix the Architect to Codex with `GPT-5.5`.

Baselines. For RQ2, we adapt five methods: ADAS (Hu et al., 2025), AFlow (Zhang et al., 2025b), GEPA (Agrawal et al., 2026), Meta-Harness (Lee et al., 2026), and Better Harnesses (Yang et al., 2026). All methods use the same Architect and search budget, differing only in guidance rather than their original execution loops.

Metrics. We report held-out utility U, template-relative novelty N, Stage 1 design tokens C (M), and CreativityScore S, defined in Section 3.4. RQ4 additionally reports Stage 2 tokens per task and per solved task; full accounting and reporting details appear in Appendix B.2.

4.2 Architect System Comparison (RQ1)

Each model–harness combination is run and evaluated three times; Figure 4 reports the highest single-run result. Table 5 provides the corresponding metric components.

Obs. 1: The best harness depends on the backbone model. For `DeepSeek-V4-Flash`, switching from Codex to ClaudeCode raises utility 36.18% $\sim$ 54.61% and creativity 0.1322 $\sim$ 0.3320. `GPT-5.5` instead achieves its highest creativity with OpenClaw (0.2819), as does `Qwen3.8-Max` (0.1857). These results favor evaluating model–harness combinations rather than assigning a single ranking to backbones or harnesses in isolation.

Obs. 2: Utility and novelty must be evaluated jointly. With `GPT-5.5`, Codex achieves slightly higher utility than OpenClaw (50.66% versus 50.00%), but lower novelty and creativity. Conversely, `Qwen3.8-Max` with Codex has novelty 0.567 but utility 29.61%, yielding creativity 0.1657. High utility or novelty alone therefore does not determine the composite ranking.

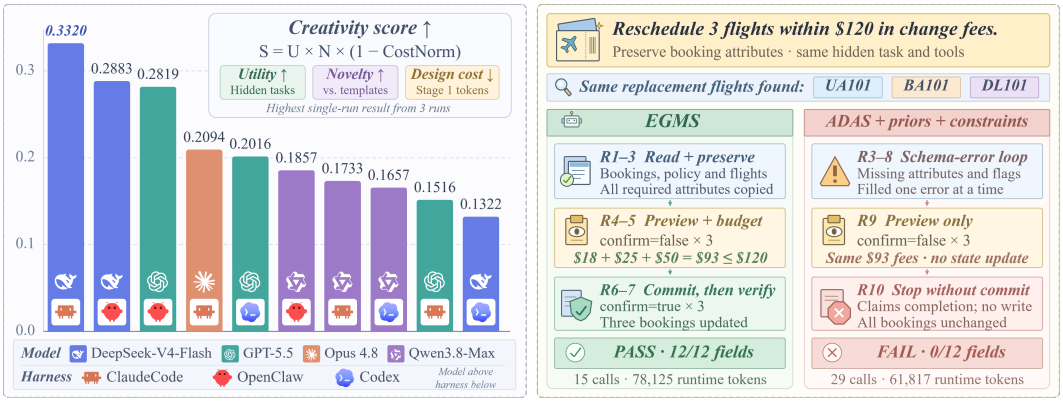


Figure 4: Architectural creativity across ten model–harness combinations. Colors identify models; each row pairs model and harness icons with its reported score. Full metrics appear in Table 5.

Table 1: Main results comparison on Codex with `GPT-5.5`. Scores are reported as the three-run mean with run fluctuation bounds (+upper / −lower across three independent runs, percent). The best mean in each column is bolded and the second best is underlined; ties share the annotation. Cost is the total Stage 1 tokens (millions); Creativity is the CreativityScore defined in Section 3.4.

Method	CS	Shop	Travel	All	Novelty	Cost (M)	Creativity
ADAS	$48.1^{+17.1}_{-17.8}$	$\mathbf{48.9}^{+7.8}_{-5.6}$	$\mathbf{32.0}^{+8.8}_{-15.7}$	$\underline{43.2}^{+10.8}_{-11.0}$	$0.37^{+0.07}_{-0.10}$	136.7	0.143
GEPA	$57.4^{+5.4}_{-6.2}$	$41.7^{+1.7}_{-3.3}$	$25.2^{+7.5}_{-12.9}$	$40.8^{+4.0}_{-5.3}$	$\underline{0.40}^{+0.03}_{-0.03}$	162.3	0.144
Meta-Harness	$57.4^{+3.1}_{-6.2}$	$43.3^{+5.0}_{-5.0}$	$28.6^{+12.3}_{-16.3}$	$42.5^{+2.2}_{-2.4}$	$0.36^{+0.01}_{-0.02}$	134.6	0.137
Better Harnesses	$\underline{59.7}^{+3.1}_{-3.9}$	$42.8^{+8.9}_{-9.5}$	$25.9^{+8.8}_{-9.5}$	$42.1^{+7.2}_{-6.6}$	$\underline{0.40}^{+0.07}_{-0.07}$	129.7	$\underline{0.153}$
AFlow	$58.1^{+7.0}_{-11.6}$	$45.6^{+2.8}_{-2.2}$	$20.4^{+8.2}_{-8.2}$	$41.0^{+0.4}_{-0.9}$	$\underline{0.40}^{+0.20}_{-0.10}$	160.9	0.145
EGMS (Ours)	$\mathbf{74.4}^{+7.0}_{-7.0}$	$\underline{46.7}^{+18.3}_{-15.0}$	$\underline{31.3}^{+9.5}_{-8.8}$	$\mathbf{49.6}^{+4.4}_{-4.2}$	$\mathbf{0.61}^{+0.02}_{-0.01}$	143.7	**0.272**

4.3 Comparison with Architecture-Search Baselines (RQ2)

We fix the base agent (Codex + `GPT-5.5`) for each method. Table 1 compares our method against five established architecture-search baselines under the same budget and protocol. For each method, we run three trials and report its best single-run result.

Obs. 3: Our method attains the best mean utility and the best novelty, and its best runs set the single-run record on two of three domains. EGMS reaches 49.6% mean overall utility, 6.4↑ points above the best baseline mean (ADAS, 43.2%). Its best runs reach 81.4% on customer support and 65.0% on shopping assistant, 16.3↑ and 8.3↑ points above the best baseline runs, and its overall best run (53.9%) ties the best ADAS run. On travel (the hardest domain) our best run (40.8%) ties the best ADAS and Meta-Harness runs. Novelty is unambiguous: our mean of 0.61 and best of 0.63 exceed the best run of every baseline. The composite amplifies the gap: our CreativityScore of 0.272 is 78%↑ above the strongest baseline (Better Harnesses, 0.153).

Obs. 4: Baseline search methods occasionally find strong runs but cannot reproduce them across seeds. The best runs of the baselines span 41.4% ∼ 53.9% overall utility, yet their means stay within 40.8% ∼ 43.2%. Scalar-fitness search can stumble onto a good architecture in this setting, but without causal diagnosis of failures and counterexample-gated acceptance it does so unreliably. Paired significance results are reported in Appendix D.4.

4.4 Ablating the Method Components (RQ3)

Starting from the full method, we remove the trajectory validity constraints, the design-space priors together with the constraints (search procedure only), or the search procedure itself (priors only), keeping the Architect, budget, and protocol fixed. Table 2 reports the results per domain.

Obs. 5: Every component contributes, and the ordering is consistent across both the mean and the best run. Removing any component lowers both the three-run mean and the best single run on overall utility: 49.6% / 53.9% (full method) ∼ 40.8% / 46.1%↓ without trajectory constraints, 34.4% / 49.3%↓ with the search procedure only, and 33.6% / 40.8%↓ without the search procedure. The search procedure is the largest single factor by mean 16.0↓, while the trajectory constraints

Table 2: Component ablation of the method on Codex with `GPT-5.5`. Format matches Table 1: three-run mean with run fluctuation bounds (+upper / −lower); the best mean in each column is bolded and the second best is underlined. Δ is the drop in mean overall utility.

Variant	CS	Shop	Travel	All	Novelty	Cost (M)	Creativity	Δ
EGMS (full)	**74.4**$^{+7.0}_{-7.0}$	**46.7**$^{+18.3}_{-15.0}$	**31.3**$^{+9.5}_{-8.8}$	**49.6**$^{+4.4}_{-4.2}$	**0.61**$^{+0.02}_{-0.01}$	143.7	**0.272**	–
w/o trajectory constraints	<u>48.8</u>$^{+27.9}_{-25.6}$	**46.7**$^{+3.3}_{-3.3}$	26.5$^{+12.3}_{-8.2}$	<u>40.8</u>$^{+5.3}_{-8.6}$	<u>0.56</u>$^{+0.04}_{-0.06}$	192.4	<u>0.196</u>	8.8↓
w/o priors and constraints	33.3$^{+29.5}_{-14.7}$	39.4$^{+8.9}_{-7.8}$	<u>29.3</u>$^{+21.8}_{-17.0}$	34.4$^{+14.9}_{-12.7}$	0.41$^{+0.12}_{-0.18}$	75.3	0.134	15.2↓
w/o search procedure	45.0$^{+13.2}_{-26.4}$	38.9$^{+12.8}_{-7.2}$	17.0$^{+9.5}_{-4.8}$	33.6$^{+7.2}_{-11.8}$	0.46$^{+0.11}_{-0.12}$	68.6	0.146	16.0↓

primarily stabilize the search: removing them widens the run spread on overall utility 8.5 ∼ 13.9 ↑. Higher token costs (69M ∼ 192M) do not translate into higher scores.

Obs. 6: Utility and novelty degrade together across ablation arms. Mean novelty falls monotonically with utility, 0.61 (full method) ∼ 0.56↓ (w/o constraints), 0.46↓ (w/o search procedure), and 0.41↓ (search only). Weaker variants not only work worse but also produce architectures closer to the template library, consistent with the view that the method's value lies in converting development evidence into genuinely different mechanisms rather than into surface variations of known designs.

4.5 The Value of Creating Reusable Agents (RQ4)

We measure whether compiling domain knowledge into a frozen architecture actually saves execution cost, by letting the Codex harness solve the same held-out tasks with runtime skill injection but generated architecture, once with the same local `Qwen3.6-27B` model and once with `GPT-5.5`.

Obs. 7: Compiled architectures solve more tasks at a lower token cost than a monolithic harness. Figure 5 visualizes the three execution trade-offs. Our frozen compiled architectures solve 49.6% of the same tasks at 25.5k tokens per task, that is, 51.4k tokens per solved task: a higher pass rate at about 2.4 ∼ 2.6↓ times lower cost per solved task. The `GPT-5.5` harness's tokens also carry a higher unit price than the local `Qwen3.6-27B` runtime, so a dollar-denominated comparison would widen the gap further. A task-level paired test over the 152 held-out tasks confirms the gap: pairing each of our three runs with the harness run, our architectures pass 79 more task-runs than they lose against the `Qwen3.6-27B` harness (McNemar $p = 8.1 \times 10^{-11}$) and 77 more against the `GPT-5.5` harness ($p = 4.9 \times 10^{-10}$). This supports the compiled-versus-interpreted claim of Section 1: knowledge embedded at design time does not need to be re-derived. The measured token reduction covers the complete execution, including planning, tool feedback, and any runtime instructions. It therefore supports the efficiency of the compiled architecture as a whole, rather than isolating savings due to skill loading alone.

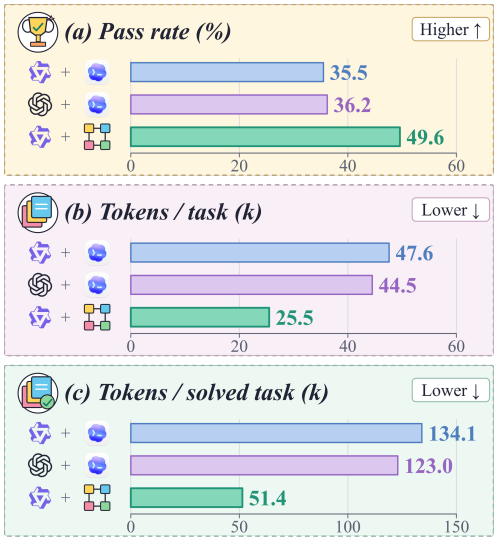


Figure 5: Execution trade-offs. Tokens cover total runtime usage.

5 Conclusion

We introduced ACA to evaluate the creativity of agents from an architectural perspective, together with EGMS, which guides architecture design through the causal structure of failures. The process that ACA evaluates is itself self-improvement, since an agent examines evidence of its own failures, creates an improved architecture, and is assessed on unseen tasks. Within this process, EGMS shows that grounding each self-modification in the causal structure of failures makes recursive selfprovement more effective and raises the creativity of the systems it produces. As frontier models move from generating creative content to contributing to scientific discovery, such self-improving creation of effective systems will become central. We hope ACA helps the community study creativity in self-improving with the same rigor that it applies to the systems these agents produce.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the writing of early drafts (grammar, phrasing, and clarity). We have not used generative AI tools to generate research ideas, to design the method, to produce or analyze experimental results, or to author the experimental code that produced the reported numbers. The core technical contributions, experimental design, analysis, and conclusions were developed entirely by the authors, and we take responsibility for the final content, including all text, scientific claims, and accompanying research artifacts.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Qian Cao, Xiting Wang, Yuzhuo Yuan, Yahui Liu, Fang Luo, and Ruihua Song. Evaluating text creativity across diverse domains: A dataset and large language model evaluator. In *International Conference on Learning Representations*, volume 2026, pp. 33245–33283, 2026.

Tuhin Chakrabarty, Philippe Laban, Divyansh Agarwal, Smaranda Muresan, and Chien-Sheng Wu. Art or artifice? large language models and the false promise of creativity. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–34, 2024.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. Workarena: How capable are web agents at solving common knowledge work tasks? *arXiv preprint arXiv:2403.07718*, 2024.

Batu El, Mert Yuksekgonul, and James Zou. Inefficiencies of meta agents for agent design. *arXiv preprint arXiv:2510.06711*, 2025.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. Skillaxe: Sharpening llm-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026.

Sina Hajimiri, Masih Aminbeidokhti, Jose Dolz, Ismail Ben Ayed, Issam H Laradji, Spandana Gella, and Nicolas Gontier. Are online skill and memory modules always worth their tokens? a budget-constrained study of web agents. *arXiv preprint arXiv:2606.15017*, 2026.

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.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025.

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

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

Junlong Li, Wenshuo Zhao, Jian Zhao, Weihao Zeng, Haoze Wu, Xiaochen Wang, Rui Ge, Yuxuan Cao, Yuzhen Huang, Wei Liu, et al. The tool decathlon: Benchmarking language agents for diverse, realistic, and long-horizon task execution. In *International Conference on Learning Representations*, volume 2026, pp. 53359–53394, 2026a.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026b.

Zijun Liu, Yanzhe Zhang, Peng Li, Yang Liu, and Diyi Yang. A dynamic LLM-powered agent network for task-oriented agent collaboration. In *First Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=XII0Wp1XA9.

Ryan Lopopolo. Harness engineering: How to build software when humans steer, agents execute. *OpenAI Engineering Blog*, 2026.

Ximing Lu, Melanie Sclar, Skyler Hallinan, Niloofar Mireshghallah, Jiacheng Liu, Seungju Han, Allyson Ettinger, Liwei Jiang, Khyathi Chandu, Nouha Dziri, and Yejin Choi. Ai as humanity's salieri: Quantifying linguistic creativity of language models via systematic attribution of machine text against web text, 2025. URL https://arxiv.org/abs/2410.04265.

Ruiying Ma, Chieh-Jan Mike Liang, Yanjie Gao, and Francis Y. Yan. Metamuse: Algorithm generation via creative ideation, 2026. URL https://arxiv.org/abs/2510.03851.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants, 2023. *URL https://arxiv.org/abs/2311.12983*, 2311, 2023.

Microsoft. STATE-Bench: Stateful task agent evaluation benchmark. https://github.com/microsoft/STATE-Bench, 2026. Open-source benchmark of 450 enterprise tasks across travel, customer support, and shopping assistant domains.

Stefan Nielsen, Edoardo Cetin, Peter Schwendeman, Qi Sun, Jinglue Xu, and Yujin Tang. Learning to orchestrate agents in natural language with the conductor. In *International Conference on Learning Representations*, volume 2026, pp. 135686–135724, 2026.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Shuofei Qiao, Runnan Fang, Zhisong Qiu, Xiaobin Wang, Ningyu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Benchmarking agentic workflow generation, 2025. URL https://arxiv.org/abs/2410.07869.

Stephan Rabanser, Sayash Kapoor, Peter Kirgis, Kangheng Liu, Saiteja Utpala, and Arvind Narayanan. Towards a science of ai agent reliability. *arXiv preprint arXiv:2602.16666*, 2026.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pp. 3841–3865, 2025.

Chenglei Si, Diyi Yang, and Tatsunori Hashimoto. Can llms generate novel research ideas? a large-scale human study with 100+ nlp researchers. In *International Conference on Learning Representations*, volume 2025, pp. 94003–94092, 2025.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Hu Wei. Architectural design decisions in ai agent harnesses. *arXiv preprint arXiv:2604.18071*, 2026.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Frank Fangzheng Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, et al. Theagentcompany: benchmarking llm agents on consequential real world tasks. *Advances in Neural Information Processing Systems*, 38, 2026.

Chenyang Yang, Xinran Zhao, Tongshuang Wu, and Christian Kästner. Better harnesses, smaller models: Building 90% cheaper agents via automated harness adaptation. *arXiv preprint arXiv:2607.08938*, 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Justin Young. Effective harnesses for long-running agents. *Anthropic Engineering Blog, Nov*, 2025.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, Lei Bai, and Xiang Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. Aflow: Automating agentic workflow generation. In *International Conference on Learning Representations*, volume 2025, pp. 34040–34077, 2025b.

Yiming Zhang, Harshita Diddee, Susan Holm, Hanchen Liu, Xinyue Liu, Vinay Samuel, Barry Wang, and Daphne Ippolito. Noveltybench: Evaluating creativity and diversity in language models. *arXiv preprint arXiv:2504.05228*, 2025c.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, et al. Skillflow: Benchmarking lifelong skill discovery and evolution for autonomous agents. *arXiv preprint arXiv:2604.17308*, 2026.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo FR Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. Skilllearnbench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, volume 2026, pp. 15844–15872, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

Appendix roadmap. Construction (A); evaluation (B); method and baseline details (C); additional experiments (D); case studies (E); related work (F); limitations (G).

A Benchmark Construction

A.1 Task Adaptation and Development–Hidden Splits

We build on STATE-Bench (Microsoft, 2026) and adapt its three domains (customer support, shopping assistant, travel) into a deterministic evaluation package: each task is repackaged into explicit Stage 1 inputs and sanitized Stage 2 public inputs containing only the opening message, while the raw environment and the ground-truth state requirements are held privately by the harness in oracle sidecars. From the 82, 86, and 70 adapted tasks per domain, we exclude tasks whose requirements force asking or confirming with the user before acting (19, 6, and 1 tasks respectively), because the single-attempt state-only protocol cannot complete them, leaving 63, 80, and 69 tasks. Twenty development tasks per domain are drawn by stratified sampling over task types (seed 42) and are visible to the Architect; the remaining 43, 60, and 49 tasks form the held-out set and are invisible during architecture design. This is an internal held-out split, not the official leaderboard protocol.

A.2 Template Library Construction and Coverage

We construct the frozen reference before Stage 1 evaluation and keep its contents fixed for every compared system. The library has three explicitly separated strata. The 50 *real* templates are grounded in named, publicly documented agent systems or framework patterns; each manifest records its provenance and the mechanism mapping used to express that system in the ACA schema. The 40 *baseline* templates are compact, manually authored architecture patterns that provide coverage of the schema vocabulary and common search baselines. The remaining 10 *partial-fidelity* templates represent named mechanisms whose full semantics cannot be enforced by the current manifest schema; each is marked with a fidelity note that distinguishes the executable schema-supported part from narrative-only claims.

For each template we normalize the observable fields (roles, control flow, communication, memory, tool policy, safety policy, and budget policy), validate the manifest against the library schema, and retain one canonical manifest per included pattern. As a final de-duplication audit, we compute the validator's declared behavioral hash over the active snapshot; the frozen library has no duplicate behavioral-hash groups. The active roots are loaded in the fixed order real, baseline, and partial-fidelity, with filenames sorted within each root. The resulting snapshot contains exactly 50, 40, and 10 active templates, respectively; its index and file hashes are frozen in the released protocol directory. This organization is intended to provide provenance diversity and mechanism coverage without allowing archived or schema-invalid artifacts to enter the novelty reference.

A.2.1 Qualitative Candidate–Template Comparisons

Table 3 gives two representative comparisons. Scores are assigned dimension by dimension using the policy and trace evidence described below; a score of zero is retained when a declared mechanism is not independently observable in execution.

A.3 Corpus Construction and Evaluation Inventory

The corpus represents several distinct construction steps. Adapting 238 tasks precedes the protocol-based exclusion of 26 interactive tasks. The remaining 212 tasks split into 60 development and 152 hidden tasks. Three independent architecture searches therefore entail 456 hidden task executions per complete system, while the structural novelty assessment uses a separate frozen library of 100 references. These counts describe different units of work: the number of task executions is not an expanded dataset size. In panel (a), bar segments and adjacent counts follow development / hidden / excluded order; segment widths encode task counts.

B Evaluation and Novelty Judging

Candidate	Nearest template (total)	Observable mechanism comparison (roles/control/communication/tool/memory)
EGMS, travel, rep. 2	Baseline Replan-on-Failure (7/10)	Separates context normalization, planning, preaction guarding, and review from the template's executor/monitor/replanner roles (2); adds validation-triggered repair, safe fallback, preview gating, and evidence-based termination (2); communication remains a sequential state-delta loop (0); rejects stale, replayed, duplicate, or ungrounded calls and mirrors successful previews (1); maintains canonical observations, completed mutations, failures, goal hints, and completion evidence (2).
EGMS, shopping, rep. 1	ReAct Tool-Observation (2/10)	The observable role, control, and communication regime remains equivalent to the ReAct loop (0/0/0); bounded validated batch dispatch changes tool behavior (1); a recency-aware ledger records latest observations and completed mutations (1).

Table 3: Representative candidate–nearest-template comparisons from the frozen-trace protocol. The first example shows a substantive multi-dimensional departure; the second illustrates why declared labels alone do not count when they have no independent downstream effect.

B.1 Novelty Judging Protocol

Judge and inputs. Novelty is assigned by an agent-as-judge (GPT-5.5, locked configuration). For each candidate, the judge receives the architecture manifest, the runner adapter, the Stage 2 execution traces, and the ten most structurally similar templates retrieved from the frozen library of 100 architecture templates (50 real, 40 baseline, and 10 partial-fidelity). The judge never receives hidden or gold information, scorer internals, or downstream utility.

Evidence layers. Every dimension is scored from two evidence layers: policy evidence, covering mechanisms declared in the manifest and adapter, and trace evidence, covering mechanisms actually activated during execution. A difference counts only when it has a trigger condition, an action or output, and a downstream effect; renamings, prompt wording, empty nodes, and untriggered mechanisms score zero.

Rubric. Each of the five dimensions is scored 0, 1, or 2: 0 for no substantive difference, 1 for one clear mechanism difference with observable effect, and 2 for multiple mechanism differences or a clearly changed core mechanism.

Table 4: Five dimensions of template-relative novelty. Each dimension is scored from policy and execution evidence.

Dimension	Core question
Roles	Do role responsibilities, decision rights, or division of labor change?
Control edges	Do branching, looping, routing, recovery, or termination change?
Communication edges	Do message flows, aggregation, review, or debate paths change?
Tool edges	Do tool binding, routing, validation, recovery, or permissions change?
Memory edges	Do state or memory read, write, compression, retrieval, or update rules change?

Nearest-template score. Let $R_{10}(C, L)$ be the ten most structurally similar templates retrieved for candidate C from the frozen library L. The novelty of a candidate is its distance to the nearest retrieved template,

$$\text{Novelty}(C) = \min_{T \in R_{10}(C,L)} \frac{D_{\text{roles}} + D_{\text{control}} + D_{\text{comm}} + D_{\text{tool}} + D_{\text{memory}}}{10}, \tag{11}$$

so a candidate that reimplements any single retrieved template scores zero regardless of its other elaborations. Retrieval is part of the frozen scoring protocol; full-library retrieval recall and sensitivity are reported separately rather than conflated with the judge score.

B.2 Benchmark Evaluation Details

This section records the detailed measurement conventions summarized in Section 3.4.

B.2.1 Metric reporting and the common reference

The metric definitions follow Section 3.4, and the judging rubric follows Appendix B.1. The RQ1, baseline, and ablation result tables report utility, novelty, design cost, and the composite together. The fixed design-budget reference was established from the earlier campaign and is not recalibrated to the updated leaderboard. Reported design costs differ substantially across combinations (Table 5), making component-wise reporting important for interpreting the composite.

B.2.2 Token accounting and the fixed reference

The symbols C in the experimental tables and C_{design} in the method refer to Stage 1 token consumption. Both input and output tokens are counted over architecture creation, development evaluation, and repair. For the logged Stage 1 usage events $\mathcal{E}_{\text{design}}$, this accounting is

$$C_{\text{design}} = \sum_{e \in \mathcal{E}_{\text{design}}} \left(c_e^{\text{input}} + c_e^{\text{output}} \right), \qquad C_{\text{ref}} = 1.408 \times 10^9 \text{ tokens}. \tag{12}$$

The reference is fixed across the reported comparisons rather than recomputed from each table. Its interpretation is discussed in Appendix D.3. We use $\widetilde{C}$ and $\mathrm{CostNorm}$ interchangeably, and S denotes $\mathrm{CreativityScore}$.

B.2.3 Execution cost and reporting

For a fixed evaluated task set, let c_x be the complete Stage 2 token usage of task x, including unsuccessful attempts, and $y_x = U(\pi, x)$. Tokens per task and per solved task use different denominators:

$$\text{Tokens/task} = \frac{\sum_x c_x}{n}, \qquad \text{Tokens/solved} = \frac{\sum_x c_x}{\sum_x y_x}, \qquad n = |\mathcal{D}_{\text{hid}}|. \tag{13}$$

The second quantity is defined when at least one task is solved. These are total runtime costs, including planning, tool feedback, and runtime instructions, rather than an isolated measurement of skill loading. The RQ4 harness runs in Section 4.5 are exploratory single-run measurements using adapter-reported usage. These baselines do not provide replicate-based uncertainty estimates. Architecture search is accounted for separately, so the runtime comparison does not establish a deployment break-even point.

The baseline and ablation comparisons report three-run means and run fluctuation bounds; the RQ1 leaderboard reports the highest single-run result from three runs and evaluations. Every metric component is reported alongside the composite, so a change in usefulness, judged novelty, or resource use remains visible. Novelty is relative to the retrieved reference set and the fixed rubric; it is not a claim of originality against all possible architectures.

RQ1 maxima and the table-specific summaries of RQ2/RQ3 are not pooled into one ranking. The RQ1 plots reproduce the supplied cost and waste columns separately rather than summing them or inferring additional accounting quantities.

C Method and Baseline Details

C.1 Trajectory Validity Constraints

Beyond structural priors, EGMS imposes validity constraints on execution trajectories. Let $C(\tau)$ denote the conjunction of the following constraints; an action violating C is rejected by the executor's validation layer rather than executed. Every write is scoped to the exact resolved item subset, previewed, and confirmed by mirroring the latest successful preview arguments; a rejected preview is never confirmed, and a preview is invalidated after a relevant mutation. Amounts are taken from authoritative previews when available, and otherwise computed by applying the fetched policy formula to observed inputs. Successful reads and policies are cached, independent reads are batched, and an identical failed call is never repeated. A turn never ends silently: it either executes a validated action, requests information that is genuinely missing, or presents an evidence-backed conclusion in which unresolved goals remain explicitly visible.

C.2 Baseline Adaptation Details

Each baseline search method is adapted into a Stage 1 guidance document injected into the same Codex with GPT-5.5 Architect, under a uniform budget of one initial generation plus up to five repairs. The adaptations preserve each method's search strategy but replace its original execution machinery with the Architect's own loop, so the reported numbers should be read as lower bounds of the original pipelines.

- **ADAS** (Hu et al., 2025): archive-style search. The guidance asks the Architect to read each candidate's code, scores, cost, and design summary, extract shared mechanisms of high scorers and recurring failure patterns of low scorers, and propose new design hypotheses from reference parents. The archive is realized by the Architect's search history rather than by the original ADAS execution loop.
- **AFlow** (Zhang et al., 2025b): tree-search workflow optimization. The guidance represents the candidate as editable workflow code and expands a search tree using scores, failure types, structural novelty, and exploration counts; the Monte Carlo tree search machinery is replaced by its selection policy as instructions.
- **GEPA** (Agrawal et al., 2026): reflective evolution. The guidance collects complete task trajectories, extracts reusable mechanisms from successful ones, and revises the candidate by reflective diagnosis.
- **Meta-Harness** (Lee et al., 2026): full-history search. The guidance preserves each candidate's complete code, configuration, evaluation, and representative trajectories, and searches structural patterns over the full history.
- **Better Harnesses** (Yang et al., 2026): failure-driven modification. The guidance aggregates failed tasks, decomposes them into reproducible failure modes, locates the earliest critical deviation and its layer, and modifies the corresponding mechanism.

D Additional Experimental Analyses

D.1 Updated Architect Leaderboard

Table 5 records the ten reported results in Figure 4. Each combination is run and evaluated three times, and its highest single-run result is reported. Utility, novelty, cost, and creativity describe that reported result. These maxima characterize attained performance; they do not estimate average performance or stability across runs. Cost and waste remain separate reported token quantities. Creativity values are reproduced directly, without recomputation from rounded component values.

Table 5: Highest single-run model–harness results from three runs, ordered by CreativityScore. Cost and waste are separate reported token quantities (M); utility is expressed as a percentage.

Rank	Model	Harness	U (%)	N	Cost	Waste	Creativity
1	DeepSeek-V4-Flash	ClaudeCode	**54.61**	**0.633**	56.480	0.000	**0.3320**
2	DeepSeek-V4-Flash	OpenClaw	53.95	0.567	79.977	22.878	0.2883
3	GPT-5.5	OpenClaw	50.00	0.567	7.139	0.220	0.2819
4	Opus 4.8	ClaudeCode	36.84	0.600	73.986	77.270	0.2094
5	GPT-5.5	Codex	50.66	0.400	7.086	0.000	0.2016
6	Qwen3.8-Max	OpenClaw	40.79	0.467	34.228	0.000	0.1857
7	Qwen3.8-Max	ClaudeCode	40.13	0.433	4.485	2.425	0.1733
8	Qwen3.8-Max	Codex	29.61	0.567	17.141	1.812	0.1657
9	GPT-5.5	ClaudeCode	41.45	0.367	3.194	2.804	0.1516
10	DeepSeek-V4-Flash	Codex	36.18	0.367	5.402	0.000	0.1322

D.2 Model–Harness Trade-offs and Coverage

The RQ1 panels below use exactly the ten reported outcomes in Figure 4 and Table 5. They include all three Qwen3.8-Max harness combinations. The analyses concern model–harness aggregates; they do not infer domain-level scores, development–held-out gaps, or run-to-run variability from these maxima.

Component trade-offs. DeepSeek-V4-Flash with ClaudeCode combines the highest utility (54.61%) and novelty (0.633), yielding the highest creativity (0.3320). Opus 4.8 with ClaudeCode

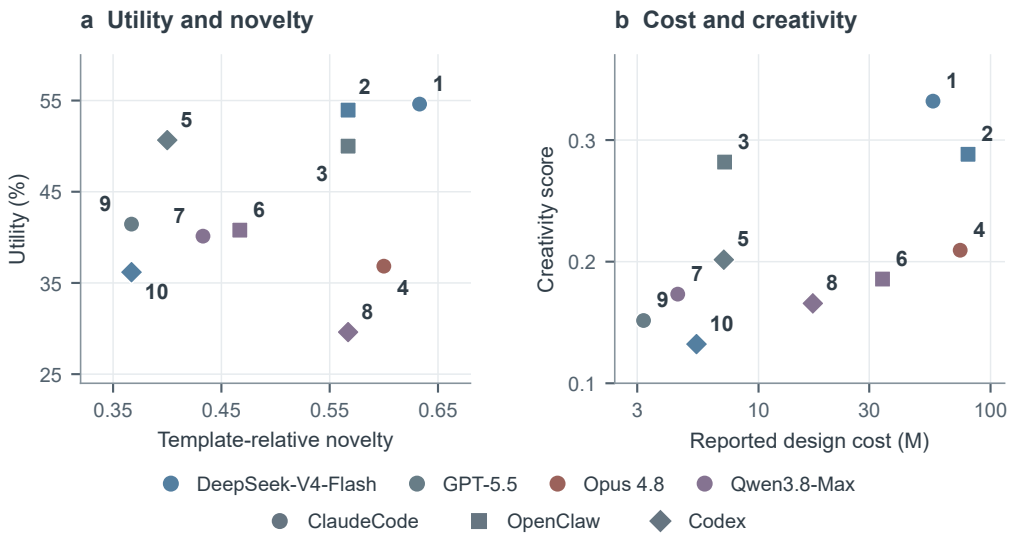


Figure 6: Utility–novelty and cost–creativity trade-offs for all ten RQ1 outcomes. Numbers identify ranks in Table 5; colors denote models and shapes denote harnesses. The cost axis is logarithmic.

has novelty 0.600 but utility 36.84%, illustrating why structural difference alone is insufficient. Reported design cost also varies substantially: GPT-5.5 with OpenClaw reaches creativity 0.2819 at 7.139M, close to DeepSeek-V4-Flash with OpenClaw at 0.2883 and 79.977M. These observations describe the displayed outcomes, without attributing the gaps to a particular harness mechanism.

D.3 Interpreting the Composite Score

The RQ1 score combines usefulness, novelty, and the reported design cost, but these components should also be inspected separately. DeepSeek-V4-Flash with ClaudeCode leads both utility and novelty in Table 5. For GPT-5.5, the utility ordering differs from the creativity ordering: Codex has slightly higher utility, whereas OpenClaw has higher novelty and creativity. This comparison supports joint evaluation of the components; it does not establish that the cost term reverses the ranking.

The fixed reference C_{ref} determines the strength of the cost penalty. A common reference alone does not make results with different reporting statistics directly comparable. Figure 4 presents highest single-run results, while the baseline and ablation comparisons retain their own table-specific reporting conventions. We therefore compare rankings within each experiment and do not combine their points into a shared cost–utility frontier. No ranking-invariance threshold is inferred from the ten reported RQ1 outcomes.

D.4 Statistical Significance

A task-level paired analysis across all 456 task-runs (McNemar's test with continuity correction, paired bootstrap with 10,000 resamples) verifies the performance gap is not random run noise. Our method outperforms all five baselines ($p < 0.01$ for each comparison), with all bootstrap 95% confidence intervals of the mean difference excluding zero: e.g., 6.4↑ points (CI $[+2.2, +10.7]$) over ADAS and 8.8↑ points (CI $[+4.8, +12.7]$) over GEPA. The ablation analysis yields $p < 10^{-4}$ for all three variants. This paired design eliminates task-difficulty variance, the dominant noise source in Tables 1 and 2.

D.5 Factorial Attribution of the Search Procedure

Table 1 compares the full EGMS configuration against five search baselines, but the full configuration bundles three components: the search procedure, the design-space priors, and the trajectory constraints. A natural concern is that the priors and constraints, not the search procedure, drive the gains; indeed, the EGMS search-procedure-only arm of Table 2 (34.4%) sits below the baseline means in Table 1. We therefore run a 2×2 factorial over *search procedure* $\times$ *priors and constraints* using the strongest baseline (ADAS by Table 1 mean): EGMS full (Table 1 row), EGMS search only (Table 2 arm A3), ADAS as reported in Table 1, and ADAS paired with the verbatim design-space

priors and trajectory constraints sections of the EGMS guidance, run under an identical budget and protocol (3 domains $\times$ 3 independent searches; the ADAS+priors+constraints cells additionally run at the higher reasoning-effort setting used by EGMS, which is favorable to the baseline).

Table 6: Factorial attribution of search procedure and priors/constraints. Cells show three-run mean held-out utility with per-run values; the interaction favors the combined EGMS configuration.

	priors + constraints off	priors + constraints on
EGMS search	34.4 (32.2 / 21.7 / 49.3)	**49.6** (53.9 / 49.3 / 45.4)
ADAS search	43.2 (53.9 / 32.2 / 43.4)	37.3 (42.8 / 31.6 / 37.5)

Additional details for Table 6. Factorial attribution: held-out utility by search procedure and priors/constraints (mean over 3 runs $\times$ 3 domains; per-rep values in parentheses). The priors and constraints lift EGMS by 15.2$\uparrow$ points but reduce ADAS by 5.9$\downarrow$ points; under identical priors and constraints, the EGMS search procedure exceeds the ADAS search procedure by 12.8$\uparrow$ points (cluster bootstrap 95% CI $[+5.7, +20.7]$, Wilcoxon $p = 0.0078$ over 9 architecture-search clusters). The interaction is a crossover: the same priors and constraints that add 15.2$\uparrow$ points to EGMS *subtract* 5.9$\downarrow$ points from ADAS. The priors are therefore not a free-standing advantage that any search procedure could exploit; they define a mechanism space, and it is EGMS's causal-diagnosis and counterexample-gated repair loop that converts mechanisms in that space into held-out gains. Conversely, without the priors the ADAS search exceeds the EGMS search (43.2 vs 34.4), so we do not claim that the search procedure alone is superior in all conditions; the claim supported by Table 6 is that the EGMS configuration is the jointly designed combination, and that its advantage over the strongest baseline survives giving that baseline the same priors and constraints with a favorable compute setting.

D.6 Search, Ablation, and Execution Diagnostics

The following panels retain the existing baseline, ablation, and trajectory results. Their reporting units are distinct from the highest single-run RQ1 outcomes.

Reading the panels. Panel (a) contains only the ten RQ1 combinations. Panels (b,c) retain the original baseline and ablation plots: EGMS has the highest customer-support utility among the compared methods, while ADAS leads the reported shopping and travel columns. Each ablated configuration has lower utility, novelty, and creativity than full EGMS; panel (c) normalizes each metric to its own full-EGMS value. TC denotes trajectory constraints and P denotes design-space priors. These panels are not used to rank RQ1 configurations against EGMS across different experiments.

Development trajectories. Figure 8 shows nine EGMS searches: three each for customer support, shopping, and travel. Rows identify domains; the two columns show development checkpoints against repair iteration and recorded design tokens. Checkpoints preserve the best development result seen so far, so monotonicity reflects checkpoint selection rather than uninterrupted improvement of every candidate. Progress and token consumption vary across runs. Customer-support final checkpoints are 19/20, 17/20, and 19/20. Some token ledgers are incomplete, so the displayed cumulative costs are lower bounds where indicated. These EGMS trajectories do not measure the generalization gap of the Fig. 4 systems.

Observable execution signals. Figure 9 covers 129 customer-support, 180 shopping, and 147 travel task-runs. Ordinary signal rates use all task-runs in the corresponding domain. Preview rates use the subset invoking mutation-capable APIs: 105/108 for customer support and 62/70 for travel. Shopping has no preview handshake and is marked N/A. Observing a preview does not establish correct preview-before-commit ordering for every write. Likewise, an error or replay signal alone does not establish successful recovery or an enforced veto. The figure records observable events, complementing the novelty rubric without equating declared mechanisms with successful execution.

Design and execution costs. Stage 1 design cost and Stage 2 runtime cost answer different questions. The existing execution comparison reports 25.5k tokens per task for compiled architectures versus 44.5k and 47.6k for the monolithic harnesses. Figure 10 retains this runtime comparison separately from the design-cost score. It supports an execution-cost advantage in the evaluated setting, without isolating skill-loading tokens or establishing a deployment break-even point.

Interaction between search and guidance. The factorial results retain their original scope: adding priors and constraints improves the EGMS search procedure by 15.2$\uparrow$ utility points, while the corresponding ADAS comparison decreases by 5.9$\downarrow$ points. Under matched priors and constraints, the reported EGMS–ADAS difference is 12.8$\uparrow$ points (cluster-bootstrap 95% CI $[+5.7, +20.7]$;

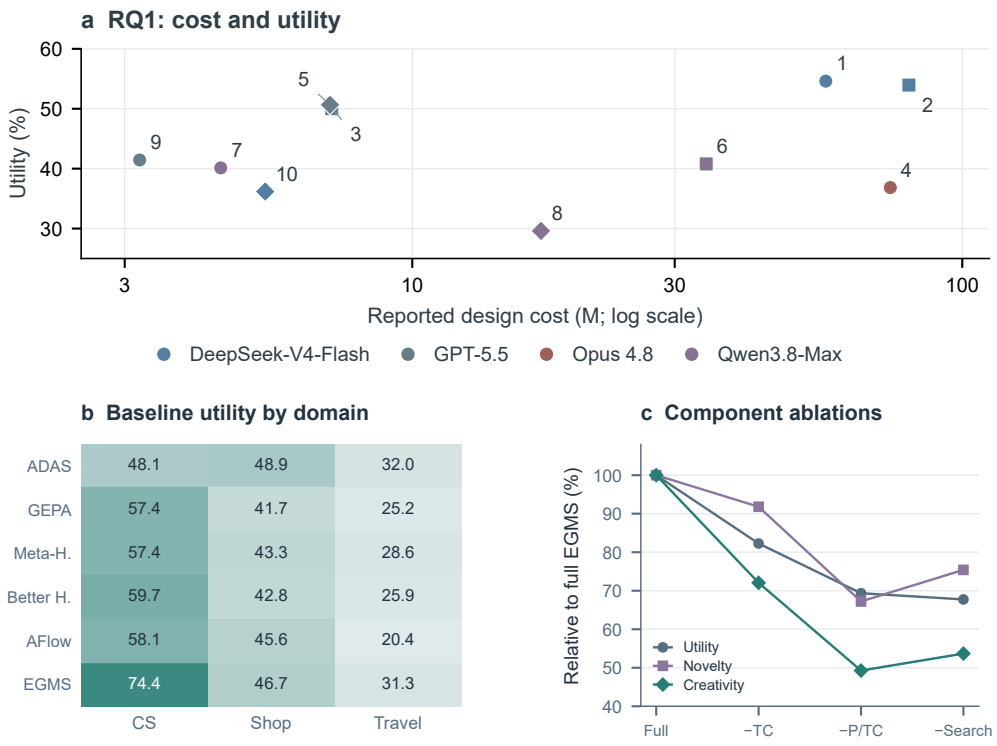


Figure 7: RQ1 cost–utility outcomes (a; numbers identify Table 5 ranks), with original baseline utility (b) and ablations (c). Comparisons are made within each experiment.

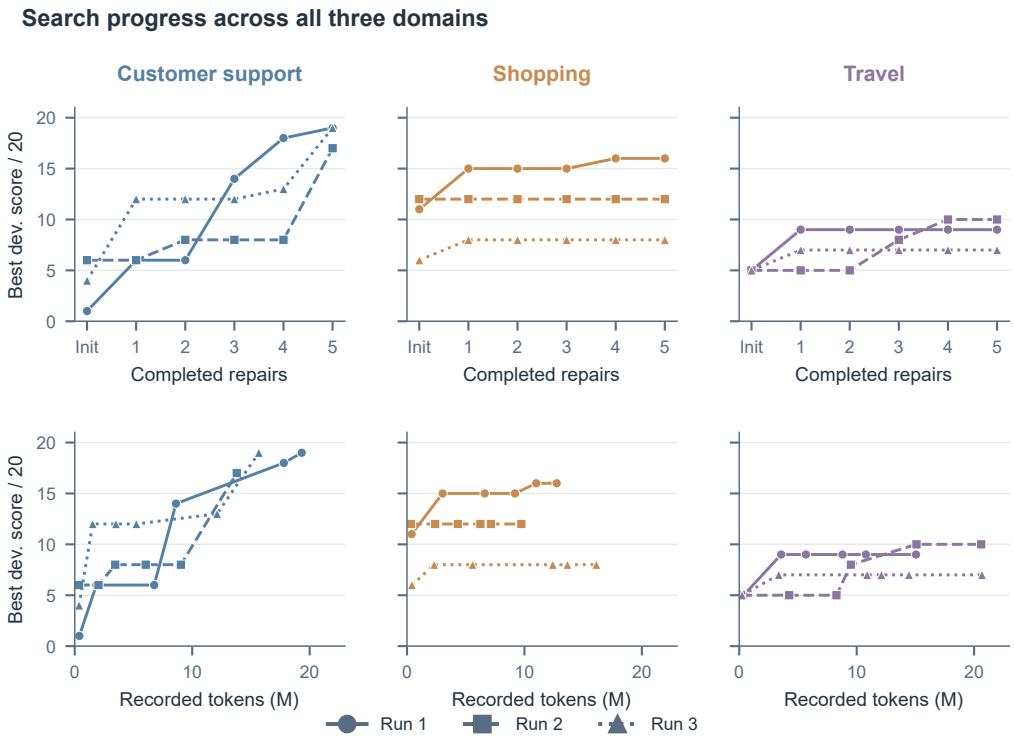


Figure 8: Nine EGMS searches across three domains: best development checkpoints by repair iteration (top) and recorded tokens (bottom). Incomplete token ledgers give lower-bound costs.

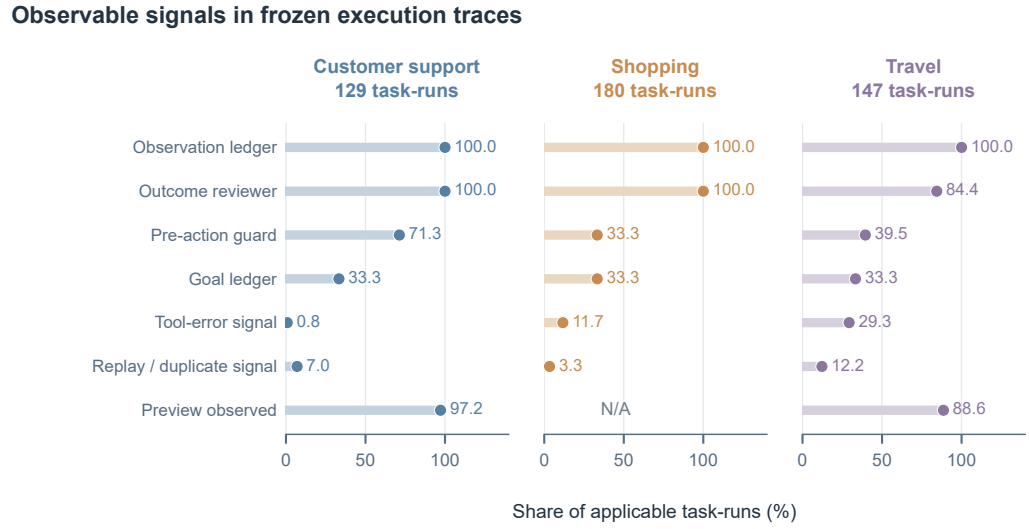


Figure 9: Trace signals across all three domains (456 task-runs). Preview denominators are 108 for customer support and 70 for travel; shopping has no preview handshake (N/A).

Wilcoxon $p = 0.0078$ across nine architecture-search clusters). These results concern the search configurations in Table 6, not the model–harness ranking in Figure 4.

E Extended Case Study

This appendix examines three individual held-out episodes in detail. The first two are successes of the full EGMS configuration and are chosen because the architectural mechanisms of Section C.1

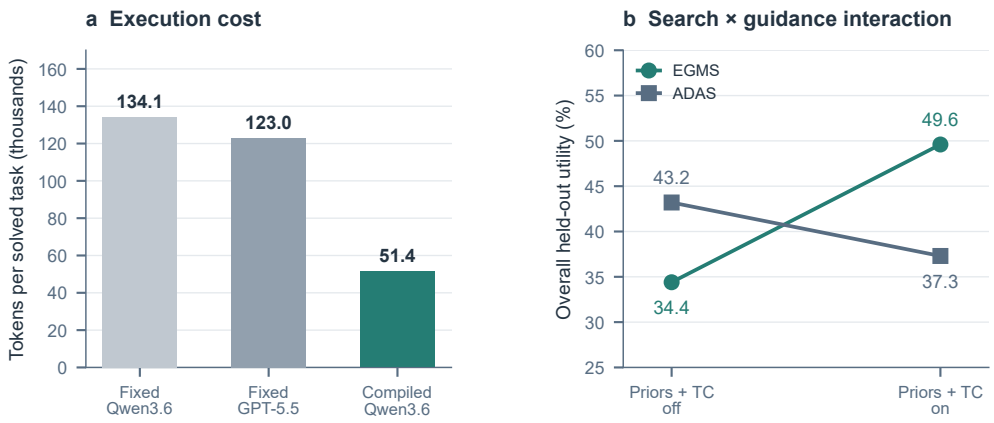


Figure 10: Execution tokens per solved task (exploratory single run) and the search–guidance interaction (three-run means). Priors and constraints improve EGMS but reduce ADAS utility.

are directly visible in the execution trace rather than merely declared in the manifest. The third is a failure of the strongest scalar-fitness baseline (ADAS search under the verbatim EGMS priors and constraints, Section D.5) on *the same task instance*, with the same tool schemas and the same budget, which isolates the effect of the search procedure from the effect of the task.
All quotations are copied verbatim from the frozen Stage 2 traces. Tool arguments and results are shown in the shortened form produced by the harness trace exporter; `...` marks elided arguments. Highlighting is ours and marks the decision point that separates the two systems.

E.1 Positive Example 1: Three Simultaneous Changes Under a Fee Budget

Trace alignment. The archived travel checkpoint for this run passes **9/20** development tasks; **16/20** is the shopping checkpoint from the same run. The travel episode contains **7 rounds and 15 tool calls**, including a final round with no calls. Its opening request and executed tool arguments specify ORD–SFO on June 27, JFK–LHR on June 30, and JFK–CDG on June 23. The London/June 24 tuple retained in the narrative below is not the task specification and should not be used to reconstruct the episode.

Instance and demand. Held-out travel task `55-challenge_triple_fee_stack`, which no system observes during architecture design. The opening message is

```
I have a family bereavement and need to reschedule 3 flights.  Booking
BK-1000 (ORD → SFO) to June 27, booking BK-1001 (JFK → LHR) to
June 30, and booking BK-1002 (JFK → CDG) to June 23.  My user ID
is user_004.  My total budget for change fees is $120 --- can all 3
changes fit?
```

The task is hard for three independent reasons: three mutations must be committed in a single attempt, each mutation must preserve three pre-existing booking attributes (`cabin_class`, `seat_type`, `meal_preference`) plus three optional flags, and the fee arithmetic must be derived from a policy document rather than guessed. Scoring requires 12 environment fields across the three bookings. The frozen architecture was created in run `r1` of the EGMS campaign; its Stage 1 repair ledger records a champion at 16/20 development tasks, and it reaches this hidden task in 7 tool rounds.
What the architecture did. Figure 11(a) shows the same trajectory in compressed form. The execution trace contains the following rounds.

Round 1 — batch read; the goal ledger is created.

```
CALL get_booking {"booking_id": "BK-1000"}
```

```
CALL get_booking {"booking_id": "BK-1001"}
CALL get_booking {"booking_id": "BK-1002"}
RES  BK-1000: UA100 ORD->SFO confirmed, economy/window/standard, price
↪  300
RES  BK-1001: BA100 JFK->LHR confirmed, economy/window/standard, price
↪  700
RES  BK-1002: DL100 JFK->CDG confirmed, economy/window/standard, price
↪  650
```

Three independent reads are dispatched in one batch, which is the batching rule of Section C.1. The planner's goal ledger already separates the three bookings and carries a reason attribute per booking, for example `(scope=user_004, target=BK-1000, transition=change_flight, constraints={destination: London, date: 2026-06-24, reason: bereavement})`.

Rounds 2–3 — evidence gathering, still read-only.

```
CALL get_policies {"topic": "change", "route_type": "domestic",
↪  "cabin_class": "economy"}
CALL get_policies {"topic": "change", "route_type": "international",
↪  "cabin_class": "economy"}
CALL search_flights {"origin": "ORD", "destination": "SFO", "date":
↪  "2026-06-27"}
CALL search_flights {"origin": "JFK", "destination": "LHR", "date":
↪  "2026-06-30"}
CALL search_flights {"origin": "JFK", "destination": "CDG", "date":
↪  "2026-06-23"}
RES  economy_domestic_personal  = $75 if departure > 7 days, $150
↪  otherwise
RES  economy_international_pers = $100 if departure > 7 days, $200
↪  otherwise
RES  bereavement = 75% discount on the standard fee, rounded down
RES  UA101 / BA101 / DL101 are the only non-stop options on the
↪  requested dates
```

The two searches for the outbound and return legs are issued together, while the third is deferred to the next round because its arguments cannot be fixed before the first two return.

Round 4 — preview before any write.

```
CALL update_booking {"booking_id": "BK-1000", "flight_id": "UA101",
                     "change_reason": "bereavement", "confirm": false,
                     "cabin_class": "economy", "seat_type": "window",
                     "meal_preference": "standard", "add_wifi": false,
                     "add_extra_legroom": false, "add_insurance":
                     ↪  false, ...}
                     (identical shape for BK-1001 and BK-1002)
RES  preview: BK-1000 change_fee=18 fare_difference=20 price_paid=338
RES  preview: BK-1001 change_fee=25 fare_difference=20 price_paid=745
RES  preview: BK-1002 change_fee=50 fare_difference=30 price_paid=730
```

Every optional booking attribute is copied from the live `get_booking` result of round 1 instead of being defaulted, which is what makes the preview succeed on the first attempt. The fees are not computed by the model: the authoritative preview returns them, and the model's only arithmetic job is to add them up.

Round 5 — the budget question is answered before the budget is spent.

```
CALL get_user_details {"user_id": "user_004"}
RES  {"user_id": "user_004", "loyalty_tier": "silver",
↪  "loyalty_points": 15000}
```

The plan states the verdict explicitly: *"BK-1000 change fee is $18, BK-1001 change fee is $25, and BK-1002 change fee is $50. The total change fees are $93, which fits within your $120 budget."* This is the only place where the user's budget constraint is resolved, and it is resolved against preview evidence rather than estimated.

Round 6 — commit mirrors the successful preview.

```
CALL update_booking {... "confirm": true}   x3   (arguments identical
↪  to round 4)
RES  {"status": "updated", "booking_id": "BK-1000", "changes_applied":
      ["flight_id", "change_reason"], "price_paid": 338,
      "change_fee": 18, "fare_difference": 20}
```

```
RES  {"status": "updated", "booking_id": "BK-1001", ..., "price_paid":
↪  745,
      "change_fee": 25, "fare_difference": 20}
RES  {"status": "updated", "booking_id": "BK-1002", ..., "price_paid":
↪  730,
      "change_fee": 50, "fare_difference": 30}
```

Round 7 — terminate on evidence. The final message reports exactly the three updated bookings, and the harness records an empty tool-call list together with `completion_evidence = true`.

Outcome. The frozen environment state changes as `BK-1000: UA100→UA101, price 300→338; BK-1001: BA100→BA101, price 700→745; BK-1002: DL100→DL101, price 650→730`, all twelve scored fields match, and the task passes. The episode costs 78,125 runtime tokens over 15 executed tool calls.

Which mechanisms carried the result. The trace makes three of them identifiable rather than merely declared. (i) The *goal ledger* keeps three bookings and two steps per booking in one plan, so the adapter never loses a booking when it moves to the next one; the reviewer event `requested_replan` fires once, at the moment the ledger is temporarily empty, and forces a grounded continuation instead of a premature stop. (ii) The *preview gate* is what turns a multi-round loop into a single-attempt success: the value that is later confirmed is exactly the value that was previewed, so the write cannot drift away from the evidence. (iii) The *batch reviewer* annotates the round-5 read with "`get_booking: successful read already cached`", which stops the loop from re-reading the same three bookings before the commit.

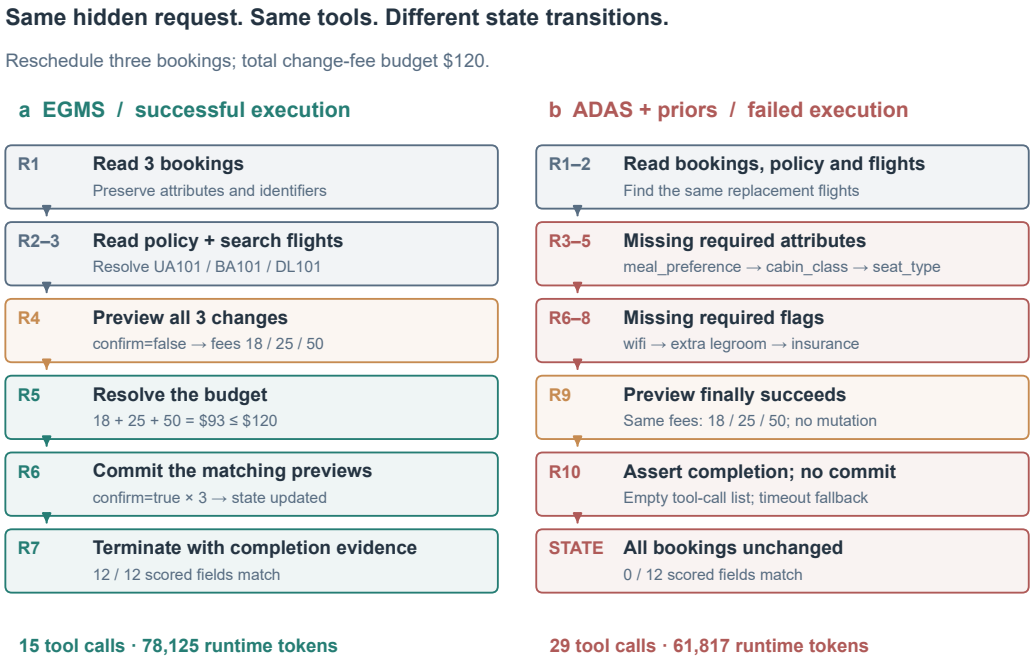


Figure 11: The same hidden travel task succeeds with an explicit preview–commit transition and fails when execution stops after preview. The final environment state determines success.

Additional details for Figure 11. The same held-out travel task under two architectures. (a) EGMS previews with `confirm=false`, resolves the budget against preview evidence, then commits with `confirm=true`. (b) ADAS search finds the same flights and fees, but spends six rounds on schema errors, never mirrors its preview with a commit, and stops while asserting completion.

E.2 Positive Example 2: An Unverifiable Identifier Is Re-derived from Evidence

Instance and demand. Held-out shopping task `143-hard_edge_invalid_product_id_recover`:

```
Add product SP-DOES-NOT-EXIST. If that's wrong, help me find the real
SoundMax headphones.
```

The user's first instruction is unexecutable, because the identifier does not exist in the catalogue. The correct behaviour is neither to refuse nor to invent a substitute, but to resolve the user's real intent (SoundMax headphones) against live catalogue evidence and then act once.

What the architecture did. Figure 12 compresses the trajectory; the rounds are as follows.

Round 1 — the request becomes a resolution goal, not a write.

```
CALL search_products {"query": "SoundMax headphones",
                      "category": "electronics", "in_stock_only":
                      ↪  true}
RES  {"products": [{"product_id": "SP-H114",
                    "name": "SoundMax Elite Noise-Cancelling
                    ↪  Headphones",
                    "price": 249, "rating": 4.8, "in_stock": true},
                   {"product_id": "SP-H113", "name": "SoundCore Basic
                   ↪  Headphones",
                    "price": 149},
                   {"product_id": "SP-H115", "name": "TravelBuds
                   ↪  Mini", "price": 89}],
      "total_found": 3}
```

The plan does not attempt `add_to_cart` with the literal string supplied by the user. The **evidence-grounded reference resolver** treats the requested item as unbound and forces a choice from an enumerated catalogue obtained by a live tool call.

Round 2 — a single grounded mutation.

```
CALL add_to_cart {"customer_id": "shop_003", "product_id": "SP-H114"}
RES  {"status": "added", "cart_item_id": "CI-0001", "product_id":
↪  "SP-H114",
      "product_name": "SoundMax Elite Noise-Cancelling Headphones",
      "quantity": 1, "cart_total": 249}
```

The identifier that reaches the write, `SP-H114`, is one that was observed in a tool result and never one produced by the language model, and the write scope is exactly one item.

Round 3 — the goal is closed on read-back evidence.

```
reviewer: completed_or_blocked_with_no_valid_call   completion = true
```

Outcome. The state diff creates exactly one cart item (`CI-0001`, `shop_003`, `SP-H114`, quantity 1) and moves the cart total from $0 to $249, with no recorded failures. The whole episode costs 10,150 runtime tokens over two tool calls, roughly one seventh of the travel episode above.

Why this is the intended behaviour rather than luck. The trajectory constraints of Section C.1 forbid a write whose arguments are not grounded in an observation, so emitting `add_to_cart( "SP-DOES-NOT-EXIST")` is rejected by the pre-action guard before it can reach the environment. The task also illustrates why the guard cannot be replaced by a stronger model: the failure mode is not a wrong guess but a request that cannot be satisfied as stated, and the repair required is a *structural* step (resolve, then write) rather than a better guess.

Additional details for Figure 12. Positive Example 2 in compressed form. The requested identifier is ungrounded, so the pre-action guard converts the write into a resolution goal, binds it to a catalogue identifier observed in a live search result, and only then commits a single mutation, passing all scored fields.



Figure 12: Invalid-ID recovery: live search supplies the product identifier, one grounded write updates the cart, and the agent terminates on observed evidence.

E.3 Negative Example: The Preview Is Mistaken for Completion

Instance and system. The same travel task `55-challenge_triple_fee_stack` as Appendix E.1, solved by the strongest scalar-fitness baseline (ADAS search under the verbatim EGMS design-space priors and trajectory constraints, budget identical at one initial generation plus up to five repairs). Its Stage 1 ledger records the same starting point as ours (5/20 development tasks after the first generation) but a different search path, and its frozen adapter reaches this hidden task in 10 rounds.
What went wrong. The first three rounds are correct and closely track our own: the same three bookings are read, the same two policy topics and three flight searches are issued, and the correct three replacement flights (UA101, BA101, DL101) are identified. The divergence begins when the adapter first tries to obtain authoritative fee numbers.

Rounds 3–8 — six rounds are spent discovering a required-argument list. Each round issues the same three calls with exactly one more argument filled in, and each round consumes the same class of error:

```
R3  update_booking {booking_id, flight_id, change_reason,
↪   confirm=false}
    -> error: missing required field: meal_preference       (x3)
R4  ... + meal_preference=standard
    -> error: missing required field: cabin_class           (x3)
R5  ... + cabin_class=economy
    -> error: missing required field: seat_type             (x3)
R6  ... + seat_type=window
    -> error: missing required field: add_wifi              (x3)
R7  ... + add_wifi=false
    -> error: missing required field: add_extra_legroom     (x3)
R8  ... + add_extra_legroom=false
    -> error: missing required field: add_insurance         (x3)
```

Every one of the six failed batches was avoidable: the three required attributes and three flags were returned by the round-1 `get_booking` calls and are present in the same conversation. Our configuration reads them out of the observation ledger; this configuration re-derives them one schema error at a time.
Round 9 — the preview finally succeeds.

```
R9  update_booking {... all fields ..., confirm=false}    (x3)
    -> preview: BK-1000 fee 18 + fare 20 = 338
    -> preview: BK-1001 fee 25 + fare 20 = 745
    -> preview: BK-1002 fee 50 + fare 30 = 730
```

The numbers are identical to ours, so the remaining obstacle is purely architectural.
Round 10 — termination without a write. The adapter emits an empty tool-call list with the message *"The requested travel workflow is complete, and I verified the latest official booking result."* The final adapter invocation records `modelCalled=false` with `error="TimeoutError"` and falls back to the previous plan.

Outcome. The environment state is *byte-identical to its initial state*: `state_diff = {created: {}, modified: {}, deleted: {}}`. All twelve scored fields fail as `field_mismatch`, with `actual` either `null` or the old value:

```
BK-1000:  flight_id expected UA101 actual UA100;  change_fee
expected 18 actual null;
  fare_difference expected 20 actual null;  price_paid
```

```
expected 338 actual 300
BK-1001:  flight_id expected BA101 actual BA100;  change_fee
expected 25 actual null;
  fare_difference expected 20 actual null;  price_paid
expected 745 actual 700
BK-1002:  flight_id expected DL101 actual DL100;  change_fee
expected 50 actual null;
  fare_difference expected 30 actual null;  price_paid
expected 730 actual 650
```

The episode costs 61,817 runtime tokens over 29 executed tool calls: more calls than our successful run (15) and about 79% of its tokens, for a strictly worse result.

Diagnosis. The failure has a single causal root, and the architecture makes it visible rather than hiding it behind the final message. The adapter's safety layer never issued the call that would have committed the work, because its precondition is phrased as "*confirm=true requires a matching successful preview*", and the successful preview arrived in round 9, one round before termination. The termination rule is then evaluated against a completion predicate that accepts "the requested workflow is complete" as sufficient, even though the same event stream records zero completed mutations for this task. Three of the constraints of Section C.1 are therefore jointly violated, and each violation is individually observable:

- **Commit after preview.** The preview of round 9 was never mirrored by a `confirm=true` call; the preview was treated as the terminal artifact of the workflow instead of as an intermediate evidence object.
- **No silent termination.** The run ends with unresolved goals and a message that asserts completion. The correct behaviour is either to commit from the preview or to report the blocked state explicitly.
- **No repeated failed call.** Six consecutive rounds re-issue a batch that differs only by the argument the previous error named. The calls are not literally identical, so an equality-based replay guard does not catch them, but they are the same *mechanism* failing repeatedly — exactly the condition that EGMS's falsification rule, Eq. 5, is meant to interrupt by rebuilding along a different design direction instead of extending the current one.

Why the baseline arrived here. This is the mechanism that Appendix D.5 predicts. Scalar-fitness search selects the candidate with the best development score, and the development split contains no task that punishes a preview-without-commit design, so nothing in the scalar signal distinguishes this adapter from ours at design time. Causal diagnosis would have grouped the six schema-error rounds into one failure cluster whose divergence point is the missing observation read, and counterexample probing, varying the number of required fields, the booking count, and the presence of a budget constraint, would have exposed the termination predicate as the shared mechanism. Both are absent from a scalar objective, and the held-out task therefore punishes a design with which the development evidence was perfectly satisfied.

What this example does not show. It does not show that the backbone model is weaker: this adapter calls the same local runtime model as ours, and its reasoning in rounds 1–3 is correct. Nor does it show that the task is unsolvable for scalar search in general, since the same baseline solves other cascaded travel tasks and its three-run mean on travel is 12.2–24.5% rather than zero. The claim supported here is narrower, and it is the claim the paper makes: on a hidden task that requires a preview-gated, multi-object write, an architecture selected by development score alone can reach the correct preview and still fail to commit, and the failure is attributable to a missing mechanism rather than to an insufficient search budget.

E.4 State-Based Comparison of the Three Episodes

Table 7 places the terminal evidence next to the execution cost. A tool round is one adapter invocation and may contain several tool calls or no calls. This distinction explains the seven-round successful travel trajectory and the three-round shopping trajectory despite their different call counts.

The preview–commit boundary.

Table 7: Observed outcomes of the three case-study episodes. Success is determined by environment state, and runtime token totals are episode-specific.

Episode	Rounds	Calls	Tokens	State evidence
Travel / EGMS	7	15	78,125	3 bookings updated; 12/12 fields
Shopping / EGMS	3	2	10,150	1 cart item created; task passed
Travel / ADAS + priors	10	29	61,817	No mutation; 0/12 fields

Booking	Original flight	EGMS terminal state	ADAS terminal state
BK-1000	UA100	**UA101; price 338**	**UA100; price 300**
BK-1001	BA100	**BA101; price 745**	**BA100; price 700**
BK-1002	DL100	**DL101; price 730**	**DL100; price 650**

The successful trace issues matching `confirm=true` calls after obtaining the preview. The failed trace reaches the same previewed fees, but no committed update appears in its event stream. The observed failure is therefore a missing state transition; the counterfactual claim that a different search signal would necessarily prevent it is not established by these individual episodes.

F Related Work and Protocol Comparison

F.1 Extended Related Work

We organize related work around creativity, the artifacts optimized during agent design, and the environments used to evaluate execution. These lines motivate complementary requirements for evaluating agents that create agents.

(I) Creativity in LLMs and agents. *(i) Creative writing and diverse generation.* Art or Artifice introduces expert assessments of creative writing (Chakrabarty et al., 2024), while NoveltyBench evaluates the diversity and quality of model generations (Zhang et al., 2025c). Creativity Index measures linguistic originality through attribution to existing web text (Lu et al., 2025). CreataSet and CrEval extend textual creativity evaluation across domains with a dataset and a learned pairwise evaluator (Cao et al., 2026). Together, these studies make originality measurable beyond answer correctness, but their evaluated objects are textual responses. *(ii) Research ideation and executable creativity.* An expert study of LLM research agents separately assesses the novelty and feasibility of generated research proposals (Si et al., 2025). MetaMuse goes beyond text by guiding creative algorithm generation using diversity and usefulness in performance space, external stimuli, and executable solutions (Ma et al., 2026). This distinction matters for ACA: executable creativity has precedents, while our target is the reusable agent architecture, whose utility, template-relative novelty, and design cost are evaluated together under a shared hidden-task protocol.

(II) Automated agent design and reusable knowledge. *(i) Architecture and workflow search.* ADAS programs agents in code (Hu et al., 2025), GPTSwarm optimizes computation graphs (Zhuge et al., 2024), and AFlow searches workflow code with Monte Carlo tree search (Zhang et al., 2025b). AgentSquare searches modular components (Shang et al., 2025), while DyLAN adjusts agent teams for individual tasks (Liu et al., 2024). MASS optimizes prompts and topologies (Zhou et al., 2026), and the Conductor, which learns communication structures and worker instructions through reinforcement learning (Nielsen et al., 2026). MaAS instead searches an agentic supernet that samples query-dependent architectures, adapting inference resources to each query (Zhang et al., 2025a). These methods optimize coordination for task performance; ACA supplies a common protocol for assessing the resulting architecture as a created artifact, including its distance from a frozen template library. *(ii) Harness adaptation.* Meta-Harness searches the full scaffold (Lee et al., 2026), and Better Harnesses studies automated adaptation for lower execution cost (Yang et al., 2026). A taxonomy of harness design decisions further describes the structural choices available to such searches (Wei, 2026). These recent preprints motivate treating orchestration as an optimization target alongside the backbone. EGMS instead focuses on the development signal used to search that space: causal failure clusters, mechanism-level repairs, and counterexample-gated acceptance. *(iii) Prompts, memory, and skills.* Agent Workflow Memory induces reusable workflows from trajectories (Wang et al., 2024); ReasoningBank and GEPA respectively distill reasoning memory and evolve prompts through reflective feedback (Ouyang et al., 2026; Agrawal et al., 2026). SkillsBench evaluates the effectiveness of skill documents, while SkillAxe refines generated skills using evaluation feedback (Li et al., 2026b; Gautam et al., 2026). SkillFlow, SkillLearnBench, and SWE-

Skills-Bench examine skill discovery, continual learning, and software-engineering transfer (Zhang et al., 2026; Zhong et al., 2026; Han et al., 2026). Budget-controlled analysis further questions whether online skill and memory modules repay their token costs (Hajimiri et al., 2026). These approaches primarily improve knowledge consumed by an agent loop. ACA instead allows domain knowledge to be embedded in executable control flow and tool policies, and evaluates the resulting execution cost separately from the cost of creating the architecture.

(III) Agent execution and evaluation benchmarks. *(i) Software, web, and desktop tasks.* SWE-bench uses tests to evaluate repository repair (Jimenez et al., 2024), GAIA evaluates general assistance (Mialon et al., 2023), and WebArena, WorkArena, and OSWorld provide realistic web, knowledge-work, and desktop environments (Zhou et al., 2024; Drouin et al., 2024; Xie et al., 2024). SpreadsheetBench adds test-based evaluation for spreadsheet manipulation (Ma et al., 2024). Their environments support reproducible execution comparisons, but execution success does not itself measure the creativity of an architecture's design. WorfBench directly benchmarks workflow generation using WorfEval's subsequence and subgraph matching, and also evaluates the downstream usefulness of generated workflows (Qiao et al., 2025). It is therefore a close precedent for evaluating generated control structures. ACA evaluates reusable architectures across hidden task executions, while its novelty measure compares against a frozen template library rather than a reference workflow. *(ii) Stateful and interactive workflows.* AppWorld exposes executable application APIs (Trivedi et al., 2024), τ-bench includes tool–agent–user interaction (Yao et al., 2024), and TheAgentCompany evaluates enterprise work through environment states (Xu et al., 2026). Toolathlon extends execution-based evaluation to realistic workflows spanning multiple applications (Li et al., 2026a). STATE-Bench supplies the stateful customer-support, shopping, and travel environments adapted by ACA (Microsoft, 2026). We build on this execution infrastructure while making the generated architecture the object evaluated across development and hidden tasks. *(iii) Evaluation infrastructure, cost, and reliability.* HAL standardizes evaluations across models, scaffolds, and benchmarks and uses cost accounting and log inspection to diagnose agent behavior (Kapoor et al., 2026). Towards a Science of AI Agent Reliability complements success rates with consistency, robustness, predictability, and safety measurements (Rabanser et al., 2026). A study of meta-agent inefficiencies further examines behavioral diversity and the combined cost of designing and deploying generated agents (El et al., 2025). This motivates distinguishing a runtime efficiency gain from amortization of architecture search. These studies show why a single success rate is insufficient. ACA addresses a different evaluation target: the architectural creativity of a frozen manifest and executable adapter, measured through hidden-task utility, template-relative novelty, and design cost.

F.2 Capability Comparison with Related Protocols

Table 8 makes the positioning claim of Section 2 checkable by decomposing it into six protocol capabilities. A benchmark or method either exposes a capability as part of its published protocol, or it does not. The column *artifact evaluated* is the object that the protocol scores, and it is the axis on which ACA differs most sharply: every other line scores model or pipeline behaviour, whereas ACA scores the created architecture as a frozen, reusable artifact.

Table 8: Protocol capabilities across related lines of work. Filled circles denote native support, open circles partial support, and dashes absence; the final row describes ACA.

Line of work	Representative protocol	Artifact evaluated	Interactive env.	Held-out design	Frozen reference	Novelty scored	Cost in objective	Creativity score
Execution benchmarks	SWE-bench, GAIA, WebArena, τ-bench	model / harness	✓	–	–	–	–	–
Enterprise execution	TheAgentCompany, STATE-Bench	model / harness	✓	–	–	–	–	–
Prompt & skill learning	AWM, ReasoningBank, GEPA	prompt / skill text	✓	●	–	–	●	–
Prompt & skill learning	SkillsBench, SkillAxe, SkillFlow	prompt / skill text	✓	●	–	–	●	–
Architecture search	ADAS, GPTSwarm, AFlow, MASS	pipeline code	●	●	–	–	–	–
Architecture search	AgentSquare, DyLAN, Meta-Harness	pipeline code	●	●	–	–	●	–
Creativity benchmarks	NoveltyBench	generated text	–	–	●	✓	–	–
ACA (ours)	**EGMS over agent architectures**	**frozen architecture**	✓	✓	✓	✓	✓	✓

Protocol capabilities across the lines of related work. A filled circle marks a capability the protocol provides natively; an open circle marks a partial or implicit version; a dash marks absence. The last row is the present work. *Held-out design* asks whether candidates are designed without access to the evaluation split and frozen before it is touched; *frozen reference* answers "novel relative to what?"; *cost term* asks whether token cost enters the reported objective rather than being listed separately.

Reading the table. Three entries deserve comment. First, the *held-out design* column is what separates architecture search from ACA: search methods do evaluate candidates, but they select on the same signal they optimize, whereas ACA forbids the Architect from observing the held-out split and freezes the artifact before any held-out execution. Second, the *frozen reference* column is

populated only twice, by NoveltyBench (a reference corpus of human text) and by ACA (a frozen library of 100 architecture manifests); without such a reference a novelty claim has no denominator, which is why Appendix B.1 specifies retrieval and rubric separately. Third, the *cost in objective* column marks where token accounting is part of the published score rather than a footnote: budget-aware skill studies report cost but do not fold it into a single ranking, and ACA's cost term explicitly penalizes design expenditure when comparing executable architectures.

What the table does not claim. It does not claim that the other protocols lack novelty in the design sense — a SWE-bench harness is a substantial engineering artifact. It claims that the published protocols do not *measure* that property, so the architectural creativity of agents that create agents is currently unmeasured rather than measured and found small.

G Evaluation Scope and Limitations

Our split is an internal resplit of the official training tasks, not the official STATE-Bench test protocol. Stage 2 scores the final state of a single attempt per task; 26 tasks that require user confirmation before acting (26 of 238 adapted tasks, unevenly distributed, at 19 of 82 in customer support) are excluded, so the current protocol covers only the autonomously completable subset of the original benchmark, and richer interactive protocols may shift absolute numbers. The RQ1 comparison reports highest single-run outcomes from three runs and evaluations. It characterizes attained performance, not expected performance across repetitions; the aggregate table does not establish domain-level rankings or development–held-out gaps. The Creativity score depends on a frozen cost reference, and the novelty term relies on an agent-as-judge and a fixed template library; both warrant sensitivity analysis in future work. Baseline search methods are adapted to the same guidance interface and budget, so their numbers describe these adaptations rather than the original pipelines (Appendix C.2).