SKILLSPEC: CONSENSUS-GATED AGENT SKILL EVOLUTION VIA REPRESENTATION SPECIALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Natural-language *skills* are textual procedural memories through which large language model (LLM) agents retain reusable task knowledge without updating model weights. Existing methods typically treat skills as either static artifacts or monolithic documents optimized using aggregate validation scores as feedback. However, representing a skill as a monolithic document restricts optimization to its textual content, without explicitly modeling the structure through which procedural knowledge is retrieved and executed. We identify a key distinction between learning what knowledge to retain and determining how to organize it: textual updates should first be validated through execution evidence, after which the retained knowledge should be structured according to its procedural dependencies and retrieval requirements. To this end, we introduce **SkillSpec**, a two-phase framework comprising consensus-gated evolution and representation specialization. In the consensus-gated phase, complementary editing intents generate complete candidate skills, which are accepted as updates only when paired evaluations reach consensus: an overall improvement over the current skill with non-negative aggregate paired gain in every repeated evaluation. In the specialization phase, signals of process and redundancy sensitivity derived from the optimization trajectory guide the selection of a *flat*, *graph*, or *hybrid* representation while preserving the validated backbone as an immutable lineage source while allowing the derived artifact to evolve. Across six benchmarks and three target language models, SkillSpec improves average success rate over SkillOpt-Lite by an absolute margin of 4.8%, averaged across the three models. These results demonstrate that reliable skill evolution and representation specialization address complementary objectives: deciding what knowledge to retain and how to structure it for inference.

1 INTRODUCTION

Large language model (LLM) agents increasingly use natural-language skills to retain and reuse task knowledge without updating model weights. A *skill* is a textual procedural memory that encodes reusable steps, task rules, tool-use guidance, answer formats, and error-recovery strategies while remaining easy to inspect and revise. However, static skills are often insufficient for long-horizon, dynamic tasks, where execution feedback continually reveals missing procedures, ineffective guidance, and evolving dependencies. This has motivated recent methods to treat prompts and skills as optimizable state (Zhou et al., 2023; Yang et al., 2024; Khattab et al., 2023; Yuksekgonul et al., 2024; Agrawal et al., 2025; Yang et al., 2026b), replacing one-shot construction with iterative revision and reorganization informed by execution trajectories.

SkillOpt (Yang et al., 2026b) takes an important step toward controlled skill evolution through bounded, validation-gated edits to a trainable skill document. Nevertheless, aggregate validation scores can obscure an edit's behavioral effects: improvements on some examples may conceal regressions on others, and apparent gains may not persist across repeated evaluations. Existing optimizers also preserve a single-document representation, changing only its wording while leaving the organization of knowledge fixed. Flat representations can obscure procedural dependencies in sequential tasks, whereas unnecessary structure can introduce spurious dependencies and irrelevant context. Our key observation is that content selection and structural organization rely on different forms of evidence and should therefore be optimized separately; conflating them in a single text-rewriting process obscures the source of improvements and makes regressions difficult to reverse.

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.

We introduce **SkillSpec**, a two-phase framework that separates reliable content evolution from representation specialization. In the first phase, SkillSpec generates candidate skills using four complementary editing intents—*repair*, *preserve*, *simplify*, and *rewrite*—and compares them with the current skill on the same validation instances. The consensus gate accepts an update only when it improves performance overall with non-negative aggregate paired gain in every repeated evaluation, while retaining evidence from rejected candidates to inform representation specialization. Building on the validated skill and the trajectory of candidate evaluations, the second phase estimates process and redundancy sensitivity, then uses these signals to select a *flat*, *graph*, or *hybrid* representation. To preserve a common starting point and provenance, Phase II leaves the hash-verified Phase I checkpoint, optimizer state, and evidence unchanged and modifies only a derived artifact in the selected representation. The source checkpoint is immutable, but the derived artifact may change both its guidance and organization.

We evaluate SkillSpec on six benchmarks, SearchQA (Dunn et al., 2017), DocVQA (Mathew et al., 2021), LiveMath (He et al., 2026), ALFWorld (Shridhar et al., 2021), SpreadsheetBench (Ma et al., 2024), and OfficeQA (Opsahl-Ong et al., 2026), using several frontier language models. Across the six benchmarks, SkillSpec improves average success rate over the no-skill baseline by an absolute margin of up to 24.59%. Compared with the state-of-the-art SkillOpt (Yang et al., 2026b), SkillSpec yields absolute gains of 10.55%, 5.92%, and 3.12% for GPT–4.1, GPT–5.4 Nano, and GPT–5.4, respectively, corresponding to an average gain of 6.53% across the three models. These results suggest that SkillSpec provides larger gains as model capability decreases, highlighting the value of explicit skill reorganization for less capable models. Our contributions can be summarized as follows:

- We introduce consensus-gated skill evolution, which combines multi-intent candidate generation with paired validation to accept reliable content updates, reject edits that fail the aggregate gain thresholds, and retain evidence from rejected candidates to inform representation specialization.
- We address the fixed, monolithic representation used by existing skill optimizers with trajectory-guided specialization, which preserves the validated source checkpoint while selecting among *flat*, *graph*, and *hybrid* representations.
- We evaluate SkillSpec across six benchmarks and several frontier language models, achieving higher average success rates than state-of-the-art SkillOpt (Yang et al., 2026b) for every evaluated model.

2 Related Work

Prompt and skill optimization. A growing line of work treats natural-language prompts as optimizable artifacts rather than static instructions. Early methods optimize prompts by searching over candidate instructions using task-level scores as feedback (Zhou et al., 2023; Yang et al., 2024), or by compiling prompts and demonstrations for compound LM programs (Khattab et al., 2023). TextGrad (Yuksekgonul et al., 2024) frames prompt optimization through an analogy to gradient descent, propagating natural-language feedback through the computation graph of a compound LLM system. GEPA (Agrawal et al., 2025) instead reflects on execution trajectories to evolve prompts and maintains a Pareto frontier of candidates that capture complementary gains across validation instances. SkillOpt (Yang et al., 2026b) extends iterative optimization from prompts to persistent skills, using execution trajectories to propose bounded revisions and held-out evaluations to select updates. SkillOpt-Lite (Shen et al., 2026) simplifies this process by replacing tree-based candidate merging with direct inspection and revision of file-based trajectories, while retaining an independent validation gate. Recent methods improve the optimization process by prioritizing skill candidates with greater potential for further refinement (Shastri et al., 2026) and by stabilizing update directions while adapting edit budgets to instance-level variability (Li et al., 2026). Complementary work trains skill editors from execution feedback (Yang et al., 2026a; Vishe et al., 2026; Peng et al., 2026), repairs weak initial skills (Liu et al., 2026b), and supports long-horizon curation of skill repositories (Ouyang et al., 2026). Collectively, these methods make skill revision and curation increasingly responsive to execution feedback. However, they primarily optimize skill content or maintenance, rather than using optimization evidence to determine how a validated skill should be structured for inference.

Skill organization and representation. Beyond their textual content, skills can be organized at multiple levels. Existing systems specify lifecycle conditions for executable skills (Mi et al.,

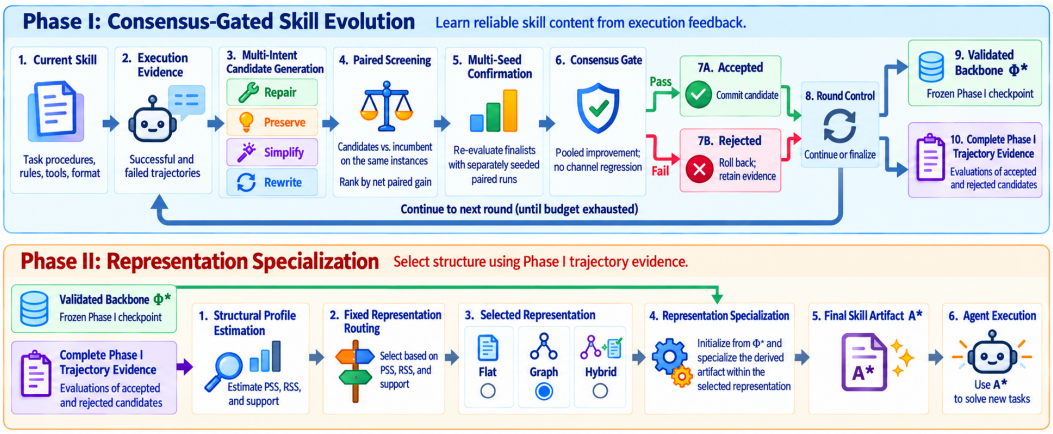


Figure 1: Overview of SkillSpec. Phase I combines multi-intent candidate generation with paired, repeated validation to produce a validated backbone and trajectory evidence. Phase II uses PSS, RSS, and evidence support to select a *flat*, *graph*, or *hybrid* representation and specializes a derived artifact initialized from the backbone.

2026), arrange skill collections through multi-level retrieval structures (Wang et al., 2026), or encode procedural relations using hypergraphs and dependency graphs (Xu et al., 2026; Liu et al., 2026a). Other work compiles prose into schema-validated execution graphs (Blumenfeld & Webber, 2026) or selects among heterogeneous artifacts such as rules, skills, and bounded subagents (Qiu et al., 2026). These approaches generally prescribe a representation or select among predefined artifact types. SkillSpec instead derives process and redundancy sensitivity from each benchmark's Phase I optimization trajectory and uses these signals to select a *flat*, *graph*, or *hybrid* representation while preserving the validated backbone as the immutable source of a derived artifact. Concretely, SkillSpec constructs a structural profile from evaluations of both accepted and rejected Phase I candidates, applies a fixed routing rule, and performs specialization only for the selected representation.

3 SkillSpec: A Two-Phase Framework for Skill Evolution

SkillSpec optimizes an external skill artifact $\mathcal{A}$ for a fixed policy model π_θ without updating the model parameters θ. Given training data $\mathcal{D}_{\text{tr}}$ and validation data $\mathcal{D}_{\text{val}}$, executing π_θ conditioned on $\mathcal{A}$ for an instance x produces a trajectory $\tau_x(\mathcal{A})$, which receives an evaluator score $s(\pi_\theta, x, \mathcal{A})$. Evidence from both datasets is then used to optimize $\mathcal{A}$, with the goal of maximizing expected performance on unseen test instances.

Given an initial textual artifact Φ_0, SkillSpec decomposes artifact optimization into content evolution and representation specialization. Phase I uses training and validation evidence to produce a validated textual backbone $\Phi^\star$ and an optimization trajectory Γ_{I} containing candidate edits, paired execution outcomes, validation decisions, and residual evidence. Phase II estimates structural sensitivity from Γ_{I} and selects a representation $r^\star \in \{\textit{flat}, \textit{graph}, \textit{hybrid}\}$. It then initializes a representation-specific artifact from $\Phi^\star$ and specializes it to obtain $\mathcal{A}^\star$. Figure 1 illustrates the two-phase workflow and the artifacts passed from Phase I to Phase II.

3.1 Phase I: Consensus-Gated Skill Evolution

Phase I evolves Φ_0 over T optimization rounds. At each round, the policy model executes the current skill on a training batch, and the resulting successful and failed trajectories inform candidate updates generated under complementary editing intents. Candidates are screened against the current skill using task-level paired gains, and only updates confirmed across multiple executions are committed. The skill after the final round is returned as the validated backbone $\Phi^\star$.

Collecting execution evidence. At round t, the fixed policy model uses the current skill Φ_t to solve a sampled training batch $\mathcal{B}_t \subset \mathcal{D}_{\text{tr}}$. Phase I retains the resulting trajectories, evaluator outcomes, and

residual diagnoses, separating successful from failed executions. Together, these records form the execution evidence $\mathcal{E}_t$ used to propose candidate updates.

Multi-intent candidate generation. Given $\mathcal{E}_t$, the skill optimizer constructs candidate skills under the intent set $\mathcal{U} = \{\textit{repair}, \textit{preserve}, \textit{simplify}, \textit{rewrite}\}$. *Repair* adds corrective guidance for recurring failures; *preserve* retains guidance supported by successful trajectories; *simplify* removes redundant or conflicting guidance; and *rewrite* restructures the skill when local edits are insufficient. Each intent yields a complete skill candidate rather than an isolated edit. Compatible changes from different intents can also be combined into an additional candidate. The resulting candidate pool is $\mathcal{S}_t = \{\widetilde{\phi}_{t,j}\}_j$, where j indexes the candidates generated at round t. Keeping these candidates distinct allows Phase I to evaluate each update strategy independently rather than fold all proposed changes into a single rewrite.

Screening with paired outcomes. Each candidate $\widetilde{\phi}_{t,j} \in \mathcal{S}_t$ and the current skill Φ_t are evaluated on the same validation instances. Their evaluator scores are compared for each $x_i \in \mathcal{D}_{\text{val}}$, yielding the paired counts

$$\begin{aligned} n^{+}_{t,j} &= \sum_{x_i \in \mathcal{D}_{\text{val}}} \mathbf{1}\Big[s(\pi_\theta, x_i, \widetilde{\phi}_{t,j}) > s(\pi_\theta, x_i, \Phi_t)\Big], \\ n^{-}_{t,j} &= \sum_{x_i \in \mathcal{D}_{\text{val}}} \mathbf{1}\Big[s(\pi_\theta, x_i, \widetilde{\phi}_{t,j}) < s(\pi_\theta, x_i, \Phi_t)\Big]. \end{aligned} \tag{1}$$

Here, $n^{+}_{t,j}$ and $n^{-}_{t,j}$ count task-level improvements and regressions, respectively; ties contribute to neither count. Their difference $g_{t,j} = n^{+}_{t,j} - n^{-}_{t,j}$ is the net paired gain. This comparison reveals regressions that gains on other tasks could mask in an aggregate score. Phase I ranks candidates in descending order of $g_{t,j}$, resolves equal gains deterministically, and sends the highest-ranked N_f candidates to the confirmation stage.

Confirming and committing an update. For each finalist $\widetilde{\phi}_{t,j}$, let $g^{(k)}_{t,j}$ denote its paired gain in evaluation $k \in \{0, \ldots, K\}$ on the complete $\mathcal{D}_{\text{val}}$. We set $g^{(0)}_{t,j} = g_{t,j}$ to reuse the screening result, while $k = 1, \ldots, K$ index additional confirmation executions. Let G_{pool} denote the required pooled gain and $G_{\min}$ the minimum gain permitted in any single execution. Candidate $\widetilde{\phi}_{t,j}$ is confirmed only if

$$\sum_{k=0}^{K} g^{(k)}_{t,j} \geq G_{\text{pool}} \qquad \text{and} \qquad \min_{0 \leq k \leq K} g^{(k)}_{t,j} \geq G_{\min}. \tag{2}$$

The pooled-gain condition requires sufficient overall improvement, while the minimum-gain condition ensures that every execution meets $G_{\min}$. With $G_{\min} = 0$, this requires non-negative aggregate paired gain in each execution; individual validation instances may still regress. If multiple candidates satisfy both conditions, Phase I selects the candidate with the largest pooled gain, resolves ties deterministically, and denotes it by $\widetilde{\phi}^{\star}_t$. The update is then applied atomically:

$$\Phi_{t+1} = \begin{cases} \widetilde{\phi}^{\star}_t, & \text{if } \widetilde{\phi}^{\star}_t \text{ passes (2)}, \\ \Phi_t, & \text{otherwise.} \end{cases} \tag{3}$$

If an update is rejected, Phase I retains the current skill and optimizer state. After each round, it appends the evidence, gains, and decision to the optimization trajectory. After T rounds, Phase I returns the validated backbone $\Phi^{\star}$, final optimizer state Ω_{I}, and complete trajectory Γ_{I}. Algorithm 1 summarizes the full update loop.

3.2 Phase II: Telemetry-Guided Structural Specialization

Phase II takes $(\Phi^{\star}, \Omega_{\text{I}}, \Gamma_{\text{I}})$ from Phase I as input. Using Γ_{I}, it estimates structural sensitivity, selects a representation, and specializes an artifact initialized from $\Phi^{\star}$ while keeping these Phase I outputs unchanged.

Algorithm 1 Phase I: consensus-gated skill evolution

Input: Fixed policy model π_θ, initial textual artifact Φ_0, training data $\mathcal{D}_{\mathrm{tr}}$, validation data $\mathcal{D}_{\mathrm{val}}$, round budget T, finalist count N_f, number K of confirmation executions, thresholds $G_{\mathrm{pool}}, G_{\min}$, and intent set $\mathcal{U}$
Output: Validated backbone $\Phi^\star$, final optimizer state Ω_{I}, and optimization trajectory Γ_{I}

Initialize optimizer state Ω_0 and trajectory $\Gamma_0 \leftarrow \varnothing$
for $t = 0, \ldots, T-1$ **do**
  Draw $\mathcal{B}_t \subset \mathcal{D}_{\mathrm{tr}}$; $\mathcal{E}_t \leftarrow$ CollectEvidence$(\pi_\theta, \Phi_t, \mathcal{B}_t)$
  $\mathcal{S}_t \leftarrow$ GenerateCandidates$(\Phi_t, \Omega_t, \mathcal{E}_t, \mathcal{U})$, including compatible composition
  for all $\widetilde{\phi}_{t,j} \in \mathcal{S}_t$ **do**
    Evaluate $\widetilde{\phi}_{t,j}$ and Φ_t on $\mathcal{D}_{\mathrm{val}}$; compute $g_{t,j}$ using Equation (1)
  end for
  $\mathcal{F}_t \leftarrow$ top N_f candidates ranked by $g_{t,j}$
  for all $\widetilde{\phi}_{t,j} \in \mathcal{F}_t$ **do**
    Set $g_{t,j}^{(0)} \leftarrow g_{t,j}$; run K additional paired evaluations on $\mathcal{D}_{\mathrm{val}}$ to obtain $\{g_{t,j}^{(k)}\}_{k=1}^{K}$
  end for
  $\mathcal{Q}_t \leftarrow \{\widetilde{\phi}_{t,j} \in \mathcal{F}_t : \sum_{k=0}^{K} g_{t,j}^{(k)} \geq G_{\mathrm{pool}} \wedge \min_{0 \leq k \leq K} g_{t,j}^{(k)} \geq G_{\min}\}$
  if $\mathcal{Q}_t \neq \varnothing$ **then**
    $\widetilde{\phi}_t^\star \leftarrow \arg\max_{\widetilde{\phi}_{t,j} \in \mathcal{Q}_t} \sum_{k=0}^{K} g_{t,j}^{(k)}$
    Atomically set $\Phi_{t+1} \leftarrow \widetilde{\phi}_t^\star$ and update Ω_{t+1}
  else
    $(\Phi_{t+1}, \Omega_{t+1}) \leftarrow (\Phi_t, \Omega_t)$
  end if
  Append evidence, gains, and the decision to Γ_t, forming Γ_{t+1}
end for
return $\Phi^\star \leftarrow \Phi_T$, $\Omega_{\mathrm{I}} \leftarrow \Omega_T$, and $\Gamma_{\mathrm{I}} \leftarrow \Gamma_T$

Estimating structural sensitivity from telemetry. Let $\mathcal{I}_{\mathrm{I}}$ index the valid candidates recorded in Γ_{I}, and let $\nu = |\mathcal{I}_{\mathrm{I}}|$. Phase II summarizes their structural sensitivity as

$$\mathcal{T} = (\psi, \rho, \nu), \tag{4}$$

where $\mathcal{T}$ is the structural profile of the validated backbone, ψ is *process sensitivity*, ρ is *redundancy sensitivity*, and ν is the number of valid candidates.

Tasks differ in how their skill guidance should be organized. For example, SearchQA (Dunn et al., 2017) relies on independently retrievable evidence-selection and answer-extraction guidance, whereas LiveMath (He et al., 2026) benefits from an ordered procedure for formalizing and comparing theorem-style claims. A fixed representation may therefore fail to capture necessary dependencies or impose structure on otherwise independent guidance. SkillSpec captures this trade-off with two signals: *process sensitivity* measures dependence on ordered steps, while *redundancy sensitivity* measures vulnerability to duplicated, conflicting, or excessive guidance. Together, these signals guide the choice among *flat*, *graph*, and *hybrid* representations.

Process and redundancy sensitivity estimation. Each candidate $\widetilde{\phi}_{t,j}$, $(t,j) \in \mathcal{I}_{\mathrm{I}}$, contributes one training tuple $(g_{t,j}, p_{t,j}, r_{t,j}, c_{t,j})$. The label $g_{t,j}$ is its screening gain. The *process* score $p_{t,j}$ measures whether the revision adds an ordered procedure whose later steps depend on earlier results; the *interference* score $r_{t,j}$ measures duplicated, conflicting, irrelevant, or excessively verbose guidance. Both use a four-level rubric: 0 (absent), 1 (weak or local), 2 (clear), and 3 (dominant). The normalized edit size is

$$c_{t,j} = D_{\mathrm{line}}(\Phi_t, \widetilde{\phi}_{t,j}) \,/\, \max\{L(\Phi_t), L(\widetilde{\phi}_{t,j})\}, \tag{5}$$

where D_{line} counts inserted, deleted, or replaced lines and $L(\cdot)$ denotes the number of lines.

For each benchmark, ridge regression fits a sensitivity function f to these tuples. Let $f_{t,j}(p, r)$ be the predicted gain for candidate (t, j) after setting its *process* and *interference* scores to (p, r) and leaving its remaining inputs unchanged. Comparing mean predictions with the zero-score baseline

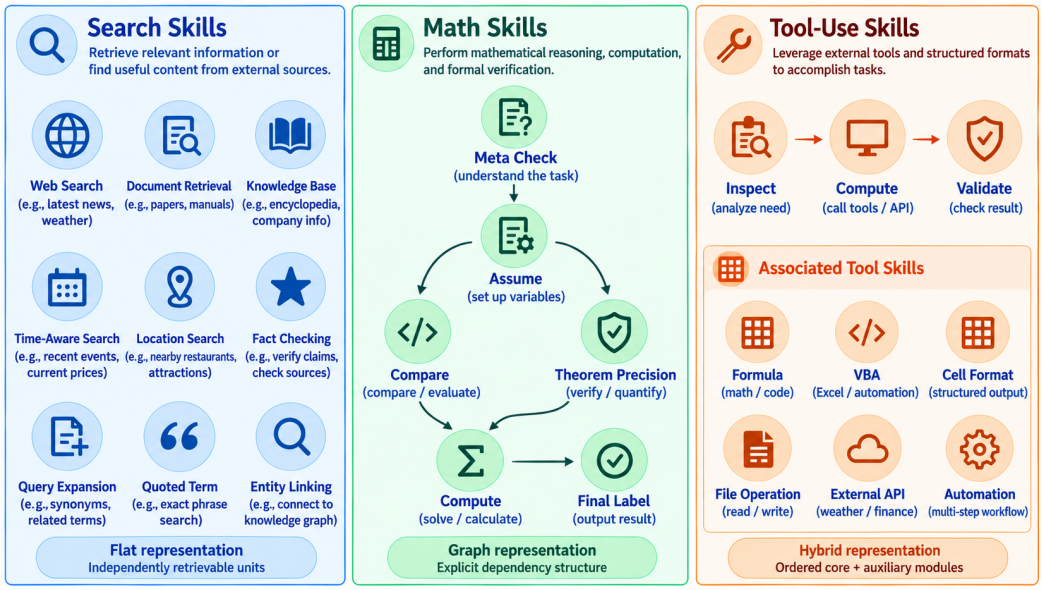


Figure 2: Illustrative skill representations. *Flat* organizes guidance as independently retrievable units; *graph* encodes explicit dependencies among units; and *hybrid* combines an ordered core with auxiliary modules.

gives

$$\Delta_{\mathrm{P}} = \max_{p\in\{1,2,3\}} \frac{1}{\nu} \sum_{(t,j)\in\mathcal{I}_{\mathrm{I}}} f_{t,j}(p,0) - \frac{1}{\nu} \sum_{(t,j)\in\mathcal{I}_{\mathrm{I}}} f_{t,j}(0,0),$$
$$\Delta_{\mathrm{R}} = \frac{1}{\nu} \sum_{(t,j)\in\mathcal{I}_{\mathrm{I}}} f_{t,j}(0,0) - \min_{r\in\{1,2,3\}} \frac{1}{\nu} \sum_{(t,j)\in\mathcal{I}_{\mathrm{I}}} f_{t,j}(0,r). \tag{6}$$

Δ_{P} compares the best nonzero *process* level with the baseline, while Δ_{R} measures the largest predicted loss across nonzero *interference* levels.

We construct N round-bootstrap datasets by sampling complete optimization rounds with replacement. PSS and RSS are the fractions with $\Delta_{\mathrm{P}} > 0$ and $\Delta_{\mathrm{R}} > 0$, respectively:

$$\psi = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\{\Delta_{\mathrm{P}}^{(n)} > 0\}, \qquad \rho = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}\{\Delta_{\mathrm{R}}^{(n)} > 0\}. \tag{7}$$

PSS (ψ) therefore measures evidence that explicit *process* structure improves candidate outcomes, while RSS (ρ) measures evidence that suppressing *interference* improves them.

Adaptive representation selection. Phase II selects among three structural representations: *(a) flat* forms a skill bank of independently retrievable units without an imposed execution order; *(b) graph* connects guidance nodes through explicit dependencies to encode the required execution order; and *(c) hybrid* combines an ordered core with independently retrievable auxiliary modules. Figure 2 illustrates these three structures.

The selector first uses PSS (ψ) to determine whether explicit ordering is warranted. For high-PSS profiles, RSS (ρ) then distinguishes *graph* from *hybrid*. RSS measures outcome evidence for harm from *interference*, not the prevalence of redundant guidance itself. Low PSS therefore selects *flat* to avoid imposing unsupported order; high PSS with low RSS selects *graph* to encode stable dependencies; and all other high-PSS profiles select *hybrid* to preserve an ordered core while modularizing redundancy-sensitive guidance. Because PSS and RSS are bootstrap proportions, we set $\zeta_\psi = \zeta_\rho = 0.50$, treating a sensitivity as present when a majority of bootstrap datasets support its positive contrast. Table 1 summarizes the resulting decision rule and the permissible specialization updates for each representation.

Table 1: Adaptive representation selection from PSS (ψ) and RSS (ρ), where ζ_ψ and ζ_ρ denote the corresponding thresholds and $\Phi^\star$ denotes the frozen validated backbone.

Structure	Condition	Specialization
flat	$\psi \leq \zeta_\psi$	Initialize a derived copy from $\Phi^\star$, split it into self-contained guidance units, and add, revise, merge, or remove units without learned edges or a prescribed order.
graph	$\psi > \zeta_\psi, \rho \leq \zeta_\rho$	Initialize a derived graph from $\Phi^\star$; jointly add, revise, or remove nodes and directed dependency edges; and preserve a valid execution order.
hybrid	Otherwise	Initialize an ordered core from $\Phi^\star$; add, revise, or remove auxiliary modules and update their attachment points.

Specialization procedure. Phase II specializes the selected representation through iterative, validation-gated updates. At each epoch, the target model uses training trajectories and evaluator feedback to propose one revision, subject to the representation-specific update rules in Table 1. The revision may modify guidance, topology, retrieval metadata, or auxiliary modules. The candidate is evaluated against the current artifact on the same held-out examples and adopted only if it passes the validation gate; otherwise, the current artifact is carried forward to the next epoch.

4 Experiments

Setup. Following the evaluation strategy of SkillOpt (Yang et al., 2026b), we evaluate SkillSpec on GPT–4.1, GPT–5.4 Nano, and GPT–5.4 across six benchmarks: SearchQA (Dunn et al., 2017), SpreadsheetBench (Ma et al., 2024), OfficeQA (Opsahl-Ong et al., 2026), DocVQA (Mathew et al., 2021), LiveMath (He et al., 2026), and ALFWorld (Shridhar et al., 2021). We use each benchmark's training split for optimization, its validation split to accept or reject candidate updates, and its complete test split only for final evaluation. We report success rate per benchmark and the macro average across all six benchmarks. Complete hyperparameters and optimization budgets are reported in Table 4.

Method comparison. We compare against six baselines. The current table reports only completed, protocol-compatible cells and leaves unavailable results blank. **(a) Baseline** executes the target model without an external skill. **(b) GEPA** (Agrawal et al., 2025) reflects on execution trajectories and retains Pareto-efficient prompt candidates. **(c) Trace2Skill** (Ni et al., 2026) distills trajectory-local lessons into reusable skills. **(d) SkillAdam** (Li et al., 2026) uses optimization memory and adaptive update magnitudes. **(e) SkillOpt** (Yang et al., 2026b) iteratively optimizes a persistent skill. **(f) SkillOpt-lite** (Shen et al., 2026) performs file-based skill optimization with Claude Sonnet 5. We additionally report the consensus-gated Phase I output, denoted SkillSpec-I, as the paired reference for the complete Phase II stage.

4.1 Main Results

Performance across models. SkillSpec achieves the highest average success rate for all three target models: 59.33% on GPT–4.1, 48.05% on GPT–5.4 Nano, and 78.27% on GPT–5.4. These scores improve over the corresponding no-skill baselines by absolute margins of 18.85%, 19.10%, and 24.59%, respectively. They also exceed the strongest fully populated external baseline for each model by 8.49% on GPT–4.1 (SkillAdam), 3.27% on GPT–5.4 Nano (SkillOpt-lite), and 1.76% on GPT–5.4 (SkillOpt-lite). At the benchmark level, SkillSpec leads on SearchQA, SpreadsheetBench, and OfficeQA with GPT–4.1; is best or tied-best on OfficeQA, DocVQA, LiveMath, and ALFWorld with GPT–5.4 Nano; and leads on SearchQA, OfficeQA, LiveMath, and ALFWorld with GPT–5.4, where it also ranks second on SpreadsheetBench. Even before specialization, SkillSpec-I achieves a higher overall average than any fully populated external method when averaged across all three target models. This result indicates that multi-intent optimization in Phase I already provides a strong foundation; Phase II specialization further improves performance by organizing the shared backbone into task-appropriate representations.

Table 2: Success rate (%) on the complete test set and six-benchmark macro averages. Improvement is the absolute success-rate difference (%) from the no-skill baseline. SkillSpec-I denotes the consensus-gated Phase I output, and SkillSpec denotes its Phase II specialized result. Best and second-best populated results are bolded and underlined, respectively.

Model	Method	SearchQA	Spreadsheet	OfficeQA	DocVQA	LiveMath	ALFWorld	Average	Improvement
GPT–4.1	Baseline	69.57	36.07	1.16	66.31	25.00	44.78	40.48	–
	GEPA	79.71	35.36	12.79	80.75	29.03	64.93	50.43	+9.95
	Trace2Skill	75.50	41.79	13.95	78.88	33.06	44.78	47.99	+7.51
	SkillAdam	80.93	46.79	10.47	82.62	29.03	55.22	50.84	+10.36
	SkillOpt	79.93	46.43	11.05	73.26	29.03	52.99	48.78	+8.30
	SkillOpt-lite	75.71	48.93	12.21	69.79	28.23	64.93	49.96	+9.48
	SkillSpec-I	81.86	47.50	15.70	**84.76**	**37.90**	**77.61**	57.56	+17.08
	SkillSpec	**82.43**	**50.36**	**27.91**	83.69	35.48	76.12	**59.33**	**+18.85**
GPT–5.4 Nano	Baseline	56.36	30.71	1.74	29.14	27.42	28.36	28.95	–
	GEPA	67.36	30.71	5.81	64.44	26.61	39.55	39.08	+10.13
	Trace2Skill	60.64	31.79	5.81	64.44	33.06	32.84	38.10	+9.15
	SkillAdam	66.57	36.79	6.40	64.71	21.77	34.33	38.43	+9.48
	SkillOpt	57.43	39.64	5.81	65.51	50.81	33.58	42.13	+13.18
	SkillOpt-lite	64.00	41.43	6.98	**66.31**	45.16	44.78	44.78	+15.83
	SkillSpec-I	**70.71**	**44.29**	6.98	**66.31**	50.81	44.78	47.31	+18.36
	SkillSpec	70.29	43.21	**7.56**	**66.31**	**52.42**	**48.51**	**48.05**	**+19.10**
GPT–5.4	Baseline	78.50	42.86	13.37	78.61	37.10	71.64	53.68	–
	GEPA	84.43	53.93	65.12	80.21	42.74	84.33	68.46	+14.78
	Trace2Skill	84.86	41.79	63.95	**90.91**	50.81	77.61	68.32	+14.64
	SkillAdam	84.50	67.14	54.65	**90.91**	47.58	83.58	71.39	+17.71
	SkillOpt	84.00	71.43	59.88	90.37	55.65	89.55	75.15	+21.47
	SkillOpt-lite	85.21	**77.50**	63.37	89.84	58.06	85.07	76.51	+22.83
	SkillSpec-I	83.71	75.71	62.79	88.50	54.84	88.06	75.60	+21.92
	SkillSpec	**86.36**	76.79	**66.28**	88.77	**58.87**	**92.54**	**78.27**	**+24.59**

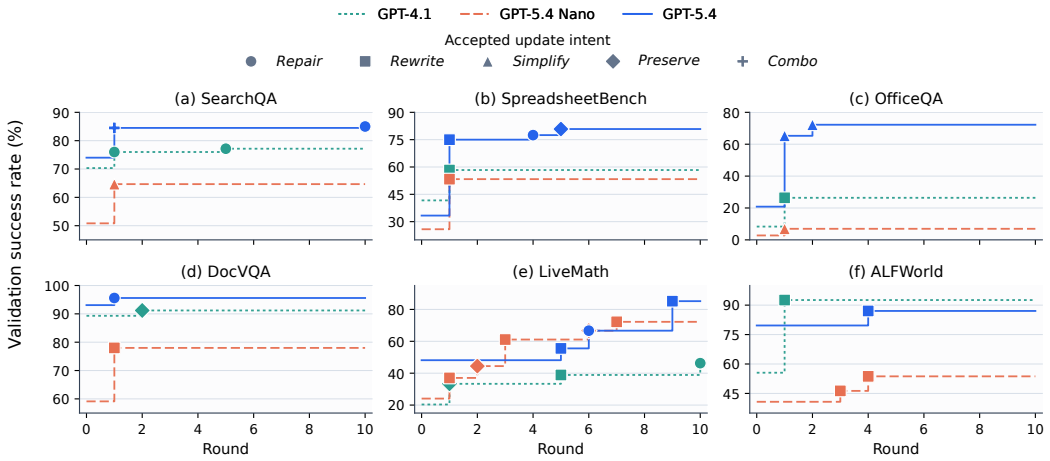


Figure 3: Optimization trajectories during Phase I across six benchmarks for GPT–4.1, GPT–5.4 Nano, and GPT–5.4. All trajectories use 10 rounds and three validation repeats.

Paired specialization gain. Across the 18 paired model–benchmark settings, SkillSpec outperforms its exact SkillSpec-I backbone in 12, ties in one, and underperforms in five. Averaged within each model, Phase II yields absolute gains of $+1.78\%$, $+0.74\%$, and $+2.67\%$ for GPT–4.1, GPT–5.4 Nano, and GPT–5.4, respectively; the average gain across all settings is $+1.73\%$. Since each pair shares the same starting checkpoint and provenance, these results capture the overall effect of Phase II, including both representation selection and continued optimization.

Table 3: Representation ablation on GPT–5.4 Nano with a shared Phase I backbone. Boxes indicate PSS/RSS-selected branches. Avg. is the unweighted mean across six benchmarks.

Variant	SearchQA	Spreadsheet	OfficeQA	DocVQA	LiveMath	ALFWorld	Avg. (6)
Phase I, equal compute	66.57	46.79	4.07	65.24	45.97	51.49	46.69
Phase II, Flat only	68.50	46.79	4.65	66.58	41.94	52.99	46.91
Phase II, Graph only	67.86	43.57	6.98	67.11	45.97	55.97	47.91
Phase II, Hybrid only	66.93	49.29	5.81	62.30	53.23	57.01	49.10
PSS (ψ)	0.428	0.925	0.619	0.411	0.998	0.778	–
RSS (ρ)	0.918	0.837	0.499	0.653	1.000	0.889	–
Router	Flat	Hybrid	Graph	Flat	Hybrid	Hybrid	–
SkillSpec	68.50	49.29	6.98	66.58	53.23	57.01	50.27

4.2 Phase I Optimization Dynamics

Figure 3 suggests that accepted edits shift from behavioral contract formation to local constraint refinement as the incumbent matures. Early trajectories are shaped primarily by *rewrite*, which accounts for 15 of 32 accepted updates across all three models and five benchmarks, and by the less frequent *simplify*, whose four accepted updates all occur in rounds 1–2. Later gains increasingly come from targeted changes: four of seven accepted *repair* updates occur after round 4, while *preserve* appears mainly on LiveMath and the already strong GPT–4.1 DocVQA trajectory. Appendix B.1–B.4 illustrates the same shift: the early cases restructure workflows or output contracts, whereas the later cases narrow or clarify specific exceptions.

4.3 Ablation Studies

Table 3 reveals distinct Phase I sensitivity profiles. **SearchQA** and **DocVQA** share a low-PSS, high-RSS profile, indicating limited support for beneficial explicit ordering but evidence of harmful interference. **OfficeQA**, in contrast, involves retrieving relevant documents, extracting evidence, and performing calculations that depend on the extracted values. Unlike answering from supplied snippets in SearchQA or a document image in DocVQA, this workflow introduces dependencies between intermediate steps, consistent with its higher PSS and the use of Graph to encode their order. **SpreadsheetBench**, **LiveMath**, and **ALFWorld** share a high-PSS, high-RSS profile in the Phase I estimates reported in Table 3, motivating both process structure and interference control. Dependencies between spreadsheet operations, mathematical reasoning steps, or state-dependent actions in ALFWorld motivate an ordered core, while sensitivity to interfering guidance motivates keeping auxiliary rules modular.

Using only Phase I telemetry, SkillSpec selects the best-performing representation on five of the six benchmarks. On the remaining benchmark, DocVQA, it selects Flat, which trails Graph by just 0.53% (66.58% versus 67.11%). Across all six benchmarks, SkillSpec's six-benchmark macro-average is 50.27%, above fixed Flat, Graph, and Hybrid by absolute margins of 3.36%, 2.36%, and 1.17%, respectively; all aggregates are computed from the displayed per-benchmark values before rounding. The six-benchmark comparison supports adapting the representation to Phase I sensitivity rather than imposing a fixed structure, without consulting test outcomes during routing.

5 Conclusion

We introduced SkillSpec, a two-phase framework for improving LLM agents through skill optimization without updating model weights. Phase I uses consensus-gated evolution to refine reusable knowledge, while Phase II uses trajectory-derived sensitivity signals to specialize its organization into flat, graph, or hybrid representations. Across six benchmarks, SkillSpec improves the average success rate over SkillOpt by absolute margins of 10.55%, 5.92%, and 3.12% on GPT–4.1, GPT–5.4 Nano, and GPT–5.4, respectively. These findings suggest that effective skill optimization requires attention not only to what knowledge an agent retains, but also to how that knowledge is structured for execution.

Ethics Statement

SkillSpec studies skill optimization for LLM agents on established benchmarks. Learned skills may inherit errors or biases from the underlying models and data, and improved benchmark performance does not guarantee safe behavior in deployment. For agents that interact with files or external tools, deployment should include appropriate access controls, human oversight for consequential actions, and safeguards against unintended modifications or disclosure of sensitive information. Use of the models and datasets should follow their applicable licenses and terms.

AI Use Statement

Generative AI tools assisted with identifying potentially relevant literature and refining search queries for related work. They also helped improve the grammar, clarity, and readability of the manuscript. The authors take responsibility for the final content of the paper.

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. *arXiv preprint arXiv:2507.19457*, 2025.

Zachary Blumenfeld and Jim Webber. Aip: A graph representation for learning and governing agent skills. In *Agents+Graphs Workshop at VLDB*, 2026. URL https://arxiv.org/abs/2606.04781.

Matthew Dunn, Levent Sagun, Mike Higgins, V Ugur Guney, Volkan Cirik, and Kyunghyun Cho. Searchqa: A new q&a dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Linyang He, Qiyao Yu, Hanze Dong, Baohao Liao, Xinxing Xu, Micah Goldblum, Jiang Bian, and Nima Mesgarani. Livemathematicianbench: A live benchmark for research-level mathematical reasoning with proof sketches, 2026. URL https://arxiv.org/abs/2604.01754.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Gaoyuan Li, Meihao Fan, Yizhe Liu, Shaolei Zhang, Ju Fan, Siyi Wang, Jiaheng Hou, Xudong Weng, Honghan Tian, and Zang Li. Skilladam: Stable and efficient skill evolution for agents. *arXiv preprint arXiv:2609.08944*, September 2026. URL https://arxiv.org/abs/2609.08944.

Dawei Liu, Zongxia Li, Hongyang Du, Xiyang Wu, Shihang Gui, Yongbei Kuang, and Lichao Sun. Graph-of-skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026a. URL https://arxiv.org/abs/2604.05333. Accepted at EMNLP 2026.

Yuxuan Liu, Zhaochen Su, Lingyun Xie, Yuhao Zhang, Qing Zong, Jiahe Guo, Zhongwei Xie, Yiyan Ji, Yauwai Yim, Hongyu Luo, Xiyu Ren, Ruan Chenyu, Haoran Li, and Yangqiu Song. Skillrevise: Improving llm-authored agent skills via trace-conditioned skill revision. *arXiv preprint arXiv:2606.01139*, 2026b. URL https://arxiv.org/abs/2606.01139.

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.

Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pp. 2200–2209, 2021.

Qirui Mi, Zhijian Ma, Mengyue Yang, Haoxuan Li, Yisen Wang, Haifeng Zhang, and Jun Wang. Skill-pro: Learning reusable skills from experience via non-parametric ppo for llm agents. *arXiv preprint arXiv:2602.01869*, 2026. URL https://arxiv.org/abs/2602.01869. ICML 2026 Spotlight.

Jingwei Ni, Yihao Liu, Xinpeng Liu, Yutao Sun, Mengyu Zhou, Pengyu Cheng, Dexin Wang, Erchao Zhao, Xiaoxi Jiang, and Guanjun Jiang. Trace2skill: Distill trajectory-local lessons into transferable agent skills. *arXiv preprint arXiv:2603.25158*, 2026. URL https://arxiv.org/abs/2603.25158.

Krista Opsahl-Ong, Arnav Singhvi, Jasmine Collins, Ivan Zhou, Cindy Wang, Ashutosh Baheti, Owen Oertell, Jacob Portes, Sam Havens, Erich Elsen, et al. Officeqa pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

Siru Ouyang, Jun Yan, Yanfei Chen, Rujun Han, Zifeng Wang, Bhavana Dalvi Mishra, Rui Meng, Chun-Liang Li, Yizhu Jiao, Kaiwen Zha, Maohao Shen, Vishy Tirumalashetty, George Lee, Jiawei Han, Tomas Pfister, and Chen-Yu Lee. Skillos: Learning skill curation for self-evolving agents. *arXiv preprint arXiv:2605.06614*, 2026. URL https://arxiv.org/abs/2605.06614.

Kang Peng, Zhiwei Zhang, Yichen Zhang, Zezhong Wang, Yiming Du, Geng Tu, Baojun Wang, Bin Liang, Ruifeng Xu, and Kam-Fai Wong. Write, execute, refine: From skill followers to skill optimizers via reinforcement learning from execution feedback. *arXiv preprint arXiv:2608.17587*, August 2026. URL https://arxiv.org/abs/2608.17587.

Libin Qiu, Zhirong Gao, Junfu Chen, Yuhang Ye, Liangyu Li, Weizhi Huang, Xiaobo Xue, Wenkai Qiu, and Shuo Tang. Autorefine: Compiling trajectories into validated typed agent artifacts. *arXiv preprint arXiv:2601.22758*, 2026. URL https://arxiv.org/abs/2601.22758.

Yash Priya Shastri, Anand Eswaran, Adnan Qidwai, Pankaj Thorat, and Sachin Joshi. Bonsai: Evolvability-guided tree search over skills. *arXiv preprint arXiv:2608.07056*, August 2026. URL https://arxiv.org/abs/2608.07056.

Yifei Shen, Bo Li, and Xinjie Zhang. Skillopt-lite: Better and faster agent self-evolution via one line of vibe. *arXiv preprint arXiv:2607.03451*, July 2026. URL https://arxiv.org/abs/2607.03451.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. {ALFW}orld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

Yash Vishe, Rohan Surana, Xunyi Jiang, Zihan Huang, Xintong Li, Nikki Lijing Kuang, Tong Yu, Ryan A. Rossi, Jingbo Shang, Julian McAuley, and Junda Wu. Skill-r1: Agent skill evolution via reinforcement learning. *arXiv preprint arXiv:2605.09359*, 2026. URL https://arxiv.org/abs/2605.09359.

Chenxi Wang, Zhuoyun Yu, Xin Xie, Wuguannan Yao, Runnan Fang, Shuofei Qiao, Kexin Cao, Guozhou Zheng, Xiang Qi, Peng Zhang, et al. Skillx: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

Ruiyao Xu, Tiankai Yang, and Wei-Chieh Huang. Hyperskill: Self-evolving llm agents via hypergraph-structured skill memory. *arXiv preprint arXiv:2608.16114*, 2026. URL https://arxiv.org/abs/2608.16114.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

Min Yang, Jinghua Piao, Xu Xia, Xiaochong Lan, Jiaju Chen, Yongshun Gong, and Yong Li. Skillmaster: Toward autonomous skill mastery in llm agents. *arXiv preprint arXiv:2605.08693*, 2026a. URL https://arxiv.org/abs/2605.08693.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026b.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The eleventh international conference on learning representations*, 2023.

A HYPERPARAMETERS AND CONFIGURATION

Table 4: Experimental and specialization settings used for the complete evaluation matrix.

Parameter	Value
Target models	GPT–4.1 / GPT–5.4 Nano / GPT–5.4
Optimization data	Official training and validation splits
Final evaluation	Complete official test split
Primary metric	Hard accuracy
Aggregation	Macro average across six benchmarks
Failure accounting	Agent, setup, and timeout failures count as incorrect
Initial skill	Method- and cell-specific; reported in Table 2
Phase I rounds / batch size	10 / 40
Candidate profiles	*repair*, *preserve*, *simplify*, *rewrite*
Combo candidate	Optional, from compatible candidates
Primary comparison	Paired on identical task IDs
Finalists N_f	2
Validation execution seeds	2; screening seed reused in confirmation
Minimum pooled net gain G_{pool}	3 tasks
Minimum per-evaluation gain $G_{\min}$	0 tasks
Phase II epochs / batch size	4 / 35
PSS/RSS estimator	Candidate-level ridge, $\ell_2 = 1$
Estimator controls	Round, candidate profile, change ratio, quadratic process/interference
Estimator eligibility	Changed, valid candidates with a recorded screening gain
Annotation evidence	Candidate skill diffs
Annotation scale	Integer process/interference strengths in $\{0, 1, 2, 3\}$
Round-cluster bootstrap	2,000 samples, seed 42
PSS threshold ζ_ψ	0.50
RSS threshold ζ_ρ	0.50
Flat policy	$\psi \leq \zeta_\psi$
Graph policy	$\psi > \zeta_\psi$ and $\rho \leq \zeta_\rho$
Hybrid policy	Otherwise
Phase I / telemetry optimizer	Claude Sonnet 5 via Copilot CLI
Phase II optimizer	Target model

Representation decisions use only sanitized Phase I telemetry and the accepted backbone. The telemetry optimizer extracts structured observations; fixed code computes PSS, RSS, and support from them. Benchmark and task identifiers are prohibited as explicit decision features, and no test-set labels or outcomes are provided. This restriction removes direct identifiers but does not imply that semantic task characteristics are absent from the skill or trajectory.

B REPRESENTATIVE PHASE I INTENT UPDATES

Below, we present representative diffs for one accepted update from each intent. Added and removed lines are highlighted in green and red, respectively, while unchanged content is omitted. Because *rewrite* can reorganize most of a skill, we show its principal edits rather than the full diff. To make the before–after contrast explicit, we render the append-only *preserve* clarification as a normalized replacement: the original rule is shown in red and the accepted rule, including the added clarification, in green. For display, Unicode punctuation is normalized to ASCII and percentage-unit wording in the *simplify* diff is rendered as %. No other wording is changed. Reported validation scores are mean hard accuracies over three repeats.

B.1 *Rewrite*

GPT–5.4 SpreadsheetBench, round 1.

Representative Skill Diff

```
@@ Common Workflow / formula handling @@
```

```
-4.  Confirm the target cells/range contain the expected values.
+4.  Confirm the target cells/range contain the expected values by re-opening the
+ saved output file and printing cell.value for every requested cell.
+ A blank/None result means the task is not done yet.
+
+## Formulas vs.  Computed Values
+
+openpyxl never evaluates formulas.  If you set cell.value = "=SUM(A1:A10)",
+it writes only the formula string; it does not compute or cache a result.
+
+- Treat "write a formula for X" as "make the correct result appear in the
+ target cell(s)", not as "the cell must literally contain an = string".
+- Reproduce the formula logic in Python and write the resulting literal value
+ with the correct type into every target cell.
+- Apply the operation to every row or column covered by the instruction.
+- Only leave an actual formula string when only its text will be inspected and
+ no computed value is required.
+
+## Cell Content Discipline
+
+- A target cell must contain only the requested value, never explanatory prose.
+- Strip surrounding labels unless the instruction asks to retain them.
+- Reopen the output and verify that requested cells are not blank or None.
```

Before: The skill only required checking that target cells contained the expected values.
After: The rewrite introduced an explicit reopen-and-verify procedure, distinguished formulas from computed values, and required literal, nonblank outputs in every requested cell.

Validation: $33.3\% \rightarrow 75.0\%$.

B.2 *Simplify*

GPT–5.4 OfficeQA, round 2.

Skill Diff

```
-- Original Skill
+++ Accepted Skill
@@ -14,3 +14,4 @@
 - Return the final answer only after one last consistency check against the retrieved
evidence.
 - Copy the final answer from a checked value, not from an unverified intermediate guess.
- Inside the final answer tags, state only the bare value the question asks for (plus a
unit/word only if the requested format explicitly calls for one) -- no restated reasoning,
source citations, intermediate steps, or explanatory sentences.
+- Inside the final answer tags, state only the bare value the question asks for -- no
restated reasoning, source citations, intermediate steps, or explanatory sentences.
+- Do not append a unit, label, or direction word (e.g., "%," "inflow," "outflow") just
because the question's narrative describes the metric that way; a phrase like "report in
millions" or "in %" describes the number's scale, not literal text to add.  Include such a
word only when the output-format instruction explicitly asks the answer string itself to
contain it (e.g., a named date format).
```

Before: A single rule combined the bare-value requirement with an underspecified exception for units or words.
After: The update separated the two concerns and clarified that narrative scale cues are not output tokens unless the requested format explicitly includes them.

Validation: $65.3\% \rightarrow 72.2\%$.

B.3 *Repair*

GPT–5.4 SpreadsheetBench, round 4.

Skill Diff

```
-- Original Skill
+++ Accepted Skill
@@ -40,9 +40,13 @@
 formula the user manually copies down row-by-row must be replicated in full for all rows
 it applies to; don't stop after solving the first cell.
- Only leave an actual `=` formula string in a cell when the instruction is explicitly about
- formula authoring/repair *and* nothing will need to read that cell's computed value (e.g.
- purely cosmetic/example formulas).  When in doubt, prefer writing the literal computed
- result -- it satisfies both "the formula works" and "the value is correct", while a bare
- formula string alone satisfies neither once opened by a non-recalculating reader.
+- Requests phrased as "I need a formula for X", "write a formula that...", or "how do I fix/
+ adjust this formula" describe the desired *result*, not a literal instruction to store an
+ `=` string -- treat them the same as any other request and write the literal computed value.
+ Do **not** leave a bare `=` formula string in a target cell just because the instruction
+ uses the word "formula":  a reader that never runs a spreadsheet engine cannot distinguish
+ an uncomputed formula cell from a blank one, so a bare formula string reads back as empty
+ and fails verification exactly like leaving the cell untouched.  Only keep a literal `=`
+ string when the instruction asks you to inspect/repair the formula text itself (e.g. "what's
+ wrong with this formula") and no cell's resulting value is referenced anywhere in the task.
 - Double-check edge cases the instruction calls out explicitly (blanks, `#N/A`, zero values,
 duplicates, date/leap-year boundaries) -- these are frequently the exact cells checked.
```

Before: The skill broadly allowed literal formula strings for formula-authoring or repair requests.
After: The repair narrowed that exception to tasks that inspect formula text and otherwise required the literal computed value.

Validation: $75.0\% \rightarrow 77.5\%$.

B.4 *Preserve*

GPT–5.4 SpreadsheetBench, round 5.

Skill Diff

```
-- Original Skill
+++ Accepted Skill
@@ Formula-text exception @@
- Requests phrased as "I need a formula for X", "write a formula that...", or "how do I fix/
```

```
- adjust this formula" describe the desired *result*, not a literal instruction to store an
- `=` string -- treat them the same as any other request and write the literal computed
value.
- Do **not** leave a bare `=` formula string in a target cell just because the instruction
- uses the word "formula":  a reader that never runs a spreadsheet engine cannot distinguish
- an uncomputed formula cell from a blank one, so a bare formula string reads back as empty
- and fails verification exactly like leaving the cell untouched.  Only keep a literal `=`
- string when the instruction asks you to inspect/repair the formula text itself (e.g.
"what's
- wrong with this formula") and no cell's resulting value is referenced anywhere in the
task.
+- Requests phrased as "I need a formula for X", "write a formula that...", or "how do I
fix/
+ adjust this formula" describe the desired *result*, not a literal instruction to store an
+ `=` string -- treat them the same as any other request and write the literal computed
value.
+ Do **not** leave a bare `=` formula string in a target cell just because the instruction
+ uses the word "formula":  a reader that never runs a spreadsheet engine cannot distinguish
+ an uncomputed formula cell from a blank one, so a bare formula string reads back as empty
+ and fails verification exactly like leaving the cell untouched.  Only keep a literal `=`
+ string when the instruction asks you to inspect/repair the formula text itself (e.g.
"what's
+ wrong with this formula") and no cell's resulting value is referenced anywhere in the
task.
+ If the instruction names or implies where the result belongs (e.g.  "it should be placed
+ into cell K6", "fill out row 11", a specific `Sheet!Cell` reference), that counts as
+ referencing the cell's resulting value -- the debug-only exception above does **not**
apply
+ even when the wording also asks you to diagnose/explain the formula, so compute and write
+ the literal value there.  Never leave such a target cell as `None`/blank as a fallback.
```

Before: The formula-text exception did not explicitly cover diagnostic requests that also named a destination cell.
After: The preserved rule was augmented to treat any named destination as requiring its computed value and to forbid a blank fallback.

Validation: $77.5\% \rightarrow 80.8\%$.

C Representative Specialization Content

Table 5 shows condensed examples of procedural content that may appear in the learned Flat, Graph, or Hybrid specialization. These are not benchmark-specific prompts written by hand, and they do not imply test-time selection among representation families. SkillSpec selects one family per model–benchmark cell before specialization training.

D PSS/RSS Annotation and Estimation Details

This section specifies the candidate-level annotation and frozen ridge estimator used for the sensitivity scores in Table 3. It complements Equations (6) and (7): the language model supplies evidence-grounded labels, while numerical code fits the outcome model and computes the scores.

Table 5: Representative specialization content. Text is lightly abbreviated for readability.

Benchmark	Content role	Purpose	Example content
LiveMath	Procedural specialization	Distinguish theorem-style statements with subtle strength differences.	*Before choosing, rewrite each option as a formal claim: hypotheses, quantifiers, conclusion strength, and endpoint cases. Rank options by logical implication, not by thematic similarity. Do not stop at the first true-looking statement; choose the strongest statement actually proved.*
DocVQA	Evidence specialization	Preserve literal values and provenance from a document image.	*Ground the answer in visible document evidence. Prefer the exact string, unit, date, or entity shown in the source. If multiple nearby values appear, compare labels and table headers before answering.*
OfficeQA	Backbone-anchored process	Support multi-turn file/search tasks that require verified evidence.	*Search broadly, then narrow by entity/date/table. Read the source document before answering. If evidence is missing, reformulate the query. Track provenance and answer only from retrieved evidence; do not answer from memory.*
Spreadsheet Bench	Transformation specialization	Preserve unrelated workbook state during a requested edit.	*Use the full workbook, not only the preview. Iterate over all actual rows and sheets. Preserve unrelated cells and formulas. Write outputs to the requested sheet/cell range and validate against the requested answer position.*
ALFWorld	Backbone-anchored process	Preserve state across long action sequences and recovery steps.	*Track the goal, current room, inventory, and object state. Explore systematically; after a failed action, inspect the observation and choose a recovery action. Use containers and object affordances explicitly; stop only after the goal is satisfied.*

D.1 Annotation evidence and rubric

The unit of annotation is a candidate's edit relative to its round's incumbent skill, not the benchmark as a whole. Each record is identified by an `observation_id` and linked to a unified skill diff. The diff-only annotation workspace exposes `observation_id`, `candidate_diff`, and `diff_file`; numerical screening gains and candidate metadata are joined separately by the scorer. The annotator is instructed not to use or infer benchmark, dataset, environment, or sample identity, and to ignore intent names, acceptance decisions, optimizer summaries, tool counts, and the outer interaction format.

A process annotation requires an added or strengthened ordered prerequisite, state handoff, or conditional dependency in the solution procedure. Generic advice to "reason step by step" is insufficient. An interference annotation requires added guidance that is visibly duplicated, conflicting, irrelevant to the surrounding skill, or disproportionately overlong. A negative screening gain is not itself evidence of interference. The same four-level strength scale is applied independently to both labels, as detailed in Table 6.

The permitted interference modes are `duplicated`, `conflicting`, `irrelevant`, and `overlong`; more than one may apply. Removing redundant text does not automatically receive a

Table 6: Candidate-diff annotation rubric. Strength describes the visible edit, not its measured benefit or the intrinsic complexity of the task.

Score	Process structure	Interference
0	No grounded addition or strengthening of a dependency.	No grounded addition of interfering guidance.
1	Weak or local prerequisite, state handoff, or conditional dependency.	Weak or local duplication, conflict, irrelevance, or excessive detail.
2	Clear dependency in the revised solution procedure.	Clear interfering guidance in the revision.
3	Dependency structure is a dominant feature of the edit.	Interfering guidance is a dominant feature of the edit.

positive interference label: the instruction concerns what the candidate adds. If the diff is missing or supplies no grounded evidence, the prompt requests zero strengths and an empty mode list.

The following illustrates the output schema, not a scored experimental observation:

```
{
  "trajectory_observations": [
    {
      "evidence_id": "<observation_id copied verbatim>",
      "evidence": "<specific evidence from the skill diff>",
      "process_structure_strength": 0,
      "interference_strength": 0,
      "interference_modes": []
    }
  ]
}
```

There must be exactly one observation per source candidate, with a unique matching identifier and a nonempty evidence string. Both strengths must be integers from 0 to 3; nonempty interference modes require a positive interference strength. The annotator does not output PSS, RSS, confidence, or a representation choice. The validated annotation runner allows up to three generation attempts when output fails JSON or schema validation, rather than accepting malformed output or heuristically repairing it.

D.2 Eligibility, regression, and round bootstrap

Annotations are joined to recorded candidate outcomes by identifier. A candidate enters the regression if and only if it changes the incumbent, passes patch validation, and has a recorded `screen_net_gain`. Zero-gain and negative-gain candidates remain eligible, including candidates rejected by the update gate. Unchanged or invalid candidates and candidates without a screening outcome are excluded. The estimator uses screening outcomes, not pooled confirmation gains or final test scores.

For Equation (5), the implementation compares the two texts after `splitlines()` using `difflib.SequenceMatcher` with `autojunk=False`. Each non-equal block contributes the larger of its before and after line counts. The denominator is the larger total line count, with a minimum of one for the empty-text case.

For each eligible candidate, the numerical features are

$$(c_{t,j},\, p_{t,j},\, p_{t,j}^2,\, r_{t,j},\, r_{t,j}^2).$$

Each column is centered and divided by its population standard deviation; a zero standard deviation is replaced by one. The design matrix also includes an intercept and one-hot indicators for round and candidate profile, omitting one reference level for each categorical variable. There is no process–interference interaction term. With X denoting this design and y the recorded screening gains, the fitted coefficients are

$$\widehat{\beta} = (X^\top X + \Lambda)^{-1} X^\top y, \qquad \Lambda = \mathrm{diag}(0, 1, \ldots, 1).$$

Algorithm 2 Frozen candidate-level PSS/RSS estimator

Input: Joined eligible candidate rows $\mathcal{D}$; bootstrap count $N = 2000$, random seed 42
Output: PSS ψ, RSS ρ, eligible count ν

```
Let R be the distinct rounds in D; set ν = |D|
Fit the full-sample ridge model and compute both contrasts
for n = 1, ..., N do
    Draw |R| rounds independently with replacement from R
    Copy all eligible rows from each drawn round into D^(n)
    Assign each drawn block a new round label, including repeated draws of the same source round
    Rebuild categorical levels and numerical standardization on D^(n)
    Refit ridge and compute Δ_P^(n), Δ_R^(n)
end for
Compute ψ, ρ using Equation (7)
return ψ, ρ, ν
```

Thus the ridge penalty is one for every non-intercept coefficient, including categorical controls. The implementation solves the linear system directly rather than forming the inverse.

To evaluate a contrast, the scorer sets all rows' process and interference strengths to the probe values, recomputes their squares, and retains each row's edit size and categorical controls. It applies the fitted standardization, averages the predictions, and computes the extrema in Equation (6). The full-sample fit yields Δ_{P} and Δ_{R}; the bootstrap yields the positive-contrast proportions.

Sampling uses NumPy's `default_rng(42)` over sorted round labels. All eligible candidates from a drawn round stay together; candidates are not independently resampled. The validated scoring runner checks that the full-sample fit and all 2,000 bootstrap fits succeed, rather than silently reporting a proportion over fewer fits. The 10th and 90th percentiles summarize the bootstrap *effect contrasts*, not intervals on PSS/RSS themselves. These resamples reuse existing round blocks and are not 2,000 independent experiments. PSS/RSS summarize fitted effect-sign stability, not the probability that one representation outperforms another.

E A Worked Representation Example from a Shared Backbone

To isolate organization from content, we use the same ALFWorld backbone for all three illustrations below. The source is the frozen seed-42 Phase I artifact `a3_auto_a2/history/round_01_before.md` (SHA-256 prefix `01bd29e825b8`), shared by the ALFWorld ablation branches. Its *Heat & Place* procedure reads:

> Find X → take X → go to microwave → heat X → go to Y → put X.

The same artifact instructs the agent to open closed containers before judging them empty, transform an object before placing it, track progress, avoid repeated-action loops, and choose only admissible actions.

Table 7 re-expresses this source content schematically. Unit names and dependency edges are introduced for exposition; these are not verbatim exports of three learned checkpoints. They illustrate representation semantics, not additional evaluation runs. All three retain the same action-format and admissibility constraints.

Observed execution evidence. Table 8 shows one actual training episode from the archived `round_00_train` batch, with source identifier `train:0028`. Its goal is "heat some egg and put it in countertop." Actions and zero-based step indices are retained verbatim; observations are condensed for readability. The episode contains 12 actions and ends with the environment marking the task complete. It was collected with the textual Phase I skill, not by executing each of the three schematic representations.

Reading the trace through the three representations. The handoff from acquisition at step 6 to heating at step 8, and then delivery at step 11, makes the relevant state dependency concrete. In the Flat illustration, the transformation unit supplies guidance after acquisition without a graph-defined

Table 7: Three schematic organizations of the same archived ALFWorld guidance. The content is held fixed to make structural differences explicit.

Form	Organization of shared guidance
Flat	Independently retrievable units for locating, acquiring, transforming, and delivering the object, alongside exploration and progress/loop guidance. No dependency graph schedules retrieval; a retrieved unit can still state a local prerequisite such as "transform before placing."
Graph	Explicit dependencies connect locate → acquire → transform → deliver. Container inspection refines locating, while state and progress checks provide guidance at the relevant nodes. Edges make prerequisite order explicit.
Hybrid	The locate–acquire–transform–deliver chain forms the ordered core. Container exploration, progress tracking, and loop recovery remain auxiliary modules that can be retrieved when relevant, rather than expanding every core step with all guidance.

Table 8: Archived training execution illustrating state dependencies in the shared backbone. This is one observed episode, not a three-way representation comparison.

Step	Action	Observed result (condensed)
0	`go to fridge 1`	The fridge is closed.
1	`open fridge 1`	Several objects are visible, but no egg.
2	`go to microwave 1`	The microwave is closed.
3	`open microwave 1`	The microwave is empty.
4	`go to fridge 1`	The same fridge contents are observed again.
5	`go to countertop 1`	An egg is visible among other objects.
6	`take egg 1 from countertop 1`	The agent picks up the egg.
7	`go to microwave 1`	The microwave is open.
8	`heat egg 1 with microwave 1`	The environment confirms heating.
9	`close microwave 1`	The microwave is closed.
10	`go to countertop 1`	The agent reaches the destination.
11	`move egg 1 to countertop 1`	The egg is placed; `done=true`.

retrieval order. In Graph, explicit edges connect those stages. In Hybrid, the same chain belongs to the core, while container exploration or loop-recovery guidance can be supplied separately. These describe how the evidence can be organized, not measured retrieval choices for this episode. The return to the already inspected fridge at step 4 is preserved in the trace: successful completion does not imply perfect adherence to every instruction or optimal action efficiency.