SKILLBACKPROP: TWO-STAGE TEXTUAL BACKPROPAGATION FOR SKILL OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimizing skills for frozen language models requires more than identifying failures. It also requires assigning task loss appropriately and determining how the skill should be revised. Unlike differentiable model parameters, skill instructions do not admit analytic derivatives that can directly guide these decisions. To provide a backward learning pathway for skills, we introduce SKILLBACKPROP, a two-stage textual backpropagation framework for structured skills. SKILLBACKPROP separates backward learning into *failure localization* and *loss-reduction estimation*. The first stage aggregates execution evidence to trace failed outcomes to the responsible parts of the skill. The second measures how candidate revisions affect validation loss and uses the resulting loss-reduction signals to guide skill updates. Together, the two stages provide a backward pathway from task-level loss to targeted modification. We evaluate SKILLBACKPROP on SpreadsheetBench, OfficeQA, SearchQA, and DeepPlanning-Shopping, where it achieves scores of 87.86, 70.24, 86.31, and 69.05, respectively. Across all benchmark and model settings, SKILLBACKPROP matches or outperforms the strongest baseline, with gains of up to 22.86 percentage points. Ablations further show complementary benefits from failure localization and loss-reduction estimation.

1 INTRODUCTION

Language model based agents increasingly carry out tasks that extend beyond a single model response, using tools and passing intermediate results across multiple steps. When such tasks recur, their procedures can be stored as agent skills: reusable artifacts that package instructions, scripts, and reference material for use during execution. Skills therefore provide an external layer for adapting how a frozen model works in a particular domain, while remaining inspectable and editable as text (Anthropic, 2025; Xu & Yan, 2026; Ling et al., 2026).

For skills that guide multi-stage tasks, their use across different stages provides useful information for subsequent improvement. Prompt optimization revises the instructions supplied to one or more model calls, while workflow optimization changes the modules or control flow that organize those calls. A skill used for a long task has a different role: it supplies the relevant guidance as the task moves through an existing process and specifies what one stage must produce for the next. Improving such a skill therefore concerns the procedural guidance distributed across the process. Recent systems take a step in this direction by treating the skill artifact itself as trainable and updating it from execution trajectories and evaluation feedback (Yang et al., 2026; Wang et al., 2026a; Gautam et al., 2026).

This process is often mediated by natural language feedback, which has been described as a *textual gradient* (Yuksekgonul et al., 2025), although recent work questions whether such feedback behaves like a gradient (Melcer et al., 2025). This criticism highlights an implicit step: failure evidence is passed to a revision model that chooses both the location and the content of an edit. For a multi-stage skill, the resulting revision can remain broad, because it is not tied to the stage where the error arose and can alter guidance used across stages. We therefore argue that this path should be made explicit by separating two questions: where in the skill should the update be applied, and whether a proposed revision at that location reduces task-level loss.

This raises a general challenge: *how can improvements be localized within a multi-stage skill so that its guidance helps the model correct recurring inference errors?* Addressing this challenge requires

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.

connecting an observed failure to the stage where the error can be traced in the execution record and aggregating similar failures before modifying skill instructions. The proposed change must then be evaluated in the full process to determine whether it improves task performance. These steps are needed because a revision that repairs one failure can alter later stages or regress cases that already succeed.

We introduce SKILLBACKPROP as a training procedure for this setting. Whereas backpropagation computes a gradient for every trainable parameter in a neural network, textual skill instructions have no corresponding derivative that identifies where they should change or predicts the effect of a revision. SKILLBACKPROP therefore compiles the skill into an execution DAG whose nodes have distinct responsibilities and outputs, and records a trace for each task. For failed tasks, it traces the error backward through the executed stages and aggregates recurring errors across the training batch to identify the stage that should be revised. We call this step *failure localization*. In the second step, *loss-reduction estimation*, candidate revisions at the selected stage are evaluated on validation data, and the resulting changes in task loss estimate the loss reduction produced by each revision. Together, these two steps form the path from task outcomes to a validated change in the relevant part of the skill, which we call *textual backpropagation*:

$$\text{task loss} \xrightarrow{\text{failure localization}} \text{stage selected for revision} \xrightarrow{\text{loss-reduction estimation}} \text{validated skill revision.}$$

Our contributions are:

1. **Two-stage textual backpropagation.** We introduce a training pathway for multi-stage skills that separates two decisions: where a recurring execution error first arises and whether a proposed revision improves the complete process.
2. **Failure localization from execution traces.** We use the existing stages and intermediate outputs of a skill to trace failures through its execution trace. Evidence is aggregated across tasks before a stage is selected for revision, while the model and DAG remain fixed.
3. **Loss-reduction estimation through validation loss.** At the identified coordinate, candidate revisions are applied to the skill and evaluated on validation data. Their changes in validation loss provide loss-reduction signals for estimating useful textual update directions.

2 RELATED WORK

Prompt and skill optimization. Natural language instructions have increasingly been treated as optimizable objects. APE and OPRO search over candidate instructions using task performance as the selection signal (Zhou et al., 2023; Yang et al., 2024), while EvoPrompt and PromptAgent introduce evolutionary or agentic optimization procedures (Guo et al., 2024; Wang et al., 2024b). ProTeGi and TextGrad move toward gradient-inspired optimization by using natural language feedback to describe how textual components should change (Pryzant et al., 2023; Yuksekgonul et al., 2025), while DSPy optimizes modular language programs from downstream objectives (Khattab et al., 2024).

Recent work extends optimization from prompts to persistent policies. GEPA revises textual components from execution feedback, while SkillOpt updates persistent skills using rollout evidence and held-out evaluation (Agrawal et al., 2026; Yang et al., 2026). Related experiential methods accumulate reusable lessons or skills from interaction (Shinn et al., 2023; Zhao et al., 2024; Wang et al., 2024a; 2025b; Klissarov et al., 2025; Zheng et al., 2025; Ni et al., 2026), and recent RL systems learn or retrieve evolving skill libraries (Wang et al., 2026b; Vishe et al., 2026; He et al., 2026; Xia et al., 2026; Tu et al., 2026; Lu et al., 2026). These approaches show that textual policies can improve from experience, but typically use task feedback to generate or select revised policies without first tracing observed failures through the internal responsibility structure of an existing skill.

Structured workflow and agent optimization. ADAS, AFlow, and AgentSquare optimize agent implementations, workflow structures, or modular configurations (Hu et al., 2025; Zhang et al., 2025; Shang et al., 2025). Agent Workflow Memory and LATS instead reuse structured execution experience (Wang et al., 2025a; Zhou et al., 2024). From a credit-assignment perspective, Agent

Lightning decomposes trajectories into credit-bearing transitions for parameter-space reinforcement learning (Sutton & Barto, 2018; Luo et al., 2025). Our setting uses the workflow topology as a fixed responsibility structure for identifying where skill updates should be applied, while optimizing only the textual skill itself.

Natural Language Feedback and Credit Assignment. TextGrad interprets natural language feedback propagated through a computation graph as a textual analogue of gradients (Yuksekgonul et al., 2025). However, recent work questions whether such feedback actually behaves like a gradient in the optimization sense (Melcer et al., 2025). In our setting, linguistic feedback may suggest where or how a skill should change, but it does not by itself establish a loss-reducing direction.

Our approach makes this form of credit assignment explicit and execution-aligned: observed failures are first traced to the responsible part of the skill. We refer to this step as *failure localization*. Candidate revisions at the identified coordinate are then evaluated on validation tasks, and their effects on validation loss determine which revision is retained. In this way, SKILLBACKPROP separates failure localization from revision selection within an explicit backward learning pathway.

3 METHOD

3.1 PROBLEM FORMULATION

Execution-aligned skill structure. We consider a frozen language model f_ϕ with model parameters ϕ and an external structured skill. The model parameters remain fixed throughout training; the learnable object is the textual content of the skill.

For complex skills, we compile the task procedure into a staged execution structure according to distinct execution responsibilities. We represent this structure as an operator DAG $G = (V, E)$, where each node $v \in V$ is an operator responsible for one stage of the task, and each edge $(u, v) \in E$ specifies the output schema through which the result of u is passed to v.

Each operator has three parts: a task description, a set of rules, and a set of sub-skills. At optimization step t, let $T_{t,v}$, $R_{t,v}$, and $S_{t,v}$ denote their current contents. The task description defines the objective and completion requirement of the operator; rules provide guidance and constraints; and sub-skills describe procedures for specific types of cases. An operator may initially have no sub-skills, so $S_{t,v}$ can be empty, and new sub-skills can be added during optimization. For compactness, we group the skill content associated with operator v as

$$\theta_{t,v} = (T_{t,v}, R_{t,v}, S_{t,v}).$$

The complete skill at step t is then $\Theta_t = \{\theta_{t,v}\}_{v \in V}$. Because each $\theta_{t,v}$ is attached to a distinct operator, the nodes in V provide the execution coordinates used for failure localization. During training, f_ϕ, the DAG topology G, and the output schemas remain unchanged, while only Θ_t is updated.

Backpropagation for skill optimization. In neural network training, parameters determine how information is transformed through successive layers during the forward pass. The final output produces a loss, and backpropagation applies the chain rule through these transformations to compute its gradient with respect to the parameters; the negative gradient gives the locally steepest descent direction. A multi-stage skill has a corresponding organization: the skill content attached to each operator guides its corresponding stage of task execution, and the execution trace records which rules and sub-skills were used, the actions they guided, and the outcomes that followed. The final task evaluation produces a task loss, but the skill content attached to each operator admits no derivatives through which that loss can be propagated.

SKILLBACKPROP constructs a backward attribution pathway from execution evidence. Failure localization aggregates recurring trace evidence, weighted by task loss, to identify the operator and its associated skill content most consistently implicated in observed failures. Loss-reduction estimation then evaluates candidate revisions at that location and selects the revision with the largest measured reduction in validation loss. We refer to this process, which connects task loss to a localized and validated direction for skill revision, as *textual backpropagation*.

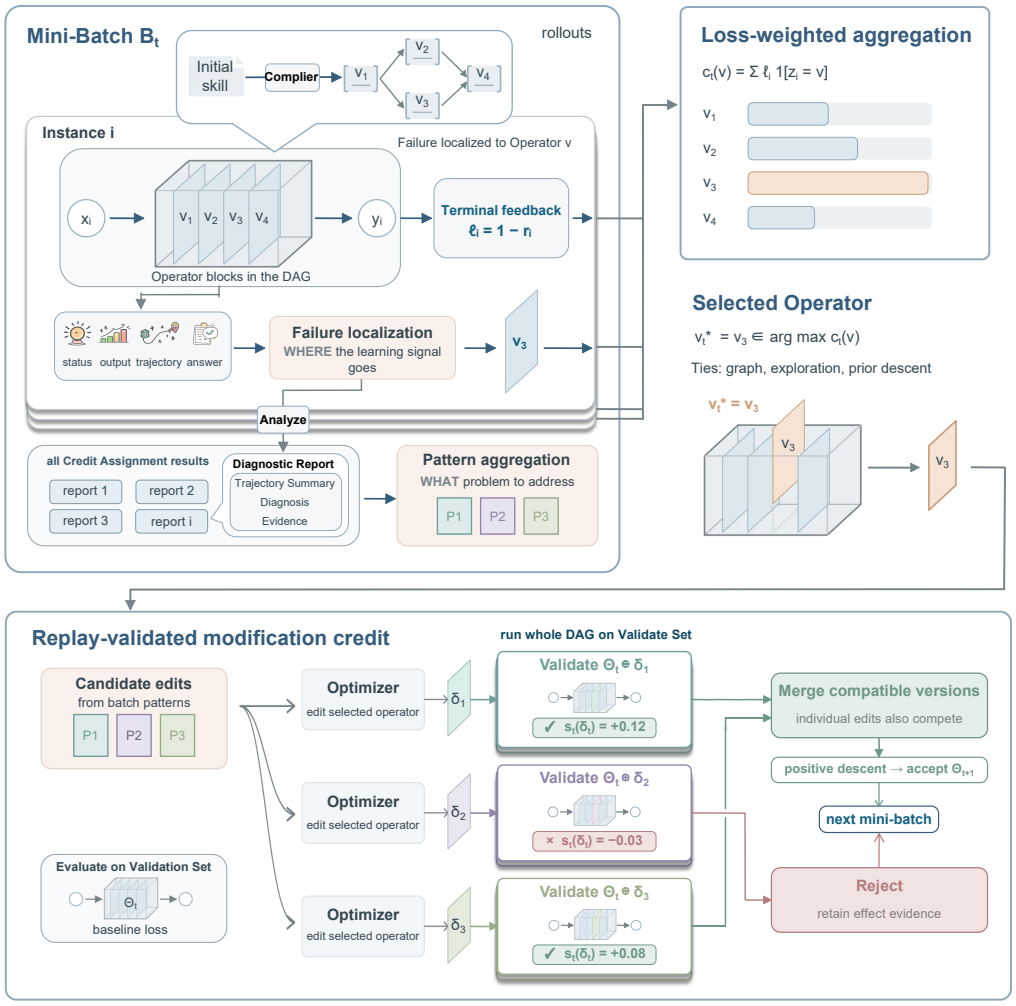

Figure 1: Overview of the SKILLBACKPROP training pipeline. Forward execution produces execution traces and terminal losses. The first backward stage traces each failure to the instructions of one operator; the second evaluates candidate revisions on validation tasks and retains updates with positive loss-reduction signals.

3.2 FORWARD PASS AND EXECUTION TRACE

Let D_{tr}, D_{val}, and D_{test} denote disjoint training, validation, and test sets, respectively, and let $\mathcal{B}_t \subseteq D_{\text{tr}}$ denote the training mini-batch at optimization step t. Each instance $i \in \mathcal{B}_t$ contains an input x_i and evaluation metadata m_i.

We use Exec to denote skill execution and μ to denote the task evaluator. Under the frozen model f_ϕ, fixed graph G, and current skill Θ_t, the forward pass produces

$$(y_i, \tau_i) = \text{Exec}(x_i; f_\phi, G, \Theta_t), \qquad r_i = \mu(y_i, m_i) \in [0, 1], \qquad \ell_i = 1 - r_i. \tag{1}$$

Here y_i is the final task output, τ_i is the observable execution trace, r_i is the terminal task score, and ℓ_i is the corresponding task loss.

The trace τ_i records the operators executed along the dependencies in G, together with their intermediate decisions, model and tool observations, generated artifacts, activated rules, and loaded sub-skills. It therefore connects the terminal task loss ℓ_i to the sequence of stage outputs and skill instructions that produced the final result.

D_{tr} supplies the execution traces used to generate updates. A validation subset $\mathcal{V} \subseteq D_{\text{val}}$ is used to evaluate candidate revisions, while D_{test} is reserved for final evaluation.

3.3 FAILURE LOCALIZATION

The first backward stage determines *where* an observed failure arose. For each failed instance i, failure localization traces the failure backward through the executed dependencies in G. Starting from the final outcome, it examines the behavior and intermediate output of each executed operator to identify the earliest evidence-supported divergence and the instructions of that operator associated with the failure. This backward tracing assigns the instance to a responsible operator $z_i \in V$.

We then aggregate these instance-level assignments according to their realized task losses:

$$c_t(v) = \sum_{i \in \mathcal{B}_t} \ell_i \, \mathbf{1}[z_i = v], \qquad v_t^* \in \arg\max_{v \in V} c_t(v). \tag{2}$$

where $\mathbf{1}[\cdot]$ is the indicator function. Here, $c_t(v)$ is the accumulated task-loss weight of failures assigned to execution coordinate v, and v_t^* is the coordinate with the largest weight across the batch. Larger task losses contribute stronger signals, while successful executions contribute none when $\ell_i = 0$.

Failure localization therefore combines backward tracing within each execution with loss aggregation across executions. The former identifies where an individual failure can be traced back, while the latter identifies which execution coordinate repeatedly accounts for task loss. Within the selected coordinate v_t^*, the attributed evidence is further grouped into recurring failure patterns in its rules and sub-skills, identifying what within θ_{t,v_t^*} needs to be revised. These patterns are used to construct candidate modifications for the second backward stage.

3.4 LOSS-REDUCTION ESTIMATION

After failure localization selects v_t^*, the second backward stage determines *which* textual modification produces the largest decrease in task loss. Since v_t^* remains fixed throughout this stage, we suppress it from the notation below.

Candidate construction. We aggregate the attributed evidence at v_t^* into recurring failure patterns and success patterns. Failure patterns identify behaviors to repair, while success patterns constrain candidate construction so that effective behavior is preserved. Together they produce the finite candidate set

$$\mathcal{D}_t = \{\delta_t^{(m)}\}_{m=1}^{M_t},$$

where each $\delta \in \mathcal{D}_t$ modifies only θ_{t,v_t^*} and may revise its task description, rules, or sub-skills. Recurring failure patterns may induce reusable sub-skills, whereas isolated failures are restricted to local edits.

Measured loss reduction. Let $\mathcal{L}_{\mathcal{V}}(\Theta)$ denote the mean task loss on the validation subset $\mathcal{V}$. Let $\Theta_t \oplus \delta$ denote the skill obtained by applying candidate δ to θ_{t,v_t^*} while leaving the other execution coordinates unchanged. We define the loss-reduction signal as

$$s_t(\delta) = \mathcal{L}_{\mathcal{V}}(\Theta_t) - \mathcal{L}_{\mathcal{V}}(\Theta_t \oplus \delta). \tag{3}$$

A positive $s_t(\delta)$ means that the candidate reduces task loss, with larger values indicating larger measured loss reduction.

Update selection. We select the candidate with the largest measured loss reduction:

$$\delta_t^* = \arg\max_{\delta \in \mathcal{D}_t} s_t(\delta). \tag{4}$$

For a non-negative threshold ϵ, the skill is updated as

$$\Theta_{t+1} = \begin{cases} \Theta_t \oplus \delta_t^*, & s_t(\delta_t^*) > \epsilon, \\ \Theta_t, & \text{otherwise.} \end{cases} \tag{5}$$

When multiple candidates have similar positive loss-reduction signals, SKILLBACKPROP prefers candidates with fewer regressions and smaller edits. Compatible edits may be composed and evaluated once on the validation subset before being applied.

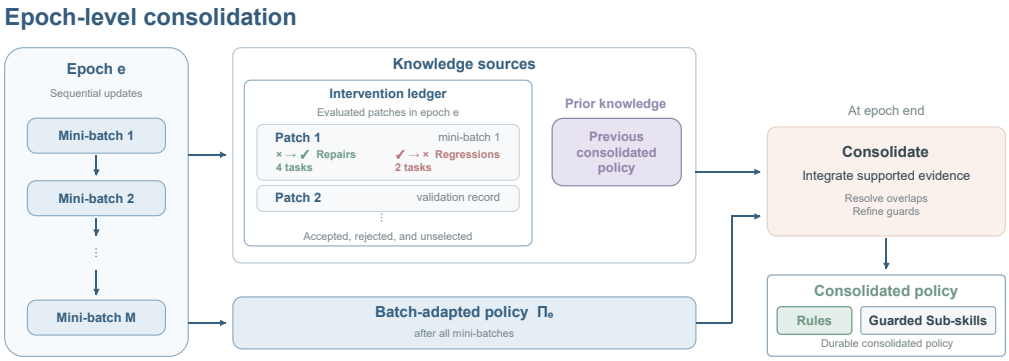

Figure 2: Epoch-level skill consolidation in SKILLBACKPROP. Across an epoch, the update record stores evaluated textual modifications and their measured loss-reduction signals. This evidence is used to consolidate compatible updates while keeping the execution topology fixed.

Knowledge scope. Accepted updates are scoped by their applicability: input-independent invariants become rules, while recurring conditional procedures become sub-skills. Both use the same acceptance criterion in Equation 3.

Epoch-level consolidation. The updates above are selected within individual mini-batches, while evidence accumulated across an epoch provides a broader basis for deciding which changes should remain in the skill. SKILLBACKPROP therefore keeps a record of evaluated edits, including the failures they target, the instances they repair, the regressions they introduce, and their measured loss-reduction signals.

At the end of each epoch, this accumulated evidence is used to merge, refine, retain, or discard rules and sub-skills. This consolidation reduces the influence of changes supported by only a single mini-batch and preserves edits that remain useful across batches. Figure 2 illustrates this process.

4 EXPERIMENTS

We evaluate SKILLBACKPROP through four questions. (1) Does textual backpropagation improve task performance across executor models and benchmarks? (2) Are the gains larger when failures must be traced across multiple execution stages? (3) Are both failure localization and loss-reduction estimation beneficial for effective optimization? (4) Does the optimized skill transfer across executor models?

4.1 EXPERIMENTAL SETUP

Benchmarks and execution structures. We evaluate on SpreadsheetBench, OfficeQA, DeepPlanning-Shopping, and SearchQA (Ma et al., 2024; Opsahl-Ong et al., 2026; Zhang et al., 2026; Dunn et al., 2017). SpreadsheetBench uses two operators for workbook exploration and solution implementation. OfficeQA uses three operators for evidence retrieval, computation and verification, and answer submission. Their separately revisable stages require both *failure localization* and *loss-reduction estimation*. DeepPlanning and SearchQA each use one operator, which fixes the update location and isolates loss-reduction estimation. For the multi-stage benchmarks, each boundary is marked by an intermediate artifact with a fixed schema. This makes it possible to distinguish an error in an upstream artifact from a later stage that processes that artifact correctly.

Skill initialization and baselines. All skill-based methods start from the same initial skill for each benchmark. For multi-stage tasks, SKILLBACKPROP compiles this skill into an Operator DAG with separately revisable operator instructions; the topology, interfaces, and permissions remain fixed throughout optimization. We compare against NO SKILL, COMPILED SKILL, GEPA, and SkillOpt. COMPILED SKILL uses the same compiled representation as SKILLBACKPROP without textual optimization. GEPA uses the same compiled DAG and initial operator instructions, while

Table 1: Task performance across three models and four benchmarks under different skill sources. Scores are percentages (↑). Values in parentheses indicate relative improvements over Compiled Skill for the same model and benchmark. Avg. is the unweighted mean across the four benchmarks.

Model	Skill Source	SpreadsheetBench	OfficeQA	DeepPlanning	SearchQA	Avg.
DeepSeek-v4-Flash	No Skill	41.43	44.05	65.48	72.38	55.84
	Compiled Skill	34.29	45.24	61.90	79.29	55.18
	GEPA	34.29 (0.0%)	59.52 (+31.6%)	73.81 (+19.2%)	83.10 (+4.8%)	62.68
	SkillOpt	55.71 (+62.5%)	48.81 (+7.9%)	**75.00** (+21.2%)	83.21 (+4.9%)	65.68
	SKILLBACKPROP	**61.43** (+79.1%)	**65.48** (+44.7%)	**75.00** (+21.2%)	**84.05** (+6.0%)	**71.49**
MiniMax-M3	No Skill	38.57	42.86	59.52	76.67	54.41
	Compiled Skill	25.71	38.10	52.38	75.00	47.80
	GEPA	30.71 (+19.4%)	48.81 (+28.1%)	65.48 (+25.0%)	81.67 (+8.9%)	56.67
	SkillOpt	61.43 (+138.9%)	48.81 (+28.1%)	66.67 (+27.3%)	81.90 (+9.2%)	64.70
	SKILLBACKPROP	**76.43** (+197.3%)	**64.29** (+68.7%)	**67.86** (+29.6%)	**82.86** (+10.5%)	**72.86**
GLM-5.3	No Skill	34.29	20.24	53.57	77.62	46.43
	Compiled Skill	35.71	29.76	40.48	76.19	45.53
	GEPA	62.86 (+76.0%)	48.81 (+64.0%)	66.67 (+64.7%)	85.95 (+12.8%)	66.07
	SkillOpt	65.00 (+82.0%)	53.57 (+80.0%)	60.71 (+50.0%)	84.64 (+11.1%)	65.98
	SKILLBACKPROP	**87.86** (+146.0%)	**70.24** (+136.0%)	**69.05** (+70.6%)	**86.31** (+13.3%)	**78.36**

SkillOpt follows its native persistent-skill optimization procedure (Agrawal et al., 2026; Yang et al., 2026).

Training and evaluation protocol. We evaluate three frozen executors: DeepSeek-v4-Flash, MiniMax-M3, and GLM-5.3. DeepSeek-v4-Flash uses the official 0731 release; MiniMax-M3 and GLM-5.3 use their official open-source releases. All model calls use the corresponding provider-default reasoning and inference settings. Each benchmark is partitioned into disjoint training, validation, and test sets. Training examples provide evidence for optimization, validation examples are used for candidate selection, and the test split is reserved for final evaluation. Within each model–benchmark setting, all methods use the same frozen executor, benchmark split, task evaluator, and initial skill where applicable. Each method otherwise follows its native optimization procedure. The training, validation, and test partitions contain 40/20/140 instances for SpreadsheetBench, 24/12/84 for OfficeQA and DeepPlanning, and 240/120/840 for SearchQA. Candidate selection uses fixed validation subsets of 8, 8, 12, and 60 instances for the four benchmarks, respectively.

Metrics. We use each benchmark's native evaluation metric: hard accuracy for SpreadsheetBench, exact match for OfficeQA and SearchQA, and Shopping case accuracy for DeepPlanning. All scores are percentages.

4.2 EVALUATION

Table 1 reports task performance across three executor models and four benchmarks. The first two observations summarize the overall comparison and then contrast multi-stage with single-stage skills to isolate the role of failure localization.

SKILLBACKPROP achieves the highest scores across the evaluated settings. Across the 12 model–benchmark settings, SKILLBACKPROP obtains the highest or tied-highest score in every setting. Averaged across all settings, it reaches 74.24, compared with 65.46 for SkillOpt and 61.81 for GEPA. Even when selecting the strongest non-SKILLBACKPROP result independently for each setting, the corresponding average is 66.95, compared with 74.24 for SKILLBACKPROP. The improvement is also consistent across executors: SKILLBACKPROP achieves average scores of 71.49, 72.86, and 78.36 on DeepSeek-v4-Flash, MiniMax-M3, and GLM-5.3, respectively, exceeding the strongest competing average for each model. This suggests that the gains are not tied to a particular executor.

The gains are larger on multi-stage tasks with multiple update coordinates. Against the strongest non-SKILLBACKPROP result in each model–benchmark setting, SKILLBACKPROP leads by 14.53 points on SpreadsheetBench and 12.70 points on OfficeQA, compared with 1.19 and 0.72

Table 2: Ablation study on MiniMax-M3 under the same training tasks and update budget. Parentheses show the relative drop from the full method. Scores are percentages (↑).

Variant	Spreadsheet Bench	OfficeQA	Deep Planning	SearchQA
SKILLBACKPROP (full)	**76.43**	**64.29**	**67.86**	**82.86**
w/o Failure Localization (first-failure)	73.57 (-3.7%)	61.90 (-3.7%)	—	—
w/o Failure Localization (random)	72.86 (-4.7%)	59.52 (-7.4%)	—	—
w/o Execution Coordinates (Monolithic Skill)	67.87 (-11.2%)	57.14 (-11.1%)	—	—
w/o Loss-reduction Estimation (accept-all)	75.71 (-0.9%)	54.76 (-14.8%)	65.48 (-3.5%)	82.02 (-1.0%)
w/o Failure Pattern Clustering (single batch revision)	59.29 (-22.4%)	60.71 (-5.5%)	64.29 (-5.3%)	81.90 (-1.2%)
w/o Epoch Consolidation	75.00 (-1.9%)	61.90 (-3.7%)	67.86 (0.0%)	82.14 (-0.9%)

Note. Scores are percentages (↑). "—" marks an inapplicable or unevaluated setting; failure localization is unnecessary for single-stage tasks because they expose only one textual update coordinate.

points on DeepPlanning and SearchQA. The largest margins therefore occur on the two multi-stage benchmarks, where optimization must first determine which execution coordinate should receive the learning signal before deciding how the instructions at that stage should be changed. In the single-stage benchmarks, this coordinate is fixed, leaving only loss-reduction estimation. The contrast is consistent with failure localization being most useful when several possible update coordinates are present; we test this contribution directly through within-benchmark ablations in Section 4.3.

4.3 ABLATION

Table 2 isolates the contributions of the execution-aligned skill structure, failure localization, loss-reduction estimation, candidate generation from recurring failure patterns, and epoch consolidation on MiniMax-M3 under the same training data, batch schedule, and edit budget.

The optimization components provide complementary benefits. Removing failure localization consistently hurts the two multi-stage benchmarks. Using first-failure attribution produces relative drops of 3.7% on both SpreadsheetBench and OfficeQA, while random attribution further increases the drop to 4.7% and 7.4%, respectively. Collapsing the execution coordinates into a monolithic skill is more damaging, reducing performance by 11.2% on SpreadsheetBench and 11.1% on OfficeQA. This removes the structured basis for failure localization, showing that effective optimization depends on identifying where the learning signal should be applied.

The remaining ablations expose complementary roles in deciding *what* to change and *which* changes to keep. Removing loss-reduction estimation has its largest effect on OfficeQA, where performance drops by 14.8%, compared with 0.9–3.5% on the other benchmarks. Removing failure pattern clustering causes the largest degradation in the study on SpreadsheetBench (22.4%), while also reducing DeepPlanning by 5.3%. Epoch consolidation has a smaller but consistent effect, with drops of up to 3.7% and no change on DeepPlanning. Overall, no single component accounts for the full improvement: different stages contribute most strongly on different benchmarks, supporting the use of the complete optimization pipeline.

We also conduct a blinded human re-audit of all 103 assignments from the two multi-stage benchmarks. The annotator receives the task outcome, execution trace, artifacts, and operator definitions, but not the assignment produced by SKILLBACKPROP, and labels the earliest stage responsible for the evaluated failure. The audit supports 96 assignments (93.2%); after excluding the 12 cases marked ambiguous, exact agreement is 96.7%.

4.4 SKILL ROBUSTNESS

A learned textual skill is more useful if the execution knowledge it captures remains effective when the executor model changes. We therefore test whether skills optimized with one model can be applied directly to another model without retraining or executor-specific adaptation.

Table 3: Cross-model generalization of optimized textual skills. Skills learned with GLM-5.3 are transferred to DeepSeek-v4-Flash and MiniMax-M3 without retraining. Transfer Gain denotes the improvement over No Skill. Scores are percentages (↑).

Transfer Source	Target Model	Benchmark	No Skill	Target-Trained Skill	Transferred Skill	Transfer Gain
GLM-5.3	DeepSeek-v4 Flash	SpreadsheetBench	41.43	61.43	84.29	+42.86
		OfficeQA	44.05	65.48	66.67	+22.62
		DeepPlanning	65.48	75.00	79.76	+14.28
		SearchQA	72.38	84.05	82.98	+10.60
		Avg.	55.84	71.49	78.42	+22.58
GLM-5.3	MiniMax-M3	SpreadsheetBench	38.57	76.43	73.57	+35.00
		OfficeQA	42.86	64.29	59.52	+16.66
		DeepPlanning	59.52	67.86	66.67	+7.15
		SearchQA	76.67	82.86	81.19	+4.52
		Avg.	54.41	72.86	70.24	+15.83

We evaluate skills optimized with GLM-5.3 on two target models, DeepSeek-v4-Flash and MiniMax-M3. For each target, we compare execution without a learned skill against execution with the transferred GLM-5.3 skill. We additionally report a target-trained skill, optimized using the target model itself, as a reference. Within each target-model block, the executor, benchmark splits, and evaluation metrics are held fixed.

Learned textual skills remain effective across executor models. Table 3 shows that skills learned with GLM-5.3 remain effective when executed by different target models. The transferred skill improves over No Skill in all eight target–benchmark combinations, with average gains of +22.58 points on DeepSeek-v4-Flash and +15.83 points on MiniMax-M3. These consistent improvements show that the benefit of skill optimization is not restricted to the model used to learn the skill.

Transfer is particularly strong on DeepSeek-v4-Flash, where the transferred skill raises the average score from 55.84 to 78.42 and exceeds the target-trained reference on three of the four benchmarks. On MiniMax-M3, the transferred skill improves the average from 54.41 to 70.24. Although it remains below the target-trained reference, it still provides consistent gains across all four benchmarks.

Overall, these results provide evidence that optimized textual skills retain useful execution behavior across model boundaries. The consistent gains over skill-free execution suggest that the learned skills capture procedures that are reusable beyond the model used during optimization, while the difference between the two target models indicates that the strength of transfer remains executor-dependent.

5 Conclusion

We introduced SkillBackprop, a two-stage textual backpropagation framework for optimizing structured skills while keeping the underlying language model frozen. Because skill instructions admit no analytic gradient, SkillBackprop recovers the two functions needed for backward learning separately: the first stage traces observed failures to parts of the skill defined by its execution structure, while the second evaluates candidate modifications through their effects on validation loss and uses loss-reduction signals to identify updates that reduce validation loss. Across four benchmarks and three executor models, SkillBackprop consistently matches or exceeds competing skill optimizers, with the largest gains on multi-stage tasks where the responsible stage must first be identified before an effective revision can be selected. Ablations isolate contributions from both backward stages, the human re-audit supports 96 of 103 failure-localization assignments, and the learned skills transfer across executor models.

AI USE STATEMENT

In this work, the authors used generative AI tools to provide feedback on the problem formulation, experimental design, analysis plan, and interpretation of results; to identify relevant literature; and to draft and refine portions of the manuscript. Generative AI was not used to generate synthetic datasets or to formulate or prove mathematical claims. The authors independently verified all AI-assisted research suggestions and literature references, reviewed and edited all AI-assisted text, and take full responsibility for the paper's content, claims, experiments, and artifacts.

REPRODUCIBILITY STATEMENT

Section 4.1 describes the benchmarks, executor models, data partitions, baselines, and evaluation metrics. Appendix A records the partition seeds, inference settings, runtime limits, and scoring rules. Appendices B and C specify the operator interfaces and optimization procedure, while Appendix I provides the initial and learned skill artifacts. The source code, including the skill compiler, training procedures, and evaluation adapters, is provided in the supplementary material, together with example commands and frozen split manifests to support reproducibility.

REFERENCES

Lakshya Agrawal et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=4oo6XTL6Oj.

Anthropic. Equipping agents for the real world with agent skills, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Matthew Dunn et al. SearchQA: A new Q&A dataset augmented with context from a search engine. *arXiv preprint arXiv:1704.05179*, 2017.

Srishti Gautam, Arjun Radhakrishna, and Sumit Gulwani. SkillAxe: Sharpening LLM-authored agent skills through evaluation-guided self-refinement. *arXiv preprint arXiv:2606.10546*, 2026. URL https://arxiv.org/abs/2606.10546.

Qingyan Guo et al. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Zhenyu He et al. ReSkill: Reconciling skill creation with policy optimization in agentic RL. *arXiv preprint arXiv:2606.01619*, 2026.

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

Omar Khattab et al. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *International Conference on Learning Representations*, 2024.

Martin Klissarov et al. MaestroMotif: Skill design from artificial intelligence feedback. In *International Conference on Learning Representations*, 2025.

George Ling, Shanshan Zhong, and Richard Huang. Agent skills: A data-driven analysis of Claude skills for extending large language model functionality. *arXiv preprint arXiv:2602.08004*, 2026. URL https://arxiv.org/abs/2602.08004.

Zhen Lu et al. SKILL0: In-context agentic reinforcement learning for skill internalization. *arXiv preprint arXiv:2604.02268*, 2026.

Xiao Luo et al. Agent Lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

Zeyao Ma et al. SpreadsheetBench: Towards challenging real world spreadsheet manipulation. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. doi: 10.52202/079017-3007.

Daniel Melcer, Qi Chen, Wen-Hao Chiang, Shweta Garg, Pranav Garg, and Christian Bock. Textual gradients are a flawed metaphor for automatic prompt optimization. *arXiv preprint arXiv:2512.13598*, 2025. URL https://arxiv.org/abs/2512.13598.

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 et al. OfficeQA Pro: An enterprise benchmark for end-to-end grounded reasoning. *arXiv preprint arXiv:2603.08655*, 2026.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7957–7968, 2023. doi: 10.18653/v1/2023.emnlp-main.494. URL https://aclanthology.org/2023.emnlp-main.494/.

Yu Shang et al. AgentSquare: Automatic LLM agent search in modular design space. In *International Conference on Learning Representations*, 2025.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Shicheng Tu et al. Dynamic dual-granularity skill bank for agentic reinforcement learning. *arXiv preprint arXiv:2603.28716*, 2026.

Yash Vishe et al. Skill-R1: Agent skill evolution via reinforcement learning. *arXiv preprint arXiv:2605.09359*, 2026.

Guanzhi Wang et al. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Hanyu Wang, Yifan Lan, Bochuan Cao, Lu Lin, and Jinghui Chen. SkillGrad: Optimizing agent skills like gradient descent. *arXiv preprint arXiv:2605.27760*, 2026a. URL https://arxiv.org/abs/2605.27760.

J. Wang et al. Reinforcement learning for self-improving agent with skill library. In *Annual Meeting of the Association for Computational Linguistics*, pp. 1529–1550, 2026b.

Xinyuan Wang et al. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, 2024b.

Zora Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *International Conference on Machine Learning*, 2025a.

Zora Zhiruo Wang, Ansh Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Conference on Language Modeling*, 2025b.

Peng Xia et al. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *arXiv preprint arXiv:2602.12430*, 2026. URL https://arxiv.org/abs/2602.12430.

Chengrun Yang et al. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Yifan Yang et al. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Mert Yuksekgonul et al. TextGrad: Automatic differentiation via text. *Nature*, 639:664–672, 2025. doi: 10.1038/s41586-025-08661-4.

Jiayi Zhang et al. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025.

Yinger Zhang, Shutong Jiang, Renhao Li, Jianhong Tu, Yang Su, Lianghao Deng, Xudong Guo, ChenXu Lv, and Junyang Lin. DeepPlanning: Benchmarking long-horizon agentic planning with verifiable constraints. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7377–7407, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.335. URL https://aclanthology.org/2026.acl-long.335/.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Andy Zhou et al. Language agent tree search unifies reasoning, acting, and planning in language models. In *International Conference on Machine Learning*, 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 *International Conference on Learning Representations*, 2023.

Table A1: Training and selection configuration. "Eval. size" is the number of validation tasks evaluated for a candidate revision. The full validation set remains separately archived.

Benchmark	Train	Validation	Test	Seed	Epochs	Batch	Eval. size	Edit budget
SpreadsheetBench	40	20	140	42	4	8	8	4
OfficeQA	24	12	84	42	4	8	8	5
DeepPlanning–Shopping	24	12	84	42	2	8	12	5
SearchQA	240	120	840	43	4	40	60	4

Table A2: Execution structures used for the four benchmarks. Arrows separate Operator responsibilities in multi-stage skills. DeepPlanning uses the Shopping setting.

Benchmark	Stage Responsibilities	Operators
SpreadsheetBench	Explore workbook → Implement/execute solution	2
OfficeQA	Retrieve evidence → Compute/verify → Submit answer	3
DeepPlanning	Fulfill constrained shopping request	1
SearchQA	Produce evidence-grounded answer	1

A Implementation and Reproducibility Details

Deterministic data partitions. All methods use the same benchmark-specific task IDs. SpreadsheetBench, OfficeQA, and DeepPlanning use seed 42; SearchQA uses the published controlled partition identified by seed 43. Split manifests store the ordered task IDs. The held-out test split is reserved for final evaluation. Table A1 records the configuration used by the archived runs.

Models and inference settings. We evaluate `deepseek-v4-flash`, `minimax-m3`, and `glm-5.3`. DeepSeek uses the official 0731 release, while MiniMax-M3 and GLM-5.3 use their corresponding official open-source releases. Executor, Analyzer, and Optimizer calls use provider-default reasoning and sampling settings. The executor remains frozen throughout optimization, and within each model–benchmark setting, all methods use the same model family for execution and Analyzer/Optimizer calls.

Runtime limits. The archived SkillBackprop configuration permits up to 131,072 completion tokens. The common GEPA/SkillOpt environment uses the same 131,072-token ceiling for rollouts. Spreadsheet execution uses a 600-second task timeout, while the other benchmark adapters use 120 seconds; provider-level request timeouts range from 300 to 1,800 seconds depending on the endpoint. Exact model identifiers and request/response payloads are retained in the archived run records.

Failure handling and scoring. An execution or tool failure receives the benchmark failure score and its trace remains available to the Analyzer. A failed upstream Operator prevents dependent Operators from being credited for the same causal failure. Network or schema errors are retried only within the configured retry bound; an exhausted request is recorded as a failure. Candidate edits are evaluated by executing the skill on the fixed validation slice. Let q_0 and q_δ be the baseline and intervened validation scores. The implementation records $\mathrm{CDG}(\delta) = q_\delta - q_0$ and uses threshold $\epsilon = 0$. A candidate is promoted only when $\mathrm{CDG}(\delta) > 0$. Candidates are ordered by descending CDG, then fewer validation regressions, lower edit cost, and candidate ID. The final consolidated policy is used for benchmark evaluation. Benchmark scores are hard accuracy for SpreadsheetBench, exact match for OfficeQA and SearchQA, and Shopping case accuracy for DeepPlanning.

B Benchmark-Specific Operator Definitions

The compiler fixes topology, typed inputs and outputs, and tool permissions before training. Optimization can change only the textual Task Description, Rules, and Sub-skills inside an Operator. It cannot add an edge, change an artifact type, or expand a tool allow-list. Table A2 summarizes the four graphs. Tables A3 and A4 give the responsibilities, completion conditions, interfaces, and tool permissions that make multi-stage failure localization nontrivial.

Table A3: Responsibilities and completion conditions for the multi-stage benchmark Operators.

Benchmark / Operator	Responsibility	Completion condition
SpreadsheetBench `explore_workbook`	Recover the smallest sufficient structural and formatting evidence.	The four-section context records every sheet preview plus any discriminative supplemental range; it contains evidence, not a proposed algorithm.
SpreadsheetBench `implement_and_execute_solution`	Implement a reusable transformation and execute it on the actual workbook.	A script accepting exactly `--input` and `--output` runs successfully, preserves unrelated content, and writes the graded result.
OfficeQA `retrieve_evidence`	Ground every operand with its entity, period, unit, row/column semantics, and source span.	All recoverable operands are grounded; unresolved targets and attempted searches are explicitly recorded. No final arithmetic is performed.
OfficeQA `compute_and_verify`	Compute from grounded operands and verify direction, sign, scale, precision, and plausibility.	Python prints one schema-valid object containing operands, operation, calculation, unrounded result, answer, and limitations.
OfficeQA `format_and_submit`	Map the verified result to the exact answer representation.	Submit only the requested value, phrase, date, or list, with the required sign, unit scale, percent/decimal convention, and trailing zeroes.

Table A4: Typed interfaces and tool permissions for the same multi-stage Operators. The named output artifact is the only state passed to the next Operator.

Benchmark / Operator	Input → output	Allowed tools
SpreadsheetBench `explore_workbook`	Runtime task → `workbook_context.md`.	Workbook overview, targeted cell inspection, workbook search, and file write.
SpreadsheetBench `implement_and_execute_solution`	Task and context → `solution.py` and output workbook.	File read/write and sandboxed solution execution.
OfficeQA `retrieve_evidence`	Runtime question → `officeqa_evidence.json`.	Offline document search, bounded span read, and file write.
OfficeQA `compute_and_verify`	Evidence JSON → `officeqa_solution.json`.	File read/write and sandboxed Python.
OfficeQA `format_and_submit`	Solution JSON → submitted answer.	File read and answer submission only.

DeepPlanning and SearchQA intentionally expose one execution coordinate. A DeepPlanning trajectory must identify one verified product per requested slot, apply valid owned coupons, construct the cart, and inspect it before finishing; the Operator can use the benchmark's official product, user-information, transport, cart, and coupon tools. SearchQA receives the question and noisy retrieval context directly and has no tools; its completion condition is one concise tagged answer. Splitting either task across multiple operators would introduce an arbitrary credit boundary without an independently checkable intermediate artifact, so the single-Operator form isolates loss-reduction estimation.

C Optimization Algorithm and Prompt Contracts

Implementation fidelity. Algorithms 1–2 specify the full method with validation-based candidate selection; ablation-only branches that replace semantic attribution, clustering, or validation-based evaluation are omitted. Every batch starts from the latest accepted tree, not the epoch-start tree. In-

Algorithm 1 Execution-aligned textual backpropagation: training and candidate construction

Require: Initial Operator DAG G_{Θ_0}; training set $\mathcal{D}_{\mathrm{tr}}$; fixed validation subset $\mathcal{V} \subseteq \mathcal{D}_{\mathrm{val}}$; epochs E; batch size K; edit budget B; descent threshold $\epsilon = 0$.
Ensure: Consolidated tree $G_{\Theta^{\mathrm{cons}}}$ and its complete training and evaluation record.

$G_\Theta \leftarrow G_{\Theta_0}$; $G_{\bar{\Theta}} \leftarrow G_{\Theta_0}$; initialize per-Operator trial/CDG memory M, an empty effect-card ledger $\mathcal{L}$, and an empty cached validation reference $\mathcal{Q}_0$.
for $e = 1, \ldots, E$ **do**
  Deterministically shuffle all of $\mathcal{D}_{\mathrm{tr}}$ with seed $s+1000e$ and partition it without replacement into batches of size K.
  for batch $\mathcal{B}$ in the scheduled order **do**
    for all $x_i \in \mathcal{B}$ **in parallel do**
      Execute the DAG G_Θ; retain score s_i, trace τ_i, artifacts, tool calls, errors, and activated policy blocks.
      Analyze (x_i, τ_i) immediately; assign exactly one responsible Operator a_i at the earliest scorer-relevant divergence and set $w_i \leftarrow \max(0, 1 - s_i)$.
    end for
    Set mode to REPAIR if any execution failed, IMPROVE if any completed execution failed evaluation, and SKIP otherwise; skip the remaining batch steps in SKIP mode.
    Compute responsibility mass $R(v) = \sum_i w_i \mathbf{1}[a_i = v]$ and graph error contribution $EC(v) = f_{\mathrm{contract}}(v) + \alpha f_{\mathrm{lineage}}(v) + \beta f_{\mathrm{downstream}}(v)$.
    Let $U(v) = \sqrt{\log(T+2)/(n_v+1)}$. Choose v^* among the maximizers of $R(v)$; break ties by normalized $EC(v)$ in REPAIR, by $U(v)$ during scheduled exploration, and otherwise by mean observed CDG (or $U(v)$ if untried).
    Cluster findings attributed to v^* into active failure patterns and reusable success patterns, each with support/contradiction IDs, policy scope, and related Rule/Sub-skill IDs; call the resulting targets $\mathcal{Z}$.
    $\mathcal{P} \leftarrow \emptyset$.
    for all learning target $z \in \mathcal{Z}$ **do**
      Propose at most one typed edit δ_z to v^* (add/replace/remove Rule or add/replace Sub-skill), preserving DAG topology, I/O contracts, and tools.
      Materialize $G_{\Theta \oplus \delta_z}$; discard no-change, duplicate, over-budget, invalid-schema, and previously rejected same-base candidates; otherwise add δ_z to $\mathcal{P}$.
    end for
    if $\mathcal{P} = \emptyset$ **then**
      Record no change, checkpoint the batch, and continue.
    end if
    $(G_\Theta, \mathcal{Q}_0, \mathcal{L}, M) \leftarrow$ VALIDATEANDPROMOTE$(G_\Theta, \mathcal{P}, \mathcal{V}, \mathcal{Q}_0, \mathcal{L}, M, B, \epsilon)$; see Algorithm 2.
  end for
  Conservatively update shadow $G_{\bar{\Theta}}$ from the current fast tree, prior shadow, and cumulative effect cards $\mathcal{L}$; expose only aggregate effects, preserve immutable contracts/IDs, and expose no validation examples or answers.
end for
$G_{\Theta^{\mathrm{cons}}} \leftarrow G_{\bar{\Theta}}$. Return the consolidated tree and evaluate it on the sealed test split.

dividual candidates are generated from the same base tree and share one cached validation reference. A positive bundle is evaluated separately on the validation tasks and competes with its component patches, which is why an adverse interaction cannot erase the best independently measured edit. The checkpoint stores the current tree, cached validation reports, batch history, accepted batches, and both optimizer memories, so resume continues at the next unfinished batch without repeating a completed update decision.

Failure localization and tie-breaking. The item Analyzer must first reconstruct the task requirement, observed run, and delivered result, then select exactly one Operator. It assigns the earliest scorer-relevant divergence to the Operator whose explicit contract could have prevented or repaired it. Each assignment carries an attribution weight; successful items contribute zero loss mass. Batch selection maximizes the aggregated responsibility mass. During repair, equal mass is resolved by normalized graph error contribution. During the first two improvement opportunities and every fourth subsequent improvement opportunity, it is resolved by uncertainty $\sqrt{\log(T+2)/(n_v+1)}$; otherwise, empirical mean CDG is used (or uncertainty for an untried Operator). This rule changes only how a tied coordinate is resolved, not the descent threshold.

Prompt templates and structured outputs. Every model-mediated training step is constrained by the operational contract in Table A5. Each request requires a JSON object validated against a

Algorithm 2 Candidate evaluation, bundle validation, and sequential promotion

function VALIDATEANDPROMOTE($G_\Theta, \mathcal{P}, \mathcal{V}, \mathcal{Q}_0, \mathcal{L}, M, B, \epsilon$)
  if $\mathcal{Q}_0 = \varnothing$ **then**
    Execute G_Θ on every $x \in \mathcal{V}$ once; cache the active reports and their mean score q_0 in $\mathcal{Q}_0$.
  end if
  for all $\delta \in \mathcal{P}$ **do**
    Execute the DAG $G_{\Theta \oplus \delta}$ on the same $\mathcal{V}$; record q_δ and $\Delta_\delta = q_\delta - q_0$.
    Align baseline and candidate reports by task ID; count regressions, improvements, remaining failures, and successes; retain edit cost, policy hash, and per-task transitions in a provisional effect record.
  end for
  Sort candidates with $\Delta_\delta > \epsilon$ by $(-q_\delta$, regressions, edit cost, candidate ID).
  Traverse this order and greedily retain every candidate whose cost keeps the cumulative edit cost at most B (later cheaper candidates may still fit).
  if at least two retained positive candidates exist **then**
    Merge them in ranked order from the shared active base, skipping incompatible edits; reject a no-op, duplicate, or previously rejected bundle.
    if a distinct merged bundle remains **then**
      Evaluate it once on $\mathcal{V}$ and add its measured result to $\mathcal{P}$; it receives no preference over individual candidates.
    end if
  end if
  Rank every measured individual and bundle candidate by $(-q_\delta$, regressions, edit cost, candidate ID) and let δ^* be the first entry.
  if δ^* edits a Sub-skill **then**
    Re-execute its supporting batch items to audit activation and support; this diagnostic is recorded but does not replace validation-based loss-reduction estimation.
  end if
  Append effect cards for every measured candidate to $\mathcal{L}$, preserving rejected and regressive evidence and marking only a promoted winner accepted.
  if $\mathcal{V} \neq \emptyset$ **and** $\Delta_{\delta^*} > \epsilon$ **then**
    Promote $G_\Theta \leftarrow G_{\Theta \oplus \delta^*}$ and set $\mathcal{Q}_0$ to δ^*'s reports and score; this becomes the base of the next batch.
  else
    Keep G_Θ and $\mathcal{Q}_0$ unchanged.
  end if
  Update Operator trials, mean/best CDG, success rate, uncertainty, the problem/update ledger, and transition feedback in M.
  Atomically checkpoint the current tree, validation reports, batch history, accepted batches, optimizer memories, and completed-batch index.
  return $(G_\Theta, \mathcal{Q}_0, \mathcal{L}, M)$.
end function

closed schema; prose outside the schema is rejected and retried within the request bound. The table expands the information that is otherwise encoded in the prompt templates, so interpretation does not depend on a local source path.

Candidate evaluation and promotion are implemented in code rather than delegated to a language model for judgment. The resulting record contains baseline and candidate scores, CDG, regression count, edit cost, validation task count, per-task transitions, decision reason, and the hashes of the candidate policy. This separation is important: prompt plausibility proposes a direction, while its measured effect on validation loss determines whether that direction is written back.

D BASELINE CONFIGURATIONS AND COMPARISON CONTROLS

Definitions. NO SKILL runs the frozen executor and benchmark tools without a persistent task policy. COMPILED SKILL uses the same initial skill and, for multi-stage tasks, the same compiler-produced Operator graph as SKILLBACKPROP, but performs no textual update. GEPA starts from the same initial text and evaluates proposed text variants through the shared benchmark adapter. Its archived MiniMax configuration requests 100 metric calls for SpreadsheetBench, OfficeQA, and DeepPlanning and 300 for SearchQA. SkillOpt uses its native persistent-skill loop, including mini-batch analysis, merge, selection gate, slow update, and meta-skill stages. It keeps one monolithic skill document rather than tracing failures across Operator coordinates.

Table A5: Operational prompt contracts used by the optimizer.

Stage	Required reasoning and exclusions	Required structured fields
Per-item attribution	Reconstruct the requirement, observed run, and delivered result; identify the earliest scorer-relevant divergence; select exactly one Operator whose contract could have prevented or repaired it; cite only trace-visible evidence and do not propose an edit.	`selected_operator_id`, attribution weight, task requirement, observed run, delivered result, decisive evidence, confidence, and reusable finding.
Failure pattern clustering	Group attributed failures only when they share a causal behavior and applicable policy scope; separate task-specific facts from reusable knowledge; audit the current target policy; preserve successful methods and list findings that cannot be safely assigned.	Problems and successful methods with pattern key, causal behavior, knowledge scope, supporting task IDs, policy audit, and unassigned findings.
Candidate revision	Revise only the selected Operator; ground each edit in identified problem IDs; preserve I/O contracts and tool permissions; prefer the smallest reusable behavioral change; do not change graph topology or inject task answers.	Target problem IDs, reasoning, and typed edits: add/replace/remove Rule or add/replace Sub-skill, each with target ID, text, rationale, and cost.
Candidate ranking (plausibility ablation only)	Replace validation-based evaluation with a model estimate of causal fit, expected coverage, regression risk, and redundancy. This contract is used only by the corresponding ablation; the full method ranks candidates deterministically from measured validation outcomes.	Reasoning, selected proposal indices, and conservative scores under the edit budget.
Epoch consolidation	Reconcile the initial, fast, and prior consolidated policy against accepted/rejected effect cards; retain supported behavior, merge compatible rules, discard harmful or redundant edits, and exclude validation questions, answers, task IDs, and traces from the resulting policy.	All retained Rules and Sub-skills plus a change list naming keep, merge, add, remove, and priority decisions with evidence.

Table A6: Training schedules and optimizer-level update opportunities in the archived MiniMax-M3 runs. SkillOpt and SKILLBACKPROP use identical task-exposure and batch-level update schedules; GEPA follows its native metric-budgeted proposal procedure.

Benchmark	Train	Epochs	Batch	Exposures	GEPA	SkillOpt	SKILLBACKPROP
SpreadsheetBench	40	4	8	160	25	20	20
OfficeQA	24	4	8	96	25	12	12
DeepPlanning	24	2	8	48	19	6	6
SearchQA	240	4	40	960	17	24	24

Shared controls and non-shared mechanisms. All methods receive identical train/validation/test task IDs, initial benchmark knowledge, executor family, tools, timeouts, and native evaluator. Test tasks are reserved for evaluation. The SKILLBACKPROP settings follow Table A1; native baseline controls are reconstructed from their archived manifests. Each baseline retains its native internal optimization loop while sharing the same data-access envelope and evaluation interface. Method-native proposal and memory mechanisms are retained: GEPA keeps its population/reflection procedure, SkillOpt keeps slow update and meta-skill logic, and SKILLBACKPROP keeps explicit attribution, localized edits, validation score gains (CDG), and consolidation.

E TRAINING SCHEDULE AND UPDATE OPPORTUNITIES

We report the training schedule and optimizer-level update opportunities for the archived MiniMax-M3 runs. These quantities characterize how often each method observes training evidence and has an opportunity to revise the textual policy.

SkillOpt and SKILLBACKPROP follow the same epoch-based training protocol: they use identical training task IDs, epoch schedules, batch sizes, and therefore the same number of training-task exposures and batch-level update opportunities. GEPA follows its population-based proposal and reflection procedure, whose optimization budget is set by metric calls; we retain this procedure under the archived metric-call budget and report the resulting proposal rounds.

Table A6 summarizes these schedules. For SkillOpt and SKILLBACKPROP, training-task exposures equal the training-set size multiplied by the number of epochs, and each training batch defines one optimizer-level opportunity to update the skill. For GEPA, a round denotes one proposal iteration under its optimization procedure.

Table A7: Agreement between the archived localization and the blinded human re-audit. "Ambiguous" means that the evidence supports multiple assignments; "Supported" means the archived Operator lies in the defensible set.

Split	N	Supported	Agreement	Ambiguous
All trajectories	103	96	93.2%	12
SpreadsheetBench	74	73	98.6%	1
OfficeQA	29	23	79.3%	11
DeepSeek-v4-Flash	41	39	95.1%	4
GLM-5.3	25	22	88.0%	5
MiniMax-M3	37	35	94.6%	3
Non-ambiguous	91	88	96.7%	0

F Human Re-Audit of Failure Localization

Population and blinding. The audit covers the full finite population of 103 archived attributed failures for which multi-stage credit assignment is meaningful: 74 SpreadsheetBench and 29 OfficeQA trajectories. SearchQA and DeepPlanning are excluded because their single-Operator structures make the label degenerate. The human annotator receives the task outcome, complete, untruncated trace and artifacts, errors, and a snapshot of the executed Operator contracts, but not SkillBackprop's archived attribution. One annotator applies the fixed protocol, so we report agreement with the blinded audit.

Decision rule. The annotation target is the earliest scorer-relevant divergence. Execution failure is assigned to the stage that exhausted its budget or blocked further execution. A wrong or incomplete upstream artifact is assigned to its producer when downstream stages consume it faithfully. Unsupported invention, substitution, or arithmetic from otherwise correct evidence is assigned to `compute_and_verify`; a representation-only mismatch is assigned to `format_and_submit`. For spreadsheets, missing or wrong structural facts in `workbook_context.md` belong to exploration, while incorrect writes or execution from sufficient context belong to implementation. The annotator always records a best label and sets `ambiguous=1` when two causes remain comparably defensible or when source coverage prevents a causal distinction.

In ambiguous cases, an archived assignment is counted as supported when it falls within the annotator's defensible set; "supported" and "ambiguous" are therefore not mutually exclusive. The results are summarized in Table A7.

The frozen audit record has three logically separate parts: (i) the decision protocol stated above; (ii) one blinded source packet per trajectory containing the task outcome, complete trace, intermediate artifacts, errors, and executed Operator snapshot; and (iii) one result row per packet containing benchmark, model, task identifier, archived label, human primary label, defensible label set, ambiguity flag, support indicator, rationale, and cited evidence. The population counts and all derived agreement statistics are printed in Table A7; the representative supported, ambiguous, and rejected cases below expose how the same protocol is applied to concrete records.

G Validation Performance During Optimization

Figure 3 shows validation scores during optimization across three executor models and three benchmarks. At each update step, candidate edits are compared with the previously retained policy. The dashed curve reports the reference score used for this comparison, while the solid curve reports the highest score among the reference policy and the evaluated candidates.

SpreadsheetBench shows early improvements followed by plateaus, whereas DeepPlanning and SearchQA exhibit more variable trajectories. In runs that re-evaluate the retained policy, its measured score can change between steps. The curves therefore report within-step comparisons rather than cumulative historical maxima.

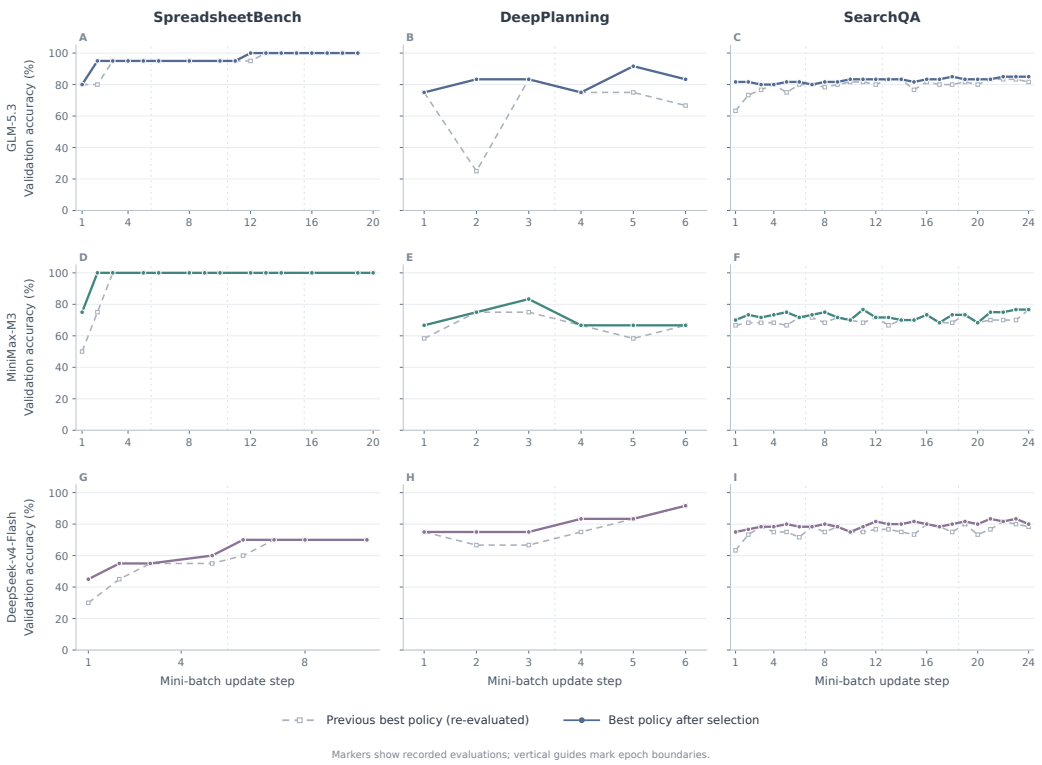

Figure 3: Validation performance during optimization across three executor models and three benchmarks. Dashed curves show the reference scores of the previously retained policies; solid curves show the highest scores among those policies and the candidates evaluated at each step. Scores are within-step maxima, not cumulative historical maxima. Lines connect recorded observations without imputing missing evaluations. Vertical dotted lines indicate epoch boundaries.

H Qualitative Optimization Traces

An accepted SpreadsheetBench correction. In MiniMax-M3 epoch 1, batch 1, failures on tasks `47766` and `50193` were localized to `implement_and_execute_solution`. Both scripts wrote formula strings into graded cells, but the workbook evaluator read no cached values. The clustered pattern proposed sharpening Rule 010: computed graded outputs should normally be written as Python-computed literals, and a formula string should be used only when the task explicitly requires a live formula and recalculation is available. The revised skill was evaluated on eight validation tasks. It improved two tasks (`33722` and `50526`), regressed none, and raised accuracy from 0.50 to 0.75. The edit was therefore accepted with CDG $+0.25$.

An ambiguous OfficeQA attribution. Task `UID0150` asks for a 1976 one-step-ahead Treasury-bond forecast and its error in DEM. Retrieval grounded the 1972–1975 bond prices and exchange rates but not the March 1976 USD/DEM rate. Computation correctly obtained the USD forecast (74.941) and difference (-7.184), then explicitly used the last available 1975 rate as a fallback. The reference pair corresponds to an approximately 2.56 rate for 1976. The human audit assigns the earliest divergence to `retrieve_evidence`, but marks the case ambiguous because the missing rate may reflect the supplied source package rather than recoverable retrieval behavior. This is the main OfficeQA boundary: incomplete evidence and incomplete source coverage are observationally entangled.

A candidate rejected after validation. In SpreadsheetBench epoch 3, batch 4, task `227-40` suggested a plausible guard against executing a stale `solution.py` after the upstream context file cannot be read. The candidate added a rule requiring workspace-relative recovery, script regeneration, and a check that the graded range is written. Despite its local rationale, the candidate reduced validation accuracy from 1.00 to 0.50: four tasks (`33722`, `48982`, `297-42`, and `477-45`) re-

gressed and none improved. With CDG -0.50, the rule was rejected and the active policy was left unchanged. This trace illustrates why plausible textual advice is not treated as a descent direction without revision evidence.

I SELF-CONTAINED SKILL ARTIFACTS

This section prints the policy artifacts needed to reconstruct the starting point and inspect the learned endpoint without access to an author machine or an external artifact archive. For each benchmark, we give the complete source skill, its compiled initial representation, and the final consolidated policy from GLM-5.3, the executor with the highest four-benchmark mean in Table 1. Using one model for all four artifacts avoids a post-hoc per-benchmark mixture of executor families. The multi-stage benchmarks have compiler-emitted JSON trees. DeepPlanning and SearchQA are intentionally single-stage: their compiled topology is one Operator whose policy body is the source skill. For those two benchmarks the small JSON object below makes that topology explicit, while the adjacent listing supplies the complete policy body. These single-node JSON objects are lossless appendix renderings of the runner contract, not additional archived compiler outputs.

I.1 SPREADSHEETBENCH

The selected endpoint is the GLM-5.3 run scoring 87.86% task-level hard accuracy. Its evolved graph is version 6; version 1 is the compiled initial graph.

Complete initial source skill.

````
# Spreadsheet Manipulation Skill (xlsx)

Compile this Skill into two stages. Stage 1 writes `workbook_context.md`;
Stage 2 writes and submits `solution.py`.

Use Python for spreadsheet manipulation. Prefer `openpyxl` when formulas,
formatting, sheets, or named ranges must be preserved. Use `pandas` only for
bulk tabular transformation, then write results back with `openpyxl` because
`pandas.to_excel()` can destroy workbook structure.

## Stage 1: Explore

Call `inspect_workbook` once to obtain the compact global preview. Write and
submit one Markdown file named `workbook_context.md` containing exactly four
sections: `User task instruction`, `Instruction type`, `Answer position`, and
`Input xlsx preview`. This artifact passes observed evidence to Stage 2; do not
add a proposed algorithm or a general analysis section.

In `Input xlsx preview`, represent every worksheet in the same format:

- worksheet name, dimensions, `max_row`, and `max_col`;
- rows 1--5 crossed with columns 1--20;
- the last 5 populated rows and last 5 populated columns returned by the
  global inspection;
- cells in `coordinate=value` form, including `coordinate=` for blanks and
  exact formula text for formulas;
- cell text longer than 40 characters truncated with `...`;
- counts of omitted populated rows or columns when nonzero; and
- useful formatting and workbook metadata already returned by inspection,
  including relevant number formats, fonts, fills, borders, alignment, merged
  ranges, hidden dimensions, tables, filters, validations, and conditional
  formatting. Omit metadata categories that were not returned.

If this compact evidence does not reveal a task-required source table, target
range, example, formula pattern, lookup key, or style, use
`inspect_workbook_cells` or `search_workbook` for the smallest discriminative
region that supplies the missing evidence. Append supplemental cells under the
same worksheet preview, labelled with sheet and range and kept in the same
`coordinate=value` format. Prefer a region with new structural information;
do not repeatedly sample equivalent rows or widen a request merely to collect
adjacent cells. Stop as soon as the evidence is sufficient to write the stage
artifact.

## Stage 2: Implement and execute

Read `workbook_context.md` and create a reusable `solution.py` that accepts
exactly:

```text
--input <input.xlsx> --output <output.xlsx>
```

Load only the input argument, perform the requested manipulation, preserve
````

```
unrelated workbook content, and save to the output argument. Do not hardcode
development paths, row counts, column letters, or preview values; inspect the
actual workbook in the script. Write graded computed results as literal values
unless the task explicitly requires formulas.

Run the script with the spreadsheet execution tool. Repair concrete execution
errors if needed. A successful execution submits `solution.py` and completes
the workflow.
```

Complete compiled initial tree (v0001).


```
{
  "format_version": "2.0",
  "tree_id": "spreadsheetbench",
  "name": "Spreadsheet Manipulation Skill",
  "description": "Compiles a two-stage workflow for spreadsheet manipulation: first explore the input
     workbook and write a context markdown file, then implement and execute a reusable solution script.",
  "root_operator_id": "implement_and_execute_solution",
  "operators": [
    {
      "id": "explore_workbook",
      "task": "Explore the input .xlsx workbook to gather structural evidence needed for later implementation
      . Use the inspect_workbook tool once to obtain a compact global preview of all sheets. If the preview
      does not reveal task-required source tables, target ranges, examples, formula patterns, lookup keys, or
       styles, use inspect_workbook_cells or search_workbook to retrieve the smallest discriminative region
      that supplies the missing evidence. Stop as soon as the evidence is sufficient. Write a Markdown file
      named workbook_context.md containing exactly four sections: 'User task instruction', 'Instruction type
      ', 'Answer position', and 'Input xlsx preview'. In the preview, represent every worksheet with its name
      , dimensions, max_row, max_col, rows 1-5 crossed with columns 1-20, the last 5 populated rows and last
      5 populated columns, cells in coordinate=value form (including coordinate= for blanks and exact formula
       text for formulas), truncate cell text longer than 40 characters with '...', include counts of omitted
       populated rows or columns when nonzero, and include useful formatting and workbook metadata returned
      by inspection (number formats, fonts, fills, borders, alignment, merged ranges, hidden dimensions,
      tables, filters, validations, conditional formatting). Omit metadata categories not returned. If
      supplemental cells are needed, append them under the same worksheet preview, labelled with sheet and
      range, in the same coordinate=value format. Do not add a proposed algorithm or general analysis section
      . The output file must be created in the workspace and its path submitted as the output.",
      "inputs": [
        {
          "name": "task",
          "source": "runtime_task",
          "source_operator_id": null,
          "source_output": null
        }
      ],
      "outputs": [
        {
          "name": "workbook_context",
          "kind": "file",
          "description": "Markdown file named workbook_context.md containing the four required sections with
      the compact global preview and any supplemental evidence needed for Stage 2. It must be complete enough
       for the downstream implementer to understand the workbook structure and the task requirements without
      re-inspecting the workbook."
        }
      ],
      "allowed_tools": [
        "inspect_workbook",
        "inspect_workbook_cells",
        "search_workbook",
        "write_file"
      ],
      "operator_version": 1,
      "rules": [
        {
          "id": "rule_001",
          "text": "Call inspect_workbook once to obtain the compact global preview."
        },
        {
          "id": "rule_002",
          "text": "Write and submit one Markdown file named workbook_context.md containing exactly four
      sections: User task instruction, Instruction type, Answer position, and Input xlsx preview."
        },
        {
          "id": "rule_003",
          "text": "In Input xlsx preview, represent every worksheet with name, dimensions, max_row, max_col,
      rows 1-5 crossed with columns 1-20, the last 5 populated rows and last 5 populated columns, cells in
      coordinate=value form including coordinate= for blanks and exact formula text for formulas, truncate
      cell text longer than 40 characters with '...', include counts of omitted populated rows or columns
      when nonzero, and include useful formatting and workbook metadata already returned by inspection. Omit
      metadata categories that were not returned."
        },
        {
          "id": "rule_004",
          "text": "If compact evidence does not reveal a task-required source table, target range, example,
      formula pattern, lookup key, or style, use inspect_workbook_cells or search_workbook for the smallest
      discriminative region that supplies the missing evidence. Append supplemental cells under the same
      worksheet preview, labelled with sheet and range and kept in the same coordinate=value format. Prefer a
       region with new structural information; do not repeatedly sample equivalent rows or widen a request
      merely to collect adjacent cells. Stop as soon as the evidence is sufficient."
```

```
        },
        {
          "id": "rule_005",
          "text": "Do not add a proposed algorithm or a general analysis section to workbook_context.md."
        }
      ],
      "sub_skills": []
    },
    {
      "id": "implement_and_execute_solution",
      "task": "Read the workbook_context.md file produced by the explore_workbook operator. Create a reusable
       solution.py that accepts exactly --input <input.xlsx> --output <output.xlsx>. Load only the input
      argument, perform the requested manipulation, preserve unrelated workbook content, and save to the
      output argument. Do not hardcode development paths, row counts, column letters, or preview values;
      inspect the actual workbook in the script. Write graded computed results as literal values unless the
      task explicitly requires formulas. Run the script with the run_spreadsheet_solution tool. If execution
      fails, repair concrete execution errors and rerun. When execution succeeds and creates the output,
      submit solution.py immediately; do not call finish_task afterward.",
      "inputs": [
        {
          "name": "task",
          "source": "runtime_task",
          "source_operator_id": null,
          "source_output": null
        },
        {
          "name": "workbook_context",
          "source": "operator_output",
          "source_operator_id": "explore_workbook",
          "source_output": "workbook_context"
        }
      ],
      "outputs": [
        {
          "name": "solution_script",
          "kind": "file",
          "description": "The solution.py file that implements the requested spreadsheet manipulation. It
      must be a reusable script accepting --input and --output arguments, preserving unrelated workbook
      content, and producing the output workbook when executed."
        }
      ],
      "allowed_tools": [
        "read_file",
        "write_file",
        "run_spreadsheet_solution"
      ],
      "operator_version": 1,
      "rules": [
        {
          "id": "rule_006",
          "text": "Use Python for spreadsheet manipulation. Prefer openpyxl when formulas, formatting, sheets
      , or named ranges must be preserved. Use pandas only for bulk tabular transformation, then write
      results back with openpyxl because pandas.to_excel() can destroy workbook structure."
        },
        {
          "id": "rule_007",
          "text": "Create a reusable solution.py that accepts exactly --input <input.xlsx> --output <output.
      xlsx>."
        },
        {
          "id": "rule_008",
          "text": "Load only the input argument, perform the requested manipulation, preserve unrelated
      workbook content, and save to the output argument."
        },
        {
          "id": "rule_009",
          "text": "Do not hardcode development paths, row counts, column letters, or preview values; inspect
      the actual workbook in the script."
        },
        {
          "id": "rule_010",
          "text": "Write graded computed results as literal values unless the task explicitly requires
      formulas."
        }
      ],
      "sub_skills": []
    }
  ],
  "result_envelope_format_version": "1.0",
  "source_skill_hash": "8a7ba476b2811861c2b69e9471d9c5a773fcdeaf3bea74d0d9ed44a736a3bade",
  "compiler_model_profile": "deepseek_main"
}
```


Final consolidated tree from the GLM-5.3 run (87.86%, v0006).


```
{
  "format_version": "2.0",
  "tree_id": "spreadsheetbench",
  "name": "Spreadsheet Manipulation Skill",
```

```
"description": "Compiles a two-stage workflow for spreadsheet manipulation: first explore the input
    workbook and write a context markdown file, then implement and execute a reusable solution script.",
"root_operator_id": "implement_and_execute_solution",
"operators": [
  {
    "id": "explore_workbook",
    "task": "Explore the input .xlsx workbook to gather structural evidence needed for later implementation
    . Use the inspect_workbook tool once to obtain a compact global preview of all sheets. If the preview
    does not reveal task-required source tables, target ranges, examples, formula patterns, lookup keys, or
     styles, use inspect_workbook_cells or search_workbook to retrieve the smallest discriminative region
    that supplies the missing evidence. Stop as soon as the evidence is sufficient. Write a Markdown file
    named workbook_context.md containing exactly four sections: 'User task instruction', 'Instruction type
    ', 'Answer position', and 'Input xlsx preview'. In the preview, represent every worksheet with its name
    , dimensions, max_row, max_col, rows 1-5 crossed with columns 1-20, the last 5 populated rows and last
    5 populated columns, cells in coordinate=value form (including coordinate= for blanks and exact formula
     text for formulas), truncate cell text longer than 40 characters with '...', include counts of omitted
     populated rows or columns when nonzero, and include useful formatting and workbook metadata returned
    by inspection (number formats, fonts, fills, borders, alignment, merged ranges, hidden dimensions,
    tables, filters, validations, conditional formatting). Omit metadata categories not returned. If
    supplemental cells are needed, append them under the same worksheet preview, labelled with sheet and
    range, in the same coordinate=value format. Do not add a proposed algorithm or general analysis section
    . The output file must be created in the workspace and its path submitted as the output.",
    "inputs": [
      {
        "name": "task",
        "source": "runtime_task",
        "source_operator_id": null,
        "source_output": null
      }
    ],
    "outputs": [
      {
        "name": "workbook_context",
        "kind": "file",
        "description": "Markdown file named workbook_context.md containing the four required sections with
    the compact global preview and any supplemental evidence needed for Stage 2. It must be complete enough
     for the downstream implementer to understand the workbook structure and the task requirements without
    re-inspecting the workbook."
      }
    ],
    "allowed_tools": [
      "inspect_workbook",
      "inspect_workbook_cells",
      "search_workbook",
      "write_file"
    ],
    "operator_version": 1,
    "rules": [
      {
        "id": "rule_001",
        "text": "Call inspect_workbook once to obtain the compact global preview."
      },
      {
        "id": "rule_002",
        "text": "Write and submit one Markdown file named workbook_context.md containing exactly four
    sections: User task instruction, Instruction type, Answer position, and Input xlsx preview."
      },
      {
        "id": "rule_003",
        "text": "In Input xlsx preview, represent every worksheet with name, dimensions, max_row, max_col,
    rows 1-5 crossed with columns 1-20, the last 5 populated rows and last 5 populated columns, cells in
    coordinate=value form including coordinate= for blanks and exact formula text for formulas, truncate
    cell text longer than 40 characters with '...', include counts of omitted populated rows or columns
    when nonzero, and include useful formatting and workbook metadata already returned by inspection. Omit
    metadata categories that were not returned."
      },
      {
        "id": "rule_004",
        "text": "If compact evidence does not reveal a task-required source table, target range, example,
    formula pattern, lookup key, or style, use inspect_workbook_cells or search_workbook for the smallest
    discriminative region that supplies the missing evidence. Append supplemental cells under the same
    worksheet preview, labelled with sheet and range and kept in the same coordinate=value format. Prefer a
     region with new structural information; do not repeatedly sample equivalent rows or widen a request
    merely to collect adjacent cells. Stop as soon as the evidence is sufficient."
      },
      {
        "id": "rule_005",
        "text": "Do not add a proposed algorithm or a general analysis section to workbook_context.md."
      }
    ],
    "sub_skills": []
  },
  {
    "id": "implement_and_execute_solution",
    "task": "Read the workbook_context.md file produced by the explore_workbook operator. Create a reusable
     solution.py that accepts exactly --input <input.xlsx> --output <output.xlsx>. Load only the input
    argument, perform the requested manipulation, preserve unrelated workbook content, and save to the
    output argument. Do not hardcode development paths, row counts, column letters, or preview values;
    inspect the actual workbook in the script. Write graded computed results as literal values unless the
    task explicitly requires formulas. Run the script with the run_spreadsheet_solution tool. If execution
    fails, repair concrete execution errors and rerun. When execution succeeds and creates the output,
    submit solution.py immediately; do not call finish_task afterward.",
```

```
  "inputs": [
    {
      "name": "task",
      "source": "runtime_task",
      "source_operator_id": null,
      "source_output": null
    },
    {
      "name": "workbook_context",
      "source": "operator_output",
      "source_operator_id": "explore_workbook",
      "source_output": "workbook_context"
    }
  ],
  "outputs": [
    {
      "name": "solution_script",
      "kind": "file",
      "description": "The solution.py file that implements the requested spreadsheet manipulation. It
must be a reusable script accepting --input and --output arguments, preserving unrelated workbook
content, and producing the output workbook when executed."
    }
  ],
  "allowed_tools": [
    "read_file",
    "write_file",
    "run_spreadsheet_solution"
  ],
  "operator_version": 3,
  "rules": [
    {
      "id": "rule_006",
      "text": "Use Python for spreadsheet manipulation. Prefer openpyxl when formulas, formatting, sheets
, or named ranges must be preserved. Use pandas only for bulk tabular transformation, then write
results back with openpyxl because pandas.to_excel() can destroy workbook structure."
    },
    {
      "id": "rule_007",
      "text": "Create a reusable solution.py that accepts exactly --input <input.xlsx> --output <output.
xlsx>."
    },
    {
      "id": "rule_008",
      "text": "Load only the input argument, perform the requested manipulation, preserve unrelated
workbook content, and save to the output argument."
    },
    {
      "id": "rule_009",
      "text": "Do not hardcode development paths, row counts, column letters, or preview values; inspect
the actual workbook in the script. When a task-required sheet, table, or column must be located,
resolve it at runtime from the workbook's own structure instead of assumed names or positions: scan the
 header row and match normalized header text (case-insensitive, whitespace-trimmed, tolerant of common
aliases for the same label), falling back to an explicit positional assumption only when no header
matches. Use the resolved sheet and column indices everywhere in place of literal sheet names or column
 letters, so structurally equivalent workbooks with reordered or differently labeled layouts still
execute correctly."
    },
    {
      "id": "rule_010",
      "text": "Write graded computed results as literal values unless the task explicitly requires
formulas. Treat how-to phrasing such as 'how can I create a formula' or 'create a formula that computes
...' as a description of the desired result, not an explicit formula requirement: compute the value in
Python and write the literal value. Consider formulas explicitly required only when the instructions
state the output cells must contain live formulas (e.g., 'enter a formula in each cell', 'the result
must be a formula'). Reason: formula strings written through openpyxl carry no cached result and read
as empty at scoring time, so formula-only cells fail the graded value contract."
    }
  ],
  "sub_skills": [
    {
      "id": "recover_from_write_file_rejection_on_an_occupied_path",
      "name": "Recover from write_file rejection on an occupied path",
      "description": "A conditional recovery method for write_file calls that are rejected because the
target path already exists or the required boolean overwrite argument was missing: classify the
rejection from the protocol feedback, verify any occupied file is stale or unrelated to the current
deliverable, then re-issue the identical write with the corrected overwrite argument instead of
changing the deliverable's content, filename, or location.",
      "applicability": [
        "A write_file call is rejected with a message indicating the target path already exists (e.g., '
File already exists and overwrite is false').",
        "A write_file call is rejected because the required boolean overwrite argument was omitted or
malformed."
      ],
      "objective": "Deliver the intended file content unchanged at the originally chosen path by
correcting the write_file invocation, without renaming, relocating, or altering the deliverable to
dodge the conflict.",
      "exploration_steps": [
        "Read the rejection message and classify it: a missing/malformed overwrite-argument rejection
versus an existing-file rejection on the target path.",
        "For a missing-argument rejection, re-issue the identical write_file call with the boolean
overwrite argument supplied correctly.",
```

```
          "For an existing-file rejection, use read_file on the occupied path and compare its content with
      the deliverable you are about to write; confirm the existing file is stale or unrelated (leftovers from
       a previous item or attempt) rather than a version of the current deliverable that must be kept.",
          "Once the existing file is confirmed stale or unrelated, re-issue the identical write_file call
      with overwrite=true, keeping the content, filename, and location exactly as originally intended.",
          "If the existing file is instead the current deliverable with different content, decide
      explicitly whether it is an obsolete prior attempt and may be overwritten; never silently change the
      deliverable's name or path to avoid the conflict."
        ],
        "stop_conditions": [
          "The write_file call succeeds and the deliverable exists at the originally chosen path with
      exactly the intended content.",
          "If the corrected call is rejected again with a new concrete error, address that error directly
      instead of blindly retrying the same invocation."
        ],
        "tool_guidance": [
          "write_file: retry the identical content with overwrite=true only after the occupied path has
      been inspected; do not rename the file or write it to a different directory to sidestep the rejection
      .",
          "read_file: inspect the occupied path first so the overwrite is a verified decision, not a blind
      retry."
        ],
        "success_invariants": [
          "The submitted deliverable (e.g., solution.py) sits at the path required by the output contract
      with exactly the intended content.",
          "No deliverable is renamed, relocated, or content-modified solely because of a write_file
      rejection.",
          "Recovery costs at most one read_file inspection and one corrected write_file retry per rejection
      ."
        ],
        "rules": [
          {
            "id": "rule_wfor_1",
            "text": "When write_file rejects with an existing-file error, first read_file the occupied path
       and confirm it is stale or unrelated to the current deliverable, then re-issue the identical write
      with overwrite=true."
          },
          {
            "id": "rule_wfor_2",
            "text": "Never respond to a write_file rejection by changing the deliverable's filename,
      directory, or content; fix the invocation (supply overwrite correctly), not the artifact."
          }
        ]
      }
    ]
  }
  ],
  "result_envelope_format_version": "1.0",
  "source_skill_hash": "8a7ba476b2811861c2b69e9471d9c5a773fcdeaf3bea74d0d9ed44a736a3bade",
  "compiler_model_profile": "deepseek_main"
}
```


I.2 OFFICEQA

The selected endpoint is the GLM-5.3 run scoring 70.24% exact-match accuracy. Its evolved graph is version 2; version 1 is the compiled initial graph.

Complete initial source skill.

```
# OfficeQA Skill

Compile this Skill into three stages: `retrieve_evidence`,
`compute_and_verify`, and `format_and_submit`. Pass evidence through
`officeqa_evidence.json` and the computed answer through
`officeqa_solution.json`. The root submits `officeqa_answer.json`.

## Stage 1: Retrieve evidence

Before retrieval, decompose the question into an operand coverage plan: the
requested entity, measure, periods, unit, answer shape, operation, qualifiers,
and every value needed by the downstream calculation. Track each target as
`unresolved`, `grounded`, or `unavailable_after_bounded_search`.

Use official parsed-page evidence already present in the task first. For a
missing operand, search the most likely source file using the exact entity,
period, measure, or table label, then read the smallest span covering the table
title, headers, requested rows, unit, and relevant footnotes. For a sequence,
retrieve all requested periods together when possible and check its endpoints,
order, count, and gaps. A search hit is only a locator, not grounded evidence.

Align every extracted value by semantic role, period, unit, table, row label,
and column label. Distinguish fiscal from calendar periods, amounts from
percentages, totals from components, estimates from actuals, and stocks from
changes. Do not substitute a nearby or numerically similar value. Stop when all
operands are grounded or bounded retrieval is unlikely to resolve the remainder.
Do not calculate the final answer in this stage.
```

````
Write and submit `officeqa_evidence.json` with this schema:

```json
{
  "question": "verbatim question",
  "answer_shape": "required final representation",
  "requested_unit": "requested answer unit",
  "required_periods": ["period"],
  "expected_observation_count": 2,
  "observations": [
    {
      "semantic_role": "operand role in the requested operation",
      "period": "exact period",
      "value": 0,
      "unit": "source unit",
      "source_file": "document filename",
      "line_range": "start-end",
      "table": "table or chart title",
      "row_label": "exact row meaning",
      "column_label": "exact column meaning",
      "evidence": "short source-grounded excerpt or description"
    }
  ],
  "requested_operation": "operation over the observations",
  "qualifiers": ["basis, scope, rounding, or other constraints"],
  "searched_targets": [
    {
      "target": "operand",
      "status": "grounded or unavailable_after_bounded_search",
      "attempts": ["queries and bounded ranges examined"]
    }
  ],
  "exploration_summary": "coverage result and unresolved evidence",
  "exploration_status": "explore_over"
}
```

Store grounded values as literal JSON numbers. Every observation must connect
one downstream operand to its exact source semantics; together the ordered
observations and `requested_operation` form the evidence chain. Never write a
placeholder or inferred value into `observations`. Describe unresolved operands,
searches attempted, and why no supporting span was found in `searched_targets`
and `exploration_summary`. Keep the artifact compact rather than copying search
transcripts or long passages.

## Stage 2: Compute and verify

Read `officeqa_evidence.json` once and confirm every operand before calculation.
Call `run_python` with a short Python script to perform the requested arithmetic
and format the final answer according to the tool's reporting contract. Do not
calculate mentally. Retain full intermediate precision and check operation
direction, sign, scale, period count, unit, and plausibility.

Make the Python script print exactly one JSON object containing `operands`,
`operation`, `calculation`, `unrounded_result`, `final_answer`, and
`limitations`.

Verify that `run_python` exits successfully and that its printed
`final_answer` satisfies the requested shape under the tool's reporting
contract. Write that JSON object unchanged to `officeqa_solution.json`. If
evidence is incomplete, produce the best defensible nonempty answer and record
the limitation rather than abandoning the task; never invent a missing operand.

## Stage 3: Format and submit

Read the solution once and submit its final answer. Follow the question's
explicit format and worked example exactly, including percent versus decimal
space, `%`, sign, requested unit scale, date format, and trailing zeroes. Submit
only the value, date, phrase, or list without explanation.
````

Complete compiled initial tree (v0001).


```
{
  "format_version": "2.0",
  "tree_id": "officeqa",
  "name": "OfficeQA Evidence Retrieval, Computation, and Submission",
  "description": "Compiles the OfficeQA skill into a three-stage pipeline: evidence retrieval, computation
     and verification, and final answer formatting and submission.",
  "root_operator_id": "format_and_submit",
  "operators": [
    {
      "id": "retrieve_evidence",
      "task": "Retrieve and ground all operands needed to answer the OfficeQA question from the runtime task.
       Decompose the question into an operand coverage plan covering the requested entity, measure, periods,
      unit, answer shape, operation, qualifiers, and every value needed by the downstream calculation. Track
      each target as unresolved, grounded, or unavailable_after_bounded_search. Use official parsed-page
      evidence already present in the task first. For a missing operand, search the most likely source file
      using the exact entity, period, measure, or table label, then read the smallest span covering the table
```

```
 title, headers, requested rows, unit, and relevant footnotes. For a sequence, retrieve all requested
periods together when possible and check its endpoints, order, count, and gaps. A search hit is only a
locator, not grounded evidence. Align every extracted value by semantic role, period, unit, table, row
label, and column label. Distinguish fiscal from calendar periods, amounts from percentages, totals
from components, estimates from actuals, and stocks from changes. Do not substitute a nearby or
numerically similar value. Stop when all operands are grounded or bounded retrieval is unlikely to
resolve the remainder. Do not calculate the final answer in this stage. Write and submit
officeqa_evidence.json with the specified schema, storing grounded values as literal JSON numbers.
Every observation must connect one downstream operand to its exact source semantics; together the
ordered observations and requested_operation form the evidence chain. Never write a placeholder or
inferred value into observations. Describe unresolved operands, searches attempted, and why no
supporting span was found in searched_targets and exploration_summary. Keep the artifact compact rather
 than copying search transcripts or long passages.",
"inputs": [
  {
    "name": "task",
    "source": "runtime_task",
    "source_operator_id": null,
    "source_output": null
  }
],
"outputs": [
  {
    "name": "officeqa_evidence",
    "kind": "file",
    "description": "The officeqa_evidence.json file containing the question, answer shape, requested
unit, required periods, expected observation count, ordered observations with grounded values and
source semantics, requested operation, qualifiers, searched targets with status and attempts,
exploration summary, and exploration_status set to explore_over. This file is the complete evidence
chain for the downstream computation."
  }
],
"allowed_tools": [
  "search_officeqa_document",
  "read_officeqa_document",
  "write_file"
],
"operator_version": 1,
"rules": [
  {
    "id": "rule_001",
    "text": "Before retrieval, decompose the question into an operand coverage plan: the requested
entity, measure, periods, unit, answer shape, operation, qualifiers, and every value needed by the
downstream calculation. Track each target as unresolved, grounded, or unavailable_after_bounded_search
."
  },
  {
    "id": "rule_002",
    "text": "Use official parsed-page evidence already present in the task first. For a missing operand
, search the most likely source file using the exact entity, period, measure, or table label, then read
 the smallest span covering the table title, headers, requested rows, unit, and relevant footnotes."
  },
  {
    "id": "rule_003",
    "text": "For a sequence, retrieve all requested periods together when possible and check its
endpoints, order, count, and gaps."
  },
  {
    "id": "rule_004",
    "text": "A search hit is only a locator, not grounded evidence."
  },
  {
    "id": "rule_005",
    "text": "Align every extracted value by semantic role, period, unit, table, row label, and column
label. Distinguish fiscal from calendar periods, amounts from percentages, totals from components,
estimates from actuals, and stocks from changes. Do not substitute a nearby or numerically similar
value."
  },
  {
    "id": "rule_006",
    "text": "Stop when all operands are grounded or bounded retrieval is unlikely to resolve the
remainder. Do not calculate the final answer in this stage."
  },
  {
    "id": "rule_007",
    "text": "Store grounded values as literal JSON numbers. Every observation must connect one
downstream operand to its exact source semantics; together the ordered observations and
requested_operation form the evidence chain. Never write a placeholder or inferred value into
observations."
  },
  {
    "id": "rule_008",
    "text": "Describe unresolved operands, searches attempted, and why no supporting span was found in
searched_targets and exploration_summary. Keep the artifact compact rather than copying search
transcripts or long passages."
  }
],
"sub_skills": []
},
{
  "id": "compute_and_verify",
```

```
    "task": "Read officeqa_evidence.json once and confirm every operand before calculation. Call run_python
     with a short Python script to perform the requested arithmetic and format the final answer according
    to the tool's reporting contract. Do not calculate mentally. Retain full intermediate precision and
    check operation direction, sign, scale, period count, unit, and plausibility. Make the Python script
    print exactly one JSON object containing operands, operation, calculation, unrounded_result,
    final_answer, and limitations. Verify that run_python exits successfully and that its printed
    final_answer satisfies the requested shape under the tool's reporting contract. Write that JSON object
    unchanged to officeqa_solution.json. If evidence is incomplete, produce the best defensible nonempty
    answer and record the limitation rather than abandoning the task; never invent a missing operand.",
    "inputs": [
      {
        "name": "officeqa_evidence",
        "source": "operator_output",
        "source_operator_id": "retrieve_evidence",
        "source_output": "officeqa_evidence"
      }
    ],
    "outputs": [
      {
        "name": "officeqa_solution",
        "kind": "file",
        "description": "The officeqa_solution.json file containing the exact JSON object printed by
    run_python: operands, operation, calculation, unrounded_result, final_answer, and limitations. This
    file holds the computed and verified final answer ready for submission."
      }
    ],
    "allowed_tools": [
      "read_file",
      "run_python",
      "write_file"
    ],
    "operator_version": 1,
    "rules": [
      {
        "id": "rule_001",
        "text": "Read officeqa_evidence.json once and confirm every operand before calculation."
      },
      {
        "id": "rule_002",
        "text": "Call run_python with a short Python script to perform the requested arithmetic and format
    the final answer according to the tool's reporting contract. Do not calculate mentally."
      },
      {
        "id": "rule_003",
        "text": "Retain full intermediate precision and check operation direction, sign, scale, period
    count, unit, and plausibility."
      },
      {
        "id": "rule_004",
        "text": "Make the Python script print exactly one JSON object containing operands, operation,
    calculation, unrounded_result, final_answer, and limitations."
      },
      {
        "id": "rule_005",
        "text": "Verify that run_python exits successfully and that its printed final_answer satisfies the
    requested shape under the tool's reporting contract. Write that JSON object unchanged to
    officeqa_solution.json."
      },
      {
        "id": "rule_006",
        "text": "If evidence is incomplete, produce the best defensible nonempty answer and record the
    limitation rather than abandoning the task; never invent a missing operand."
      }
    ],
    "sub_skills": []
  },
  {
    "id": "format_and_submit",
    "task": "Read the solution once and submit its final answer. Follow the question's explicit format and
    worked example exactly, including percent versus decimal space, %, sign, requested unit scale, date
    format, and trailing zeroes. Submit only the value, date, phrase, or list without explanation. Use
    submit_officeqa_answer to write and submit the final concise answer for official EM/F1 scoring. The
    answer must contain only the value or phrase requested, without explanation or an <answer> wrapper.",
    "inputs": [
      {
        "name": "officeqa_solution",
        "source": "operator_output",
        "source_operator_id": "compute_and_verify",
        "source_output": "officeqa_solution"
      }
    ],
    "outputs": [
      {
        "name": "officeqa_answer",
        "kind": "file",
        "description": "The submitted OfficeQA answer file containing only the final concise answer value
    or phrase, without explanation or an <answer> wrapper, formatted exactly as requested by the question."
      }
    ],
    "allowed_tools": [
      "read_file",
      "submit_officeqa_answer"
```

```
      ],
      "operator_version": 1,
      "rules": [
        {
          "id": "rule_001",
          "text": "Read the solution once and submit its final answer."
        },
        {
          "id": "rule_002",
          "text": "Follow the question's explicit format and worked example exactly, including percent versus
       decimal space, %, sign, requested unit scale, date format, and trailing zeroes."
        },
        {
          "id": "rule_003",
          "text": "Submit only the value, date, phrase, or list without explanation."
        }
      ],
      "sub_skills": []
    }
  ],
  "result_envelope_format_version": "1.0",
  "source_skill_hash": "ec6addd6e7fbc462edfc8aa1bd1b5245da556a14e9eb250946589c06f3030a6c",
  "compiler_model_profile": "deepseek_main"
}
```


Final consolidated tree from the GLM-5.3 run (70.24%, v0002).


```
{
  "format_version": "2.0",
  "tree_id": "officeqa",
  "name": "OfficeQA Evidence Retrieval, Computation, and Submission",
  "description": "Compiles the OfficeQA skill into a three-stage pipeline: evidence retrieval, computation
     and verification, and final answer formatting and submission.",
  "root_operator_id": "format_and_submit",
  "operators": [
    {
      "id": "retrieve_evidence",
      "task": "Retrieve and ground all operands needed to answer the OfficeQA question from the runtime task.
       Decompose the question into an operand coverage plan covering the requested entity, measure, periods,
      unit, answer shape, operation, qualifiers, and every value needed by the downstream calculation. Track
      each target as unresolved, grounded, or unavailable_after_bounded_search. Use official parsed-page
      evidence already present in the task first. For a missing operand, search the most likely source file
      using the exact entity, period, measure, or table label, then read the smallest span covering the table
       title, headers, requested rows, unit, and relevant footnotes. For a sequence, retrieve all requested
      periods together when possible and check its endpoints, order, count, and gaps. A search hit is only a
      locator, not grounded evidence. Align every extracted value by semantic role, period, unit, table, row
      label, and column label. Distinguish fiscal from calendar periods, amounts from percentages, totals
      from components, estimates from actuals, and stocks from changes. Do not substitute a nearby or
      numerically similar value. Stop when all operands are grounded or bounded retrieval is unlikely to
      resolve the remainder. Do not calculate the final answer in this stage. Write and submit
      officeqa_evidence.json with the specified schema, storing grounded values as literal JSON numbers.
      Every observation must connect one downstream operand to its exact source semantics; together the
      ordered observations and requested_operation form the evidence chain. Never write a placeholder or
      inferred value into observations. Describe unresolved operands, searches attempted, and why no
      supporting span was found in searched_targets and exploration_summary. Keep the artifact compact rather
       than copying search transcripts or long passages.",
      "inputs": [
        {
          "name": "task",
          "source": "runtime_task",
          "source_operator_id": null,
          "source_output": null
        }
      ],
      "outputs": [
        {
          "name": "officeqa_evidence",
          "kind": "file",
          "description": "The officeqa_evidence.json file containing the question, answer shape, requested
      unit, required periods, expected observation count, ordered observations with grounded values and
      source semantics, requested operation, qualifiers, searched targets with status and attempts,
      exploration summary, and exploration_status set to explore_over. This file is the complete evidence
      chain for the downstream computation."
        }
      ],
      "allowed_tools": [
        "search_officeqa_document",
        "read_officeqa_document",
        "write_file"
      ],
      "operator_version": 2,
      "rules": [
        {
          "id": "rule_001",
          "text": "Before retrieval, decompose the question into an operand coverage plan: the requested
      entity, measure, periods, unit, answer shape, operation, qualifiers, and every value needed by the
      downstream calculation. Track each target as unresolved, grounded, or unavailable_after_bounded_search
      ."
        },
```

```
        {
          "id": "rule_002",
          "text": "Use official parsed-page evidence already present in the task first. For a missing operand
      , search the most likely source file using the exact entity, period, measure, or table label, then read
       the smallest span covering the table title, headers, requested rows, unit, and relevant footnotes."
        },
        {
          "id": "rule_003",
          "text": "For a sequence, retrieve all requested periods together when possible and check its
      endpoints, order, count, and gaps."
        },
        {
          "id": "rule_004",
          "text": "A search hit is only a locator, not grounded evidence."
        },
        {
          "id": "rule_005",
          "text": "Align every extracted value by semantic role, period, unit, table, row label, and column
      label. Distinguish fiscal from calendar periods, amounts from percentages, totals from components,
      estimates from actuals, and stocks from changes. Do not substitute a nearby or numerically similar
      value. When the requested measure is a total or aggregate and the source block prints component columns
       alongside a printed Total column (whose label may express it as a combination of other columns, such
      as a sum or a sum net of deductions), select the operand column by matching the requested aggregate
      scope to the printed Total column and its combination label --- never by lexical similarity between the
       question wording and a component column label --- and record the selected column's label, including
      its combination expression, as the observation's column_label; only then is the operand grounded. If no
       printed Total column matches the requested aggregate scope, ground the exact component columns needed
      and state the required combination in the evidence instead of substituting a single component for the
      requested total."
        },
        {
          "id": "rule_006",
          "text": "Stop when all operands are grounded or bounded retrieval is unlikely to resolve the
      remainder. Do not calculate the final answer in this stage."
        },
        {
          "id": "rule_007",
          "text": "Store grounded values as literal JSON numbers. Every observation must connect one
      downstream operand to its exact source semantics; together the ordered observations and
      requested_operation form the evidence chain. Never write a placeholder or inferred value into
      observations."
        },
        {
          "id": "rule_008",
          "text": "Describe unresolved operands, searches attempted, and why no supporting span was found in
      searched_targets and exploration_summary. Keep the artifact compact rather than copying search
      transcripts or long passages."
        }
      ],
      "sub_skills": []
    },
    {
      "id": "compute_and_verify",
      "task": "Read officeqa_evidence.json once and confirm every operand before calculation. Call run_python
       with a short Python script to perform the requested arithmetic and format the final answer according
      to the tool's reporting contract. Do not calculate mentally. Retain full intermediate precision and
      check operation direction, sign, scale, period count, unit, and plausibility. Make the Python script
      print exactly one JSON object containing operands, operation, calculation, unrounded_result,
      final_answer, and limitations. Verify that run_python exits successfully and that its printed
      final_answer satisfies the requested shape under the tool's reporting contract. Write that JSON object
      unchanged to officeqa_solution.json. If evidence is incomplete, produce the best defensible nonempty
      answer and record the limitation rather than abandoning the task; never invent a missing operand.",
      "inputs": [
        {
          "name": "officeqa_evidence",
          "source": "operator_output",
          "source_operator_id": "retrieve_evidence",
          "source_output": "officeqa_evidence"
        }
      ],
      "outputs": [
        {
          "name": "officeqa_solution",
          "kind": "file",
          "description": "The officeqa_solution.json file containing the exact JSON object printed by
      run_python: operands, operation, calculation, unrounded_result, final_answer, and limitations. This
      file holds the computed and verified final answer ready for submission."
        }
      ],
      "allowed_tools": [
        "read_file",
        "run_python",
        "write_file"
      ],
      "operator_version": 1,
      "rules": [
        {
          "id": "rule_001",
          "text": "Read officeqa_evidence.json once and confirm every operand before calculation."
        },
        {
          "id": "rule_002",
```

```
          "text": "Call run_python with a short Python script to perform the requested arithmetic and format
      the final answer according to the tool's reporting contract. Do not calculate mentally."
        },
        {
          "id": "rule_003",
          "text": "Retain full intermediate precision and check operation direction, sign, scale, period
      count, unit, and plausibility."
        },
        {
          "id": "rule_004",
          "text": "Make the Python script print exactly one JSON object containing operands, operation,
      calculation, unrounded_result, final_answer, and limitations."
        },
        {
          "id": "rule_005",
          "text": "Verify that run_python exits successfully and that its printed final_answer satisfies the
      requested shape under the tool's reporting contract. Write that JSON object unchanged to
      officeqa_solution.json."
        },
        {
          "id": "rule_006",
          "text": "If evidence is incomplete, produce the best defensible nonempty answer and record the
      limitation rather than abandoning the task; never invent a missing operand."
        }
      ],
      "sub_skills": []
    },
    {
      "id": "format_and_submit",
      "task": "Read the solution once and submit its final answer. Follow the question's explicit format and
      worked example exactly, including percent versus decimal space, %, sign, requested unit scale, date
      format, and trailing zeroes. Submit only the value, date, phrase, or list without explanation. Use
      submit_officeqa_answer to write and submit the final concise answer for official EM/F1 scoring. The
      answer must contain only the value or phrase requested, without explanation or an <answer> wrapper.",
      "inputs": [
        {
          "name": "officeqa_solution",
          "source": "operator_output",
          "source_operator_id": "compute_and_verify",
          "source_output": "officeqa_solution"
        }
      ],
      "outputs": [
        {
          "name": "officeqa_answer",
          "kind": "file",
          "description": "The submitted OfficeQA answer file containing only the final concise answer value
      or phrase, without explanation or an <answer> wrapper, formatted exactly as requested by the question."
        }
      ],
      "allowed_tools": [
        "read_file",
        "submit_officeqa_answer"
      ],
      "operator_version": 1,
      "rules": [
        {
          "id": "rule_001",
          "text": "Read the solution once and submit its final answer."
        },
        {
          "id": "rule_002",
          "text": "Follow the question's explicit format and worked example exactly, including percent versus
       decimal space, %, sign, requested unit scale, date format, and trailing zeroes."
        },
        {
          "id": "rule_003",
          "text": "Submit only the value, date, phrase, or list without explanation."
        }
      ],
      "sub_skills": []
    }
  ],
  "result_envelope_format_version": "1.0",
  "source_skill_hash": "ec6addd6e7fbc462edfc8aa1bd1b5245da556a14e9eb250946589c06f3030a6c",
  "compiler_model_profile": "deepseek_main"
}
```


I.3 DEEPPLANNING-SHOPPING

The selected endpoint is the GLM-5.3 run scoring 69.05% Shopping case accuracy. Because this benchmark fixes a single update coordinate, the source skill is also the complete initial Operator body.

Complete initial source skill and initial Operator body.

```
# DeepPlanning Shopping Skill

## Understand the request

- Separate the request into independently requested product slots and preserve
  the quantity of each slot.
- Record every explicit hard constraint, including product type, brand, color,
  size, demographic, season, stock, rating, sales, delivery time, budget, and
  coupon requirements.
- Use explicit request values first. Call `get_user_info` only when an omitted
  size, demographic, destination, or owned-coupon fact affects the answer.
- Respect the task objective: Level 1 minimizes product subtotal, Level 2 must
  satisfy the total-price range and then minimize cost, and Level 3 minimizes
  final price after valid owned coupons.

## Find and verify products

- Start each slot with one discriminative `search_products` query using the
  requested product type, name, or brand. Carry the returned IDs through later
  checks instead of restarting from the full catalog.
- Apply exact brand, color, size, coupon, and numeric filters when relevant. If
  a search is empty, make at most one justified broader search for that slot.
- Use `get_product_details` in bounded batches to verify every hard constraint
  and obtain prices, stock, and coupon applicability. A search result is only a
  candidate, not proof that the product satisfies the request.
- Apply every written comparison independently. For example, `fewer than 20
  one-star or two-star reviews'' requires each count to be below 20 unless the
  request explicitly asks for their sum.
- For delivery constraints, call the official `calculate_transport_time` once
  for each remaining candidate with its single product ID and the exact
  destination. Compare the returned days against the requested condition; do
  not repeat a completed candidate check.
- Permanently reject a product after a grounded hard-constraint failure. Keep
  all verified alternatives needed for cross-slot budget or coupon
  optimization; do not discard a feasible product merely because another is
  cheaper.

## Choose the combination

- Select exactly one verified product per requested slot and preserve each
  quantity. Never substitute one product for two different requested slots.
- Account for quantity when computing the subtotal.
- For Level 2, start with the cheapest valid combination and use verified
  alternatives only when necessary to enter the requested total-price range.
- For Level 3, compare supported product-and-coupon combinations. Apply only
  coupons owned by the user and applicable to the selected products. Respect
  coupon thresholds, repeated-discount semantics, and owned quantity;
  cart-wide coupons may depend on the combined subtotal.
- Never invent product IDs, prices, constraint values, or coupons. If no fully
  verified candidate exists, use the best supported evidence without entering
  a repeated search loop.

## Build and verify the cart

- Add every selected product and quantity with `add_product_to_cart`, and every
  selected coupon and quantity with `add_coupon_to_cart`.
- Call `get_cart_info` once after construction. Check product IDs, coupon names,
  quantities, and the number of requested slots against the plan. Use deletion
  tools only to repair a concrete mismatch.
- Once the inspected cart is complete and correct, finish directly without
  restarting product discovery or adding any extra item or coupon.
```

Complete compiled initial single-Operator topology.

```
{
  "format_version": "single_operator_v1",
  "benchmark": "DeepPlanning-Shopping",
  "root_operator_id": "solve_deepplanning",
  "operators": [
    {
      "id": "solve_deepplanning",
      "kind": "single_stage",
      "inputs": ["runtime_task"],
      "outputs": ["shopping_plan"],
      "policy_body": "verbatim DeepPlanning initial skill printed immediately above"
    }
  ]
}
```

Final consolidated single-Operator policy from the GLM-5.3 run (69.05%).

```
# DeepPlanning Shopping Skill

## Understand the request
```

```
- Separate the request into independently requested product slots and preserve
  the quantity of each slot.
- Record every explicit hard constraint, including product type, brand, color,
  size, demographic, season, stock, rating, sales, delivery time, budget, and
  coupon requirements.
- Use explicit request values first. Call `get_user_info` only when an omitted
  size, demographic, destination, or owned-coupon fact affects the answer.
- Respect the task objective: Level 1 minimizes product subtotal, Level 2 must
  satisfy the total-price range and then minimize cost, and Level 3 minimizes
  final price after valid owned coupons.

## Find and verify products

- Start each slot with one discriminative `search_products` query using the
  requested product type, name, or brand. Carry the returned IDs through later
  checks instead of restarting from the full catalog.
- Apply exact brand, color, size, coupon, and numeric filters when relevant. If
  a search is empty, make at most one justified broader search for that slot.
- Use `get_product_details` in bounded batches to verify every hard constraint
  and obtain prices, stock, and coupon applicability. A search result is only a
  candidate, not proof that the product satisfies the request.
- Apply every written comparison independently. For example, `fewer than 20
  one-star or two-star reviews'' requires each count to be below 20 unless the
  request explicitly asks for their sum.
- For delivery constraints, call the official `calculate_transport_time` once
  for each remaining candidate with its single product ID and the exact
  destination. Compare the returned days against the requested condition; do
  not repeat a completed candidate check.
- Permanently reject a product after a grounded hard-constraint failure. Keep
  all verified alternatives needed for cross-slot budget or coupon
  optimization; do not discard a feasible product merely because another is
  cheaper.
- Treat known user-profile attributes as implicit hard constraints on every slot whose text does not
      explicitly restate them, especially when the request names the user as the recipient (e.g., "for myself
      "). This covers the profile demographic (reject youth, kids, or unisex items when the profile states a
      specific demographic) and the profile's standard size for the candidate's product category, enforced on
       the matching size variant exactly like a written size constraint. Apply both checks at candidate
      admission before any price comparison --- even if a cheaper product passes all written numeric
      constraints --- and repeat them during final cart verification.

## Choose the combination

- Select exactly one verified product per requested slot and preserve each
  quantity. Never substitute one product for two different requested slots.
- When one verified product satisfies two different requested slots, neither
  discard it nor use it for both: assign it to exactly one slot and fill the
  other from the retained verified alternatives, choosing the assignment that
  yields the cheaper valid combined total.
- Account for quantity when computing the subtotal.
- Level 1 funnel (multi-slot cheapest combination): in each slot, spend checks from cheapest to most
      expensive --- explicit plus profile-implied attribute/size admission, then one bounded detail batch
      with every written rating-count or sales threshold evaluated independently per candidate, and the
      official transport-time check last, only for survivors in slots that state a delivery constraint. Keep
      every verified alternative until selection, then pick the cheapest fully verified candidate per slot;
      because slots are independent, the per-slot minima together prove the minimum subtotal. Reject a
      cheaper candidate only on a grounded hard-constraint failure, never on price alone.
- Level 2 total-price range: once every slot has fully verified candidates, build each slot's price ladder
      from every verified, priced alternative and sum the per-slot cheapest prices as the baseline; slots
      with a single feasible candidate contribute a fixed subtotal. If the baseline already lies inside the
      band, keep it. If it lies below the floor, compute the shortfall and apply the smallest set of upgrades
       over the retained alternatives whose delta sum clears the floor; a cheaper alternative that would drop
       the total below the floor is infeasible, and per-slot cheapest picks that merely happen to land in-
      band do not suffice when a swap yields a smaller in-range total. Before declaring any slot unique or
      the band unreachable, re-read the already-fetched detail and delivery results for that slot; never fall
       back to a cheapest cart outside the band while any ladder may be incomplete. Stop once the cheapest in
      -band combination is identified from verified prices; do not guess upgrades or restart searches.
- For Level 3, compare supported product-and-coupon combinations. Apply only
  coupons owned by the user and applicable to the selected products. Respect
  coupon thresholds, repeated-discount semantics, and owned quantity;
  cart-wide coupons may depend on the combined subtotal.
- Never invent product IDs, prices, constraint values, or coupons. If no fully
  verified candidate exists, use the best supported evidence without entering
  a repeated search loop.

## Build and verify the cart

- Add every selected product and quantity with `add_product_to_cart`, and every
  selected coupon and quantity with `add_coupon_to_cart`.
- Call `get_cart_info` once after construction. Check product IDs, coupon names,
  quantities, and the number of requested slots against the plan. Use deletion
  tools only to repair a concrete mismatch.
- Once the inspected cart is complete and correct, finish directly without
  restarting product discovery or adding any extra item or coupon.
```

I.4 SEARCHQA

The selected endpoint is the GLM-5.3 run scoring 86.31% exact-match accuracy. As in DeepPlanning, the benchmark fixes one update coordinate, so the source skill is the complete initial Operator body.

Complete initial source skill and initial Operator body.

```
# Question Answering Skill

(No learned rules yet. Rules will be added through the reflection process.)
```

Complete compiled initial single-Operator topology.

```
{
  "format_version": "single_operator_v1",
  "benchmark": "SearchQA",
  "root_operator_id": "answer_question",
  "operators": [
    {
      "id": "answer_question",
      "kind": "single_stage",
      "inputs": ["question", "retrieved_context"],
      "outputs": ["answer"],
      "policy_body": "verbatim SearchQA initial skill printed immediately above"
    }
  ]
}
```

Final consolidated single-Operator policy from the GLM-5.3 run (86.31%).

```
# Question Answering Skill

(No learned rules yet. Rules will be added through the reflection process.)
- Question-echo span minimization (demonstrative/definitional framing): When the question frames the answer
      with a demonstrative category phrase or definiendum (e.g., "this <category>", "this possible <group>",
      or "a <generic subject> that ..."), locate the demonstrative slot or defined term the question leaves
      open; extract from the supplied context only the distinguishing value or head noun that fills that slot
      . Even when the candidate is a fixed lexicalized compound name, test each component token before
      emitting it, and test restatement at the synonym level, not lexical repetition alone: strip any
      component whose content merely restates what the question itself supplies --- a modifier repeating an
      attribute the question already describes, or a generic medium/category head noun that repeats or near-
      synonymously paraphrases the question's category noun (e.g., "power" or "energy" restating an "
      electricity" category word) --- whenever the remaining tokens still conventionally name the referent on
       their own; keep the full compound only when every component contributes distinguishing content the
      question omits, with fixed lexicalized proper names and titles still governed by verbatim emission.
      Then strip leading articles and any token the question itself already states or near-synonymously
      restates, such as the category noun, group descriptor, or a restated generic subject. Stop once the
      span contains only tokens the question neither supplies nor paraphrases, and emit that minimal span
      inside <answer>...</answer>.
- **Answer-type binding (anchor entity vs. requested type)**: Trigger: the clue names a salient anchor entity
       --- a deity or mythological figure, a formal name, a portrayed character or actor, a person tied to a
      distinguishing fact, or an artifact bearing a distinctive inscription or spelling variant --- but its
      interrogative head explicitly requests a different type, typically a "this <category noun>" phrase or a
       relational phrase such as "called him this". Action: (1) parse the stated category or relational
      phrase as the answer type before considering any named entity in the clue; (2) pivot from the anchor
      entity to the entity of the requested type --- the real-world referent behind a mythic name, the bare
      component noun of the organism or object, the in-fiction character rather than actor or series, the
      group's relational epithet rather than the formal name, the standard modern name behind the inscription
       variant; (3) keep any mechanism, process, or supporting detail outside the tagged answer span.
      Boundary: stop at the bare noun or canonical name of the requested type --- no adjacent entities, no
      articles, no qualifiers specific to the myth, formal name, portrayal, or inscription.
- When emitting the final answer span, strip leading articles ("a", "an", "the") and never append
      parenthetical aliases, abbreviations, full birth names, or subspecies names; strip only tokens the
      question itself supplies or generic articles, never a qualifier the question forces the solver to infer
      . Unresolved-container exception: when a clue refers to a container only through an unresolved
      possessive or relational phrase (e.g., "in its state", "in its country") without naming it, resolve
      that container from the supplied context or world knowledge and emit the conventional "Head, Container"
       form; when the question itself names the container, the bare head entity remains the correct minimal
      span.
- Granularity matching for descriptive clues: before emitting the span, compare its specificity to the
      specificity of the clue's own descriptors. When a demonstrative or definitional clue enumerates only
      class-level traits (e.g., habitat class, overall body form, broad behavior) and supplies no detail that
       distinguishes a particular species or subtype, stop at the bare class-level head noun and drop any
      species, habitat, or similar qualifier token --- above all any qualifier that contradicts a descriptor
      the clue itself states. Retain a more specific multiword name only when the clue itself provides the
      distinguishing detail that pins the answer to that finer level.
- **Term-variant to canonical-name resolution**: Trigger: the clue fills the open answer slot with a non-
      canonical term variant --- a colloquial alias, an acronym or initialism, a foreign-language term (often
       with a locale qualifier), a translation gloss ("the name means '...'"), or a compositional name-
      meaning gloss --- instead of the canonical name. Action: (1) locate the variant in the clue; (2)
      resolve its canonical counterpart from the supplied context --- map the alias to its entity, expand the
```

acronym, translate the foreign term to its English equivalent, or compose the meaning-gloss parts into the fixed lexicalized name; (3) validate the candidate against the clue's residual constraints (category, locale, function). Boundary: emit the bare canonical form in its most common transliteration or spelling --- no articles, no component-by-component spellings, no hyphenation or alias variants, no question-supplied category tokens --- and stop once the span is the canonical name alone.
- **Grammatical-number fidelity to the conventional standalone name**: Trigger: the question's open slot is a generic category noun or a definitional term (e.g., "this <category>", "this is the term for X"), so grammatical number must be deliberately chosen at emission. Action: (1) Before fixing the number, scan the demonstrative's own clause for residual plural cues --- a contrastive exemplar introduced by negation ("not <plural noun>", "rather than <plural noun>") or parallel plural head nouns in the same construction --- and, when such cues conflict with the demonstrative's surface number or the citation form, mirror those contrast/parallel nouns' number, since a contrastive exemplar typically preserves the original clue's construction that the expected answer follows. (2) Only absent such cues, fix the number solely from the referent's conventional standalone dictionary/citation form, bidirectionally --- singular for terms conventionally cited in the singular, plural for terms conventionally cited in the plural --- and never inherit it from the clue's other surface cues: a singular demonstrative ("this") does not license singularizing a plural-conventional term, an inclusive exemplar or category phrase (e.g., "nonconductors like plastics") does not license pluralizing a singular-conventional term, and a longer source phrase containing the referent does not impose its own number; when the conventional form is uncertain, match the number of the clue's own parallel construction or the supplied context's citation of the term. Boundary: emit the bare term alone with the number chosen above; under normalization that does not stem, a single inflectional mismatch shares zero tokens and defeats both exact match and token F1 despite correct entity identification.
- **Primary-term selection over coordinated alternatives**: When the supplied context defines an origin, source, or composition as a primary term plus secondary alternatives ("prepared from X, sometimes/also Y", "usually X, occasionally Z") and the clue leaves a single open answer slot, emit only the primary, most strongly associated term. A plural question slot such as "these <category>" is not an enumeration cue --- one noun (including a plural-invariant noun) satisfies it. Boundary: prune coordinating conjunctions and every secondary alternative before emitting; stop once the span names the primary term alone.
- **Personal-name minimal standalone form**: When the answer slot is filled by a personal name --- a person requested through a generic slot (e.g., "this president") or pinned by a distinguishing fact --- emit the shortest conventional standalone form: given name plus surname. Strip middle initials, middle names, generational suffixes, and expansions of initials before emitting, and retain such a component only when the question or the supplied context itself supplies it or makes it the distinguishing fact (e.g., a rare middle name). Boundary: stop once the span is given name plus surname --- surplus name components add no distinguishing content the clue lacks and defeat exact match while diluting token F1.
- **Interrogative-less titled-name prompts --- associated-entity reading**: Trigger: the prompt consists solely of one or more titled person names --- an honorific or office title attached to a personal name, including coordinated "A & B" pairs of such names --- with no interrogative, category phrase, or other question framing. Action: (1) treat the honorific not as a cue to describe or define the named person but as a relational pointer of the form "<title> of X"; (2) for a single name, resolve from the supplied context the polity, house, or institution that the title points to; for a coordinated pair, resolve the single entity both titles share; (3) prune the title, any ordinal, every relational clause, and all biographical detail. Boundary: stop once the span is the bare proper noun of the referenced or shared entity alone --- no titles, no ordinals, no "of X" phrases, no multi-clause description of how the names relate.
- **Abbreviation expansion from the context's definitional punctuation**: Trigger: the clue's open answer slot is filled only by an acronym, initialism, or portmanteau, and the clue asks what it "stands for" or "is short for". Action: scan the supplied context for definitional punctuation patterns --- "Full Expansion (ABBR)" parentheses or "ShortName: FullExpansion" colon titles --- and copy the complete expansion verbatim from that pattern, setting aside any non-answer framing in the clue. Boundary: emit the entire expansion as one fixed lexicalized unit and stop; never prune a component even when the question's category phrase paraphrases it, and append no articles, category nouns, or parenthetical aliases --- every component is needed to reconstruct the abbreviation, so trimming any component defeats exact match and token F1 despite correct identification.
- **Conjunctive anchor verification for demonstrative slots**: Trigger: the clue frames the answer with a demonstrative category phrase ("this <category>") surrounded by multiple distinguishing anchors --- dates, places, agents, numeric values, or downstream consequences. Action: (1) enumerate every anchor the clue states rather than seizing on the single most salient one; (2) bind the answer type from the category noun, then intersect all anchors to pin a single entity of that type, treating each stated year as a hard filter over candidate entities and adjusting granularity to the requested category's level (e.g., a named locality or landmark is raised to its containing country; a defunct or proposed territory is traced to its admitting successor state); (3) when context is supplied, verify every anchor jointly against it before committing to the entity. Boundary: stop once exactly one entity of the requested type satisfies every anchor, and emit its bare canonical proper noun or head noun --- excluding every anchor token, the category noun, articles, and any adjacent entity or qualifying detail.
- **Conventional citation-name fidelity for person slots**: Trigger: the answer slot is a person (a demonstrative person phrase such as "this man", or a possessive-pronoun person clue), and the referent's conventional standalone public/citation name deviates from plain given-name-plus-surname --- a middle initial intrinsic to the fixed lexicalized name, or a famous nickname that has replaced the formal given name. Action: (1) resolve the person by jointly intersecting the clue's stated anchors; (2) determine the name form the person is conventionally and publicly cited by; (3) emit that form verbatim --- retain an intrinsic middle initial even when neither the question nor the supplied context restates it, and keep a conventional nickname inline as the given-name component when it is the component the person is best known by. Boundary: stop at exactly the conventional citation name --- never prune an intrinsic initial, never substitute the formal given name for the nickname, and never parenthesize or quote the nickname; mechanical minimization (dropping middle initials, generational suffixes, expansions of initials) applies only to removable surplus components, never to components intrinsic to the conventional name.
- **Lexical-category (part-of-speech) fidelity to the elicited slot**: Trigger: a bare or interrogative-less prompt --- e.g., a locational construction like "In X & Y" --- leaves an open slot that an adjective naturally fills (what the named places share or are located in). Action: (1) resolve the shared country/region/entity from the named anchors; (2) before emitting, test the candidate's part of speech against the lexical category the clue's construction elicits; (3) when the slot elicits a descriptor of the places rather than a naming of the location, convert the resolved country/region noun to its adjectival demonym. Boundary: stop at the demonym adjective alone --- no article, no country noun --- because under normalization that does not stem, a noun/adjective derivational alternation shares zero tokens and defeats both exact match and token F1 despite correct entity identification, exactly parallel to an inflectional-number mismatch.

```
- **Anti-echo anchor pivot**: Never emit a span that echoes material the clue itself supplies as given
      information. Before emitting, test the candidate against the clue: when it repeats a named work, title,
       person, or date --- above all one supplied by an appositive or colon construction with no
      interrogative or category phrase --- treat that named entity as an anchor, not the answer. Pivot to the
       distinct associated entity the framing implies (a noted adaptation points to its source work; a
      portrayal to the portrayed figure; a dated work to its creator), resolving it from the supplied context
       or canonical world knowledge. Boundary: stop at the bare canonical name of that distinct associated
      entity, sharing no tokens with the anchor; never echo the anchor, its date, or its framing.
- **Verbatim emission of fixed lexicalized proper names and titles**: Trigger: the resolved answer is a fixed
       lexicalized proper name --- a work title, a character name, or a conventional conflict/event name ---
      rather than a compositional noun phrase. Action: resolve the entity by intersecting the clue's anchors
      (dates, author or belligerent, signature descriptor) against the requested category, then emit the
      conventional citation form exactly as conventionally written, preserving internal hyphens, possessive
      apostrophes, capitalization, and any leading article intrinsic to the title. Generic minimization
      heuristics --- leading-article stripping and component-token echo testing --- govern compositional noun
       phrases, not fixed citation names; prune no component of a fixed name as a question echo. Boundary:
      stop at the complete canonical name; normalization forgives article and punctuation variation, so
      character-level fidelity to the canonical form is cost-free, while dropping an intrinsic component
      risks exact-match loss for zero gain.
- **Modifier licensing --- clue-stated content only**: every modifier token in the emitted span must be
      licensed by the clue's own descriptors or by the supplied context; world knowledge alone never licenses
       a qualifier. Before emitting, test for two recurring unlicensed sources and strip them: (1) qualifiers
       inherited from resolving an anchor proper name --- when a resemblance frame ("shaped like one of these
      ") elicits a bare generic category noun, the subtype or medium of the anchor's specific historical
      referent is given information the clue already supplies via the name, not answer content; (2)
      qualifiers imported from iconic knowledge of a locale named alongside a foreign-term gloss --- emblems,
       heraldic colors, emblem species --- noting that a first-person deictic phrase ("our <noun>") anchors
      to the quiz's home context, not the foreign locale. Boundary: stop at the licensed bare head noun; a
      single surplus modifier zeroes exact match and cuts token F1 despite correct entity identification.
- **Scare-quoted category word --- fixed-name precedence test**: When the clue places the category noun
      inside quotation marks (e.g., the "War to End All Wars"), the quotes mark the word as non-generic --- a
       mention of the word itself, often an intrinsic component of the answer's conventional proper name.
      Action: before any echo-stripping, resolve the answer entity from the clue's anchors and test whether
      its fixed citation name incorporates the quoted word; when it does, classify the candidate as a fixed
      lexicalized proper name, so verbatim emission takes precedence and question-echo minimization --- which
       governs compositional noun phrases only --- never strips the quoted component. Boundary: stop at the
      complete conventional name of the event, crisis, conflict, or work with every intrinsic component
      intact; dropping a quoted component defeats exact match while normalization forgives only article and
      punctuation variation.
- **Unseen-image deictic fallback**: Trigger: the clue contains a deictic image reference ("seen here", "
      shown here", "pictured here") but no image is actually supplied with the context. Action: (1) do not
      stall, abstain, or wait on the missing visual --- treat the reference as a non-load-bearing anchor,
      since the clue's text alone must identify the referent; (2) bind the answer type from the category noun
      , then enumerate the small candidate set implied by that category plus every stated textual anchor (
      locales, dates, qualitative descriptors, outcomes); (3) use the most distinctive qualitative descriptor
       or outcome anchor to discriminate within that candidate set to exactly one entity, eliminating sibling
       members of the same category; (4) emit at the clue's granularity --- the bare canonical proper noun
      when the category noun is not intrinsic to the referent's citation name, and the full fixed compound
      when the remaining tokens alone do not conventionally name the referent (a category-noun component of a
       fixed compound is not an echo to strip). Boundary: stop once the residual textual anchors jointly pin
      a unique entity of the requested type; the absent image neither blocks extraction nor licenses any
      extra qualifier.
```

Interpretation. The two multi-stage endpoints change local Operator instructions and may attach local Rules or Sub-skills while preserving graph topology and I/O contracts. The two single-stage endpoints preserve their one-node topology and change only the node's policy text. Thus every endpoint above is directly executable under the benchmark structure defined in Table A2; no hidden tree topology or omitted policy file is required to interpret it.