SKILL TREE: HIERARCHY-GUIDED EXPERIENCE CONSOLIDATION FOR SELF-EVOLVING AGENT SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents increasingly improve through reusable skills distilled from execution experience. Multi-trajectory skill evolution enables broader evidence aggregation, but many existing methods rely on LLM-generated consolidation, which can alter or omit useful details while leaving semantic redundancy unresolved. We propose Skill Tree, a framework that instead groups edits by their structural context, explicitly removes redundant guidance, and preserves useful content without generative rewriting. Skill Tree further organizes accumulated knowledge into a budgeted core skill and supplementary references, reorders core content under importance and workflow constraints, and retrieves task-relevant reference subtrees at inference time. Experiments across skill construction, out-of-distribution generalization, and cross-model reuse show substantial improvements in downstream task performance, with controlled ablations further demonstrating the importance of structure-aware consolidation, organization, and access.

1 INTRODUCTION

Large language model (LLM) agents increasingly rely on external resources beyond model parameters to support task execution, including tools, persistent context, and reusable skills (Anthropic, 2024; 2025). Agent skills package procedural knowledge, tool-use instructions, scripts, and domain-specific guidance into reusable resources that can be loaded when needed, allowing agents to acquire specialized capabilities without modifying the underlying model (Wang et al., 2023). Because these resources are external and editable, they can also be improved as agents accumulate execution experience. Recent work therefore studies skill evolution, where successful and failed trajectories are distilled into persistent guidance that improves subsequent task execution (Zhao et al., 2024).

Existing skill-evolution methods differ in how much experience they accumulate before revision. Single-trajectory incremental methods update the skill after each execution (Zhang et al., 2026b; Zhou et al., 2026; Ouyang et al., 2026a; Yang et al., 2026a), enabling rapid adaptation but making updates sequential and locally conditioned. Multi-trajectory methods instead aggregate evidence from several executions before revision (Ni et al., 2026; Yang et al., 2026b; Chen et al., 2026; Jiang et al., 2026; Alzubi et al., 2026), enabling cross-example comparison but shifting the challenge to reliable consolidation: the system must determine which information is redundant, which distinctions should be preserved, and which details must survive the aggregation process.

For multi-trajectory methods that consolidate experience through LLM-based merging, deciding what to retain is coupled with regenerating the retained content. This makes skill consolidation an instance of a broader context-rewriting problem: prior work on evolving prompts and agent memories has shown that repeatedly rewriting external context can progressively compress or distort useful information (Zhang et al., 2026a). Multi-trajectory skill evolution inherits this preservation problem while introducing an additional requirement: redundant experience must also be identified and removed across trajectories. Trace2Skill (Ni et al., 2026) provides a representative example through its recursive merging of trajectory-local patches, while several recent methods likewise rely on LLM-generated summarization or merging for experience integration (Yang et al., 2026b; Jiang et al., 2026; Chen et al., 2026). Figure 1 (left) illustrates this limitation. Our later analysis shows that effective consolidation must balance two competing goals: removing redundant guidance while preserving context-specific information.

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

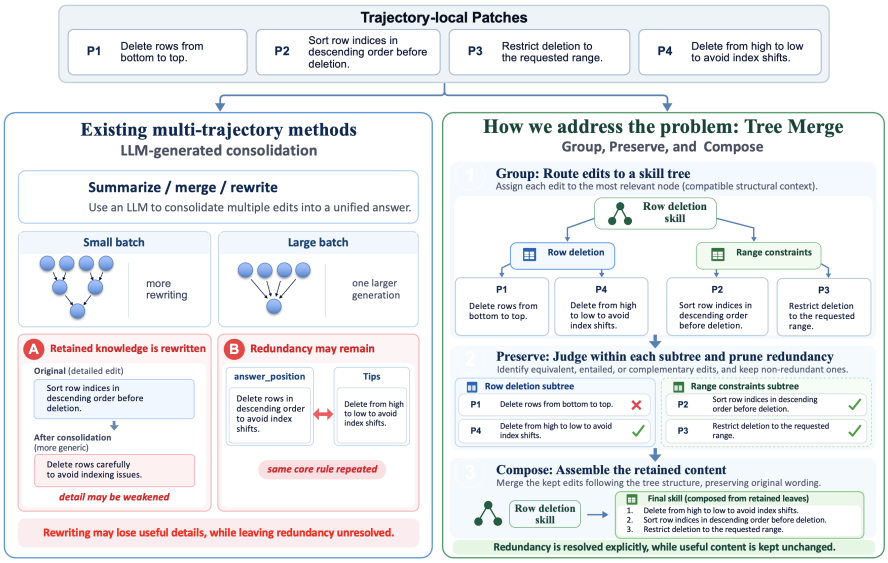

Figure 1: Processing the same trajectory-local patches with rewriting-based merging (left) and Tree Merge (right). Smaller batches produce deeper merge hierarchies and repeated summarization, risking loss of useful details. Larger batches reduce rewriting depth but may still leave semantic redundancy unresolved. Tree Merge instead groups patches by their target tree nodes, removes redundant candidates within compatible contexts, and assembles retained content without rewriting it.

We address this problem by keeping useful edits unchanged and explicitly removing redundant ones, instead of repeatedly rewriting them through generative merging, as in Figure 1 (right). Candidate edits are compared only within compatible structural contexts, allowing redundant guidance to be filtered while complementary content is preserved without rewriting. This separates redundancy resolution from regeneration of the knowledge being retained.

Building on this principle, we introduce Skill Tree, a framework that uses document structure to guide skill consolidation, organization, and access. **Tree Merge** groups edits by target section and removes redundant candidates without rewriting retained content, reducing opportunities for detail loss. **Tree Shaping** controls context growth by allocating related content together between a budgeted core and supplementary references. **Tree Reordering** brings important core guidance forward while respecting structural and workflow dependencies. **Inference-time Routing** selects task-relevant reference subtrees under a separate budget, supplementing the always-loaded core without loading all accumulated knowledge. Together, these modules determine which experience to retain, how to organize it, and what to present during execution.

Our contributions are threefold:

- We introduce **Skill Tree**, a unified framework that uses document structure to coordinate skill consolidation, organization, ordering, and access across the skill lifecycle.
- We design four structure-aware modules: **Tree Merge** for redundancy-aware experience consolidation, **Tree Shaping** for budgeted core–reference allocation, **Tree Reordering** for importance-aware presentation, and **Inference-time Routing** for task-conditioned reference access.
- We conduct extensive evaluations across skill construction, OOD generalization, cross-model reuse, and controlled ablations to assess the effectiveness and robustness of Skill Tree.

2 Related Work

Agent Skill Evolution. Some methods revise skills after individual executions. SkillFlow (Zhang et al., 2026b) studies task-by-task skill discovery, repair, and retention; Memento-Skills (Zhou et al.,

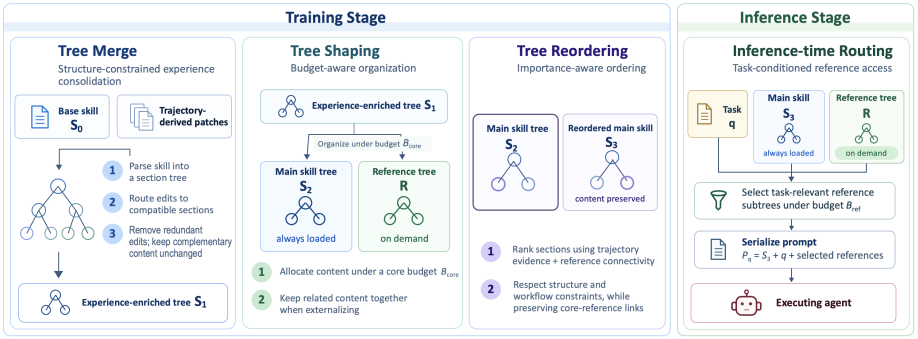

Figure 2: Overview of Skill Tree. Tree Merge consolidates trajectory-derived edits, Tree Shaping organizes a budgeted main skill and references, Tree Reordering prioritizes retained content while preserving links, and inference-time routing retrieves task-relevant references.

2026) attributes failures to individual skills and validates targeted revisions; SkillOS (Ouyang et al., 2026a) learns a curator that maintains the skill repository after each task; and SkillMaster Yang et al. (2026a) jointly trains execution and skill-editing policies. These methods support rapid adaptation, but skill evolution proceeds through sequential, trajectory-local updates.

Another line aggregates evidence across multiple trajectories before revising the skill. Trace2Skill independently distills trajectory-local patches and consolidates them through hierarchical LLM merging (Ni et al., 2026). SkillOpt generates edits from rollout minibatches and hierarchically aggregates them before validation-based acceptance Yang et al. (2026b). SkillCAT improves the quality of candidate edits through contrastive extraction and assessment, while retaining a hierarchical merging stage for accepted patches (Chen et al., 2026). XSkill summarizes and critiques experience across multiple rollouts to accumulate task-level skills and action-level experience Jiang et al. (2026), while EvoSkill proposes revisions from batches of failed executions and historical feedback Alzubi et al. (2026).

These methods benefit from cross-trajectory evidence, but many still rely on LLM-generated merging to consolidate experience. This couples deciding what to retain with rewriting the retained content. Our work instead explicitly removes redundant edits within compatible contexts while preserving complementary content unchanged.

Structured Skill Organization and Access. Beyond skill evolution itself, recent work studies how accumulated knowledge should be organized and exposed during execution. SkillReducer separates persistently available instructions from supplementary material to reduce context cost (Gao et al., 2026). SkillCAT (Chen et al., 2026) and XSkill (Jiang et al., 2026) selectively retrieve structured skill knowledge according to the current task. More broadly, hierarchical retrieval methods such as RAPTOR organize external text into multi-level structures for selective access (Sarthi et al., 2024). Our work differs in treating structure not only as an inference-time access mechanism, but as a shared representation across the full skill lifecycle: the same section structure constrains experience consolidation, guides the placement and ordering of retained knowledge, and defines the units used for task-conditioned access.

3 Method

3.1 Problem Setup and Hierarchical Representation

We consider constructing a reusable agent skill from a base skill S_0 and training-time execution trajectories $\mathcal{T}_{S_0} = \{\tau_i\}_{i=1}^{N}$, including both successful and failed executions collected using S_0. We distinguish a target budget B_{skill} for the main skill document and a per-task reference budget B_{ref} for supplementary content injected at inference time.

We represent each Markdown document as an ordered rooted tree $T_d = (V_d, E_d)$ induced by its heading hierarchy. Each node $v \in V_d$ contains a heading title $h(v)$ and its direct body $b(v)$, exclud-

ing descendant sections. We denote the complete heading path to v by $\pi(v)$ and the subtree rooted at v by T_v, which respectively capture the section context and its nested content.

Figure 2 summarizes the framework. At training time, we first consolidate trajectory-derived edits into S_0 without rewriting retained guidance, obtaining an experience-augmented tree S_1, derive it into a budgeted main-skill tree S_2 and a reference tree R, and reorder S_2 into S_3. At inference time, task-relevant subtrees from R augment the persistently available S_3:

$$\underbrace{S_0 \to S_1 \to (S_2, R) \to (S_3, R)}_{\text{training-time construction}} \qquad \underbrace{(q, S_3, R) \to \mathcal{P}_q}_{\text{inference-time routing}}. \tag{1}$$

Here, $\mathcal{P}_q$ denotes the serialized prompt context presented to the execution agent for task q.

3.2 Tree Merge: Structure-Constrained Consolidation

Rather than generatively merging trajectory-derived edits, we formulate consolidation as structure-constrained filtering: useful edits are retained without rewriting their content, while redundant edits are explicitly removed. Because semantically similar edits may belong to different section contexts, we first constrain redundancy comparison by structural location.

Given the base skill S_0 and training trajectories $\mathcal{T}_{S_0}$, we distill each trajectory into structured candidate patch edits and collect them into a set $\mathcal{P}$. We parse S_0 into a Markdown tree, resolve each patch to its target node, and group patches by target node subject to operation and anchor compatibility. Within each group, an NLI model identifies equivalence and directional entailment, removing redundant edits while retaining complementary guidance. The surviving edits are then applied to the tree without regenerating their content, yielding the experience-augmented tree S_1. Detailed target resolution, grouping, deduplication, and provenance tracking are provided in Appendix A.1. The resulting S_1 is then organized under the main-skill budget in the next stage.

3.3 Tree Shaping: Derivation

The resulting S_1 accumulates guidance distilled from training trajectories, but persistently exposing the full tree becomes increasingly costly as the skill grows, while many details may be irrelevant to a given task. Prior work similarly shows that full-context exposure can be token-inefficient and that task-adaptive context selection provides a more efficient alternative (Taguchi et al., 2025). We therefore treat skill organization as content placement rather than deletion: guidance is divided between persistently visible content and task-conditioned references under a fixed main-skill budget.

Budget-aware content planning. We organize S_1 into two execution-time representations: a persistently available main-skill tree S_2 and a reference tree R for task-conditioned access. The planner operates on structurally coherent content groups rather than individual paragraphs, so that related guidance is placed together. Using retention priorities, it determines which groups remain in S_2 and which are externalized to R under the main-skill budget B_{skill}. These retention priorities govern placement only and are distinct from the importance scores used for reordering in Section 3.4. Detailed group construction and allocation rules are provided in Appendix A.2.

Hierarchy-preserving tree derivation. Externalized groups retain their relative Markdown hierarchy in the reference tree R, while retained content preserves its hierarchy in the main-skill tree S_2. At the corresponding location in S_2, we keep a compact reference entry linking to the externalized content. Thus,

$$(S_2, R) = \text{Derive}(S_1), \qquad L(S_2) \le B_{\text{skill}}, \tag{2}$$

where $L(\cdot)$ denotes the rendered Markdown length.

With the skill-reference allocation fixed, we next optimize the presentation order of the retained main-skill content.

3.4 Importance-Aware Tree Reordering

Although Tree Derivation determines which guidance remains persistently available, the Markdown tree is ultimately serialized into a linear context, making the order of sibling subtrees consequential.

Prior work has shown that language models can be sensitive to information position in long contexts (Liu et al., 2024). We therefore prioritize high-importance subtrees while preserving structural and workflow dependencies.

Node importance and subtree priority. For each node v in S_2, we estimate importance from two signals: trajectory-level support $c(v)$, the number of distinct training cases supporting retained guidance in the node, and reference connectivity $r(v)$, the number of distinct reference paths directly cited by the node. We combine log-normalized $c(v)$ and $r(v)$ with equal weights to obtain a node importance score $I(v)$, omitting unavailable signals rather than treating them as zero-valued evidence. The exact normalization and attribution procedure are provided in Appendix A.3.

Because presentation operates on heading-rooted subtrees rather than isolated nodes, we define subtree priority as

$$P(v) = \max_{u \in V(T_v)} I(u). \tag{3}$$

Using the maximum allows a strongly supported descendant to raise the priority of its enclosing subtree without favoring larger subtrees.

We reorder eligible sibling subtrees according to $P(v)$, placing higher-priority content earlier while respecting structural and workflow constraints. The resulting reordered main-skill tree S_3 completes training-time construction. Detailed ordering rules are provided in Appendix A.3.

3.5 Inference-Time Tree Routing

At inference time, the constructed pair (S_3, R) remains fixed. The main-skill tree S_3 is persistently available to the execution agent, while R provides task-conditioned supplementary guidance selected before the first model call under the reference budget B_{ref}.

Hierarchy-aware routing. To preserve the structural context established during skill construction, we parse the reference tree R according to its Markdown hierarchy and use heading-rooted subtrees as retrieval units, retaining each candidate's heading path and content. Candidates are ranked using deterministic weighted term matching between the task query and their structural and textual fields. When an operation required by the task is not represented in the preliminary results, operation-intent supplementation adds a compatible candidate when available. We then select subtrees subject to routing constraints and B_{ref}:

$$Z_q = \mathrm{Select}\big(q, Z(R); \mathcal{C}_{\mathrm{route}}, B_{\mathrm{ref}}\big), \tag{4}$$

where $Z(R)$ denotes the candidate subtrees and $\mathcal{C}_{\mathrm{route}}$ collects the routing constraints. The selected subtrees are serialized with their heading context and injected alongside S_3 before the first model call. Detailed query construction, matching, supplementation, and selection rules are provided in Appendix A.4. Algorithm 1 in Appendix summarizes the complete Skill Tree pipeline.

4 Experiments

4.1 Experiment Setup

Tasks and Evaluation Metrics. Following Trace2Skill (Ni et al., 2026), we use SpreadsheetBench-Verified, a 400-task subset of SpreadsheetBench (Ma et al., 2024), as our primary benchmark. The first 200 tasks are used for skill evolution and the remaining 200 for held-out evaluation. We report *Vrf* on this split, together with *Soft* (sub-problem pass rate) and *Hard* (all sub-problems pass) on the full SpreadsheetBench. We further evaluate frozen spreadsheet skills on WikiTableQuestions (Pasupat & Liang, 2015) without additional evolution and report answer accuracy. For multimodal evaluation, we follow the Trace2Skill/SkillCAT DocVQA protocol (Ni et al., 2026; Chen et al., 2026): 2,700 of the 5,349 validation examples are used for error-driven skill evolution and the remaining 2,649 for evaluation. We report ANLS and Accuracy, with ANLS $\geq$ 0.5 counted as correct. Additional dataset and preprocessing details are provided in Appendix A.7 .

Table 1: **Main results on skill deepening and skill creation.** Within each block, the same model serves as both the skill author and the skill user. Vrf reports held-out SpreadsheetBench-Verified accuracy; Soft and Hard are evaluated on full SpreadsheetBench; WikiTQ measures OOD generalization. Higher is better for all metrics and results of our method are average results of three seeds. Avg is the arithmetic mean of the available metrics. ReasoningBank has no results on WikiTQ because it is a memory retrieval method that cannot generalize across semantically distant queries. ReasoningBank is included only for Deepening, as it retrieves episodic memories at inference time rather than creating a standalone skill from a weak initial seed, as required by the Creation setting.

Setting	Method	SpreadsheetBench			OOD	Overall
		Vrf↑	Soft↑	Hard↑	WikiTQ↑	Avg↑
Skill Author = Skill User: Qwen3.5-35B-A3B						
Deepening	Human-Written	9.67	13.03	3.37	9.02	8.77
	ReasoningBank	20.50	17.30	4.97	N/A	14.26
	Trace2Skill	29.67	18.80	5.73	51.22	26.36
	SkillCAT	55.50	39.02	20.61	81.55	49.17
	SkillOPT	58.00	36.46	17.38	69.52	45.34
	SkillGrad	61.50	37.26	19.45	73.84	48.01
	Ours	**64.33**	**39.11**	**32.68**	**82.90**	**54.76**
Creation	Parametric	20.17	13.70	3.87	20.14	14.47
	Trace2Skill	20.00	18.10	5.13	38.14	20.34
	SkillCAT	54.50	39.35	19.41	72.82	46.52
	SkillOPT	53.50	36.26	18.72	72.03	45.13
	SkillGrad	52.50	36.73	18.64	73.14	45.25
	Ours	**62.67**	**40.46**	**33.88**	**83.20**	**55.05**
Skill Author = Skill User: Qwen3.5-122B-A10B						
Deepening	Human-Written	48.33	36.30	17.03	74.68	44.09
	ReasoningBank	56.00	40.10	21.30	N/A	39.13
	Trace2Skill	69.83	47.17	29.53	79.24	56.44
	SkillCAT	69.50	47.42	31.91	84.47	58.33
	SkillOPT	70.33	48.37	32.41	83.62	58.68
	SkillGrad	71.00	48.25	33.08	84.77	59.28
	Ours	**75.00**	**49.36**	**41.84**	**85.73**	**62.98**
Creation	Parametric	26.17	36.60	17.50	23.73	26.00
	Trace2Skill	26.33	27.37	16.10	56.05	31.46
	SkillCAT	64.50	40.73	24.45	84.47	53.54
	SkillOPT	60.33	41.24	25.73	83.92	52.81
	SkillGrad	59.67	40.93	26.49	86.74	53.46
	Ours	**73.00**	**48.61**	**40.90**	**87.03**	**62.39**

Skill Evolution Settings. We evaluate both **Deepening** and **Creation** following Trace2Skill. Deepening starts from the same human-written skill, whereas Creation starts from the same trajectory-free parametric skill; these unevolved skills serve as the corresponding initial baselines. Skill Tree, Trace2Skill, SkillGrad, and SkillCAT use the same fixed trajectory evidence. SkillOpt retains its native iterative protocol, which recollects trajectories after skill updates. Further protocol details are provided in Appendix A.8.

Baselines and Models. We compare against **Trace2Skill** (Ni et al., 2026), **SkillCAT** (Chen et al., 2026), **SkillOpt** (Yang et al., 2026b), **SkillGrad** (Wang et al., 2026), and **ReasoningBank** (Ouyang et al., 2026b); the latter is a retrieval-based reasoning-memory baseline rather than an evolved skill. For same-model evaluation, Qwen3.5-35B-A3B and Qwen3.5-122B-A10B each serve as both skill author and skill user. For cross-family evaluation, frozen Qwen-authored skills are used by Gemma-4-31B-it without re-evolution.

Table 2: **Cross-family model generalization.** The skill user is gemma-4-31B-it under the Deepening setting. Skills are authored by the model indicated in each block. Higher is better for all metrics and results of our method are average results of three seeds.

Method	SpreadsheetBench			OOD	Overall
	Vrf↑	Soft↑	Hard↑	WikiTQ↑	Avg↑
Skill Author: Qwen3.5-35B-A3B → Skill User: gemma-4-31B-it					
No-Skill	46.00	36.84	25.22	79.61	46.92
Human-Written	39.00	23.76	16.89	78.64	39.57
SkillCAT	61.00	42.47	29.82	79.61	53.23
SkillOPT	62.00	42.54	29.71	80.15	53.60
SkillGrad	62.33	42.87	30.04	80.81	54.01
Ours	**66.67**	**43.83**	**35.74**	**82.60**	**57.21**
Skill Author: Qwen3.5-122B-A10B → Skill User: gemma-4-31B-it					
No-Skill	46.00	36.84	25.22	79.61	46.92
Human-Written	39.00	23.76	16.89	78.64	39.57
SkillCAT	70.50	45.18	31.14	76.70	55.88
SkillOPT	71.33	44.85	33.77	77.41	56.84
SkillGrad	70.33	43.97	32.24	79.05	56.40
Ours	**74.67**	**46.38**	**41.57**	**87.18**	**62.45**

Implementation Details. SpreadsheetBench and WikiTQ agents use a ReAct-style harness (Yao et al., 2023) with filesystem and spreadsheet tools. Semantic deduplication uses the corresponding skill-author model as the NLI judge with threshold 0.75. We set $B_{\text{skill}} = 1500$ Markdown lines, $w_c = w_r = 0.5$, $B_{\text{ref}} = 40{,}000$ characters, and $K_{\text{route}} = 6$. DocVQA uses the multimodal harness with otherwise identical Skill Tree hyperparameters. More details are provided in Appendix A.9.

4.2 Main Results on Skill Deepening and Creation

Table 1 shows that Skill Tree improves both **Deepening** and **Creation** across the two Qwen3.5 models. The largest gains appear on the strict *Hard* metric, which requires all test cases associated with an instruction to pass. With Qwen3.5-35B-A3B, Skill Tree improves Hard accuracy by 12.07 points in Deepening and by 14.47 points in Creation. With Qwen3.5-122B-A10B, the corresponding gains are 8.76 points and 14.41 points. More concretely, in the 35B Deepening setting, Skill Tree turns 83 of the 912 SpreadsheetBench tasks that Trace2Skill solves only partially (i.e., passes Soft but not Hard) into complete successes. This shift from partial to complete success suggests that Skill Tree improves the *robustness of task execution across input variations*, consistent with its emphasis on preserving detailed procedural guidance and making it accessible in the appropriate task context.

The same trend extends to held-out SpreadsheetBench-Verified and OOD WikiTQ. Skill Tree achieves the best Vrf score in all four model/setting combinations, and its frozen spreadsheet skills remain effective on WikiTQ, reaching $82.90/83.20$ for Qwen3.5-35B-A3B and $85.73/87.03$ for Qwen3.5-122B-A10B under Deepening/Creation.

4.3 Cross-Family Model Generalization

Table 2 shows that Skill Tree transfers effectively across model families: Qwen-authored skills consistently improve Gemma performance without re-evolution, with particularly strong gains on the strict Hard metric. The advantage further increases with the stronger 122B skill author, while WikiTQ performance remains high, indicating robust cross-model transfer.

4.4 Multimodal Evaluation on DocVQA

Figure 3 shows that Skill Tree also generalizes to multimodal skill evolution on DocVQA. In this evaluation, we use the same model as skill author and user. With Qwen3.5-35B-A3B, our method reaches 0.931 ANLS and 94.3% accuracy, outperforming the strongest baseline by 0.009 ANLS and 0.6 accuracy points. With Qwen3.5-122B-A10B, the gains remain consistent, reaching 0.936



Figure 3: Results on DocVQA. We report ANLS and accuracy for skills evolved independently in the multimodal DocVQA setting.

ANLS and 95.0% accuracy, compared with 0.927 and 94.1% for the strongest baseline. These results indicate that the proposed skill-construction framework is not specific to spreadsheet agents and remains effective when extended to multimodal document understanding.

Table 3: Ablation results on the held-out SpreadsheetBench-Verified split. Component ablations replace one major stage while preserving the remaining pipeline; structural ablations retain the corresponding functionality but remove its use of Markdown hierarchy.

Variant	Intervention	Vrf↑	Δ
Component ablations			
Full Skill Tree	–	**64.0**	0.0
w/o Tree Merge	Trace2Skill merge	55.0	−9.0
w/o Semantic Deduplication	Retain all compatible edits	57.5	−6.5
w/o Tree Shaping Priority	Source-order allocation	52.5	−11.5
w/o Tree Reordering	Original subtree order	51.0	−13.0
w/o Inference-Time Routing	No automatic pre-routing	57.5	−6.5
Structural ablations			
w/o Merge Scope	Scope-agnostic consolidation	54.5	−9.5
w/o Hierarchical Shaping Units	Paragraph-wise allocation	51.5	−12.5
w/o Hierarchical Routing Units	Paragraph-chunk routing	52.0	−12.0

4.5 Ablation Analysis

Effect of Structure-Constrained Consolidation. We first examine the design of the consolidation stage. *w/o Tree Merge* replaces our structure-constrained filtering with the rewriting-based Trace2Skill merge, while *w/o Semantic Deduplication* retains the structural grouping but keeps all compatible edits without removing semantic redundancy. Both variants reduce performance. *w/o Merge Scope* retains semantic deduplication but applies it without structural scope constraints, which also leads to a substantial drop. Together, these results show that effective consolidation requires both removing redundant guidance and restricting that removal to structurally compatible contexts.

Effect of Skill Organization. The next variants examine how retained knowledge is organized after consolidation. *w/o Tree Shaping Priority* replaces priority-aware allocation with source-order placement, while *w/o Tree Reordering* keeps the original subtree order; both substantially reduce performance. Meanwhile, *w/o Hierarchical Shaping Units*, which replaces structurally coherent allocation groups with paragraph-level units, causes a similarly large degradation. These results indicate that effective skill organization depends on both *where* knowledge is placed and *how* structurally related knowledge is kept and presented together.

Grouping	# Groups	SC ↑
Trace2Skill ($B = 8$)	25	0.5457
Trace2Skill ($B = 16$)	13	0.5422
Trace2Skill ($B = 32$)	6	0.5431
Skill Tree	**33**	**0.6997**

Table 4: Semantic cohesion of merge groups under different grouping strategies. Higher is better.

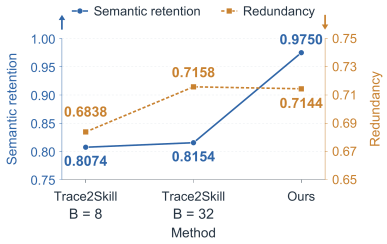

Figure 4: Semantic retention and redundancy after consolidation. At comparable within-scope redundancy, Skill Tree preserves substantially more trajectory-derived information.

Effect of Hierarchy-Aware Reference Access. The final variants examine access to supplementary references. *w/o Inference-Time Routing* disables automatic pre-routing while retaining later tool-based reference access, reducing Vrf by 6.5 points. In contrast, *w/o Hierarchical Routing Units* keeps automatic routing but replaces heading-rooted subtrees with paragraph chunks, leading to a substantially larger 12.0-point drop. This contrast shows that the benefit of routing depends not only on exposing additional reference content, but also on retrieving it in units that preserve its structural context.

4.6 Mechanistic Analysis

Patch-Group Cohesion. We analyze whether Skill Tree forms more semantically coherent merge groups than the batch-based grouping used by Trace2Skill. Using the same original patch set, we compare Tree Merge with Trace2Skill under different batch sizes B and report the macro-averaged within-group semantic cohesion (*SC*) over non-singleton groups. Table 4 shows that changing the Trace2Skill batch size leaves semantic cohesion nearly unchanged at around 0.54, whereas Skill Tree reaches 0.6997. This supports the intended mechanism of Tree Merge: structural grouping places semantically related edits into substantially more coherent consolidation groups.

Semantic Retention and Redundancy. The *w/o Semantic Deduplication* ablation first establishes that within-scope redundancy has a real downstream cost: retaining redundant compatible edits reduces Vrf from 64.0 to 57.5. However, the *w/o Merge Scope* result shows that indiscriminate deduplication is also harmful, indicating that redundancy should be removed selectively within compatible structural contexts. Against this premise, Figure 4 shows that, at a comparable level of within-scope redundancy, Skill Tree preserves substantially more trajectory-derived information than Trace2Skill. Thus, its advantage lies not in minimizing redundancy alone, but in preserving complementary guidance while controlling repetition. Metric definitions are provided in Appendix A.11.

More Results. We provide the sensitivity analysis of the skill budget in Appendix A.10, indicating that $B_{\text{skill}} = 1500$ lines balances persistent knowledge coverage against attention dilution from excessive context.

5 Conclusion

We introduced **Skill Tree**, a framework that uses Markdown hierarchy as a shared structure for skill consolidation, organization, and access. By preserving useful guidance, removing redundancy within compatible structural contexts, and organizing accumulated knowledge for efficient reuse, Skill Tree consistently improves skill evolution across multiple settings. The resulting gains extend beyond in-domain evaluation to OOD generalization, cross-model transfer, and multimodal tasks, while ablation and mechanistic analyses further support the importance of structure-aware consolidation and access. These results show that document structure can serve not merely as presentation metadata, but as an effective operational scaffold for constructing and evolving reusable agent skills.

AI USE STATEMENT

In this work, we used generative AI tools to improve the readability, grammar, and wording of the manuscript, assist with literature search and reference formatting, and improve the organization and structure of the paper. We did not use generative AI tools for tasks requiring mandatory disclosure, such as developing the methodology, designing experiments, or interpreting experimental results. All AI-assisted content was reviewed and verified by the authors. The authors take full responsibility for the final content of this work.

ETHICS STATEMENT

This work studies the construction and reuse of agent skills from execution trajectories and does not involve human subjects or the collection of private or sensitive personal data. Our experiments use existing benchmarks and evaluate agents in controlled task environments. As with other methods that improve autonomous agent capabilities, the proposed framework could potentially be applied in settings beyond those evaluated here. We therefore emphasize that deployment in high-stakes or safety-sensitive environments should include appropriate task-specific validation and safeguards.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the Skill Tree pipeline and its individual components in the main paper and appendix. Appendix A.1–A.4 specifies Tree Merge, Tree Shaping, Tree Reordering, and inference-time routing, while Appendix A.5 summarizes the complete procedure in pseudocode. Appendix A.7 describes dataset construction and evaluation protocols, Appendix A.8 details baseline-specific experimental procedures, and Appendix A.9 reports the main implementation settings and hyperparameters. Across comparable methods, we match the initial skills, trajectory evidence, data splits, target models, and evaluation protocols whenever their native procedures permit.

REFERENCES

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

Anthropic. Building effective agents. https://www.anthropic.com/engineering/building-effective-agents, December 2024. Anthropic Engineering, published December 19, 2024.

Anthropic. Equipping agents for the real world with agent skills. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills, October 2025. Anthropic Engineering, published October 16, 2025.

Kunfeng Chen, Qihuang Zhong, Juhua Liu, and Bo Du. Skillcat: Contrastive, assessment-augmented and topology-awareskill self-evolution for llm agents, 2026. URL https://arxiv.org/abs/2606.13317.

Yudong Gao, Zongjie Li, Yuanyuan Yuan, Zimo Ji, Pingchuan Ma, and Shuai Wang. Skillreducer: Optimizing llm agent skills for token efficiency, 2026. URL https://arxiv.org/abs/2603.29919.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents, 2026. URL https://arxiv.org/abs/2603.12056.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation, 2024. URL https://arxiv.org/abs/2406.14991.

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, 2026. URL https://arxiv.org/abs/2603.25158.

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

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL https://openreview.net/forum?id=jL7fwchScm.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables, 2015. URL https://arxiv.org/abs/1508.00305.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. Raptor: Recursive abstractive processing for tree-organized retrieval, 2024. URL https://arxiv.org/abs/2401.18059.

Chihiro Taguchi, Seiji Maekawa, and Nikita Bhutani. Efficient context selection for long-context QA: No tuning, no iteration, just adaptive-k. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20105–20130. Association for Computational Linguistics, 2025.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

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

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

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills, 2026b. URL https://arxiv.org/abs/2605.23904.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=eC4ygDs02R.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents, 2026b. URL https://arxiv.org/abs/2604.17308.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents, 2026. URL https://arxiv.org/abs/2603.18743.

A APPENDIX

APPENDIX CONTENTS

A.1 Markdown Tree Construction Details 13
A.2 Tree Derivation Planner Details 14
A.3 Importance Estimation and Structure-Constrained Reordering Details 15
A.4 Inference-Time Routing Details 16
A.5 Overall Algorithm 17
A.6 Semantic Cohesion Analysis 17
A.7 Detailed Experimental Setup 17
A.8 Baseline-Specific Protocols 18
A.9 Additional Implementation Details 19
A.10 Sensitivity Analysis of Skill Budget 19
A.11 Semantic Retention and Redundancy Metrics 19
A.12 Spreadsheet Question 2786 Example and Injected References 20

A.1 MARKDOWN TREE CONSTRUCTION DETAILS

This section provides implementation details for the training-time Markdown tree construction described in Section 3.2, including patch localization, grouping, semantic deduplication, and provenance tracking.

Patch representation and target localization. Each candidate patch $p \in \mathcal{P}$ contains an edit operation, the proposed edit content, and information identifying where the edit should be applied in the base skill. We parse the base skill into an initial Markdown heading tree

$$T^{(0)} = \mathrm{Parse}(S_0), \tag{5}$$

and resolve each patch to a target node

$$t(p) \in V\left(T^{(0)}\right). \tag{6}$$

Target resolution uses the Markdown heading structure together with the target specification and available text anchors. Because headings may repeat, a resolved node is identified by its document, complete heading path, and occurrence within that path:

$$\mathrm{id}(v) = \big(d, \pi(v), k(v)\big), \tag{7}$$

where d is the source document, $\pi(v)$ is the complete heading path, and $k(v)$ distinguishes repeated occurrences of the same path. This structural identity avoids relying on absolute line positions, which may change as edits are applied.

Target-node-based grouping. After localization, candidate edits are partitioned into merge groups according to their resolved application locations. Edits assigned to the same group must target the same Markdown node and be mutually compatible with respect to their edit operations and text anchors:

$$p_i, p_j \in G_m \implies t(p_i) = t(p_j) \wedge C_{\mathrm{op}}(p_i, p_j) \wedge C_{\mathrm{anchor}}(p_i, p_j), \tag{8}$$

where C_{op} and C_{anchor} denote operation and anchor compatibility, respectively. The resulting groups are

$$\mathcal{G} = \{G_1, \ldots, G_M\}. \tag{9}$$

Grouping therefore follows the location at which an edit will affect the Markdown tree, rather than the trajectory from which the edit was generated. Edits targeting different structural contexts are kept separate even when their surface content is similar.

Group-local semantic deduplication. Different trajectories may produce overlapping guidance for the same target location. For each group G_m, we compare the candidate edit contents using a natural language inference model. The comparison distinguishes semantic relations including equivalence and directional entailment. If the information expressed by one edit is fully covered by another compatible edit, the redundant candidate can be removed; candidates containing additional or complementary information are retained. We denote the resulting group by

$$\widetilde{G}_m = \mathrm{Dedup}_{\mathrm{NLI}}(G_m). \tag{10}$$

Importantly, semantic deduplication is restricted to edits within the same merge group. Thus, semantic similarity alone cannot remove guidance that belongs to a different target node or structural context.

Sequential application and provenance. The deduplicated groups are applied sequentially to the tree initialized from S_0. After processing group $\widetilde{G}_m$, the current Markdown tree is updated and passed to the next application step. The final tree is the unshaped skill tree S_1 used by the derivation stage in Section 3.3.

Alongside the edit content, we retain the identifiers of the training cases from which the corresponding patches were generated. These provenance records are propagated with retained patch content through tree construction and subsequent tree derivation. They provide the traceable trajectory-level support signal used for node-importance estimation in Section 3.4. Importantly, this provenance records observed support from distinct training cases; it is not treated as an estimate of semantic coverage or downstream task frequency.

A.2 Tree Derivation Planner Details

This section provides the implementation details of the budget-aware planner and hierarchy-preserving derivation described in Section 3.3.

Allocation-group construction. The planner operates on allocation groups rather than individual paragraphs. Eligible content in S_1 is grouped according to its target section and a normalized topic or heading title, so that structurally related updates are placed together whenever possible. We denote the resulting groups by

$$\mathcal{U}(S_1) = \{g_1, \ldots, g_K\}. \tag{11}$$

When a suitable content heading is available, it is used to identify the topic of the group; otherwise, the corresponding target-section heading provides the grouping key. This construction prevents closely related guidance from being split independently between the main skill document and the reference tree.

Retention priority. Each allocation group receives a heuristic retention priority that determines how strongly it should be preferred for persistent exposure in the main skill document. The priority combines the amount of source evidence associated with the group, measured from its contributing source edits, with predefined domain priorities that favor broadly useful content such as common procedures and output requirements. These retention priorities are used only for the placement decision in Section 3.3; they are distinct from the trajectory-level node importance used for reordering in Section 3.4.

Progressive externalization and restoration. The planner begins from the experience-augmented tree S_1 and progressively externalizes eligible allocation groups in increasing order of retention priority. After each allocation change, the candidate main skill tree is materialized and rendered, and its Markdown length $L(S_2)$ is recomputed. Externalization continues until the target budget is reached or no further eligible group can be externalized.

When multiple groups have the same retention priority, larger groups are considered earlier for externalization, since moving them provides a larger reduction in the main-document length. After the externalization pass, the planner considers restoring previously externalized groups in decreasing order of retention priority. A restoration is accepted only when the resulting rendered main skill remains within the target budget. Under equal retention priority, smaller groups are considered first

during restoration, allowing the remaining budget to accommodate useful content more efficiently. The process continues until no additional eligible group can be restored without violating the budget.

Hierarchy-preserving materialization. Once the placement decisions Π are fixed, the planner materializes the main skill-document tree S_2 and the reference tree R. For a group assigned to skill, its content remains at the corresponding location in the Markdown hierarchy of S_2. For a group assigned to $\mathrm{reference}$, the externalized content is transferred to R while preserving its relative heading hierarchy. The corresponding location in S_2 retains a compact reference entry pointing to the externalized content. Thus, a reference entry acts as the persistent connection between the section role represented in the main skill document and the more detailed guidance stored in R.

The relative hierarchy retained in R also defines the subtree boundaries subsequently used by inference-time routing. Externalization therefore changes the exposure location of the content without flattening its Markdown structure or discarding the relationship between the main skill and its supplementary details.

Support propagation through externalization. Source provenance is retained across the derivation process so that later importance estimation can still trace retained skill content to its contributing training cases. When a group is externalized, the traceable support associated with that group is attributed to the retained reference entry in the corresponding node of S_2. This attribution represents support for the main-skill entry point to the externalized material; it is not propagated indiscriminately to every descendant node in the reference tree. This distinction is used when computing the trajectory-level support signal described in Section 3.4.

A.3 Importance Estimation and Structure-Constrained Reordering Details

This section provides the detailed importance estimation and constrained reordering procedure used in Section 3.4.

Trajectory-level support attribution. For a node v in the main skill-document tree, let $c(v)$ denote the number of distinct training cases that can be traced to retained patch content in its direct body. The source training-case identifiers associated with candidate patches are propagated through Markdown tree construction and tree derivation, allowing retained content to remain linked to the trajectories from which it was distilled. Thus, $c(v)$ measures traceable empirical support across distinct training cases rather than the raw number of candidate edits.

Let $r(v)$ denote the number of distinct reference paths directly cited in the direct body of v. This quantity captures the connectivity of the node to supplementary guidance externalized in the reference tree.

Node-importance normalization. Because trajectory-support and reference counts may have different numerical scales, we first normalize them using logarithmic transformations:

$$f_c(v) = \frac{\log(1 + c(v))}{\log(1 + \max\{1, C_{\max}\})}, \qquad f_r(v) = \frac{\log(1 + r(v))}{\log(1 + \max\{1, R_{\max}\})}, \tag{12}$$

where $C_{\max}$ and $R_{\max}$ are the maximum known trajectory-support and reference counts, respectively, among the corresponding nodes in the document. The logarithmic transformation compresses large count differences before they contribute to the node score.

Let

$$A(v) \subseteq \{c, r\} \tag{13}$$

denote the set of signals available for node v. We compute

$$I(v) = \frac{\sum_{k \in A(v)} w_k f_k(v)}{\sum_{k \in A(v)} w_k}, \qquad w_c = w_r = \frac{1}{2}. \tag{14}$$

Unavailable signals are omitted rather than assigned a value of zero, and the remaining weights are renormalized accordingly. If neither signal is available, $I(v)$ remains undefined.

Subtree priority. For a subtree rooted at v, its priority is the maximum defined importance score among its nodes:

$$P(v) = \max_{u \in V(T_v)} I(u). \tag{15}$$

If no node in T_v has a defined importance score, its subtree priority remains undefined. Using the maximum allows a high-support descendant to increase the priority of its containing subtree without accumulating scores as a function of subtree size. It also avoids diluting localized high-support content through averaging.

Protected blocks and admissible regions. Reordering is performed only among eligible sibling subtrees. Fixed-position content remains unchanged, while protected dependency groups are treated as indivisible blocks whose internal order is preserved. Parent–child relationships are never modified. Procedural dependencies, including ordered steps and other protected relations, restrict which sibling permutations are admissible. Where configured, predefined workflow-stage relations introduce additional precedence constraints between eligible blocks.

Subtrees that are not eligible for reordering, including those without the required traceable support, retain their original positions. Fixed positions also delimit the regions within which movable blocks may be reordered.

Stable priority-aware ordering. Within each movable sibling region, the structural and workflow constraints induce a partial order over the eligible blocks. We perform a stable topological ordering over this partial order. At each step, among blocks whose prerequisites have already been satisfied, the block with the highest available subtree priority is selected. When priorities are equal, the original document order is preserved.

The ordering procedure changes only eligible sibling permutations. Node bodies, code blocks, and parent–child relationships are left unchanged, so reordering does not alter the content allocation between the main skill-document tree and the reference tree.

A.4 Inference-Time Routing Details

This section provides the implementation details of the deterministic inference-time routing procedure described in Section 3.5.

Query construction. Before the first model call, we construct a retrieval query from the current task instruction and the task information available to the execution agent. The query is normalized using the same text-processing rules as the candidate fields used for retrieval. No additional model call is introduced during this process.

Hierarchy-derived candidates. Each reference document is parsed into its Markdown heading tree. Heading-rooted subtrees form the candidate retrieval units, preserving both the content of the corresponding section and its structural context. For a candidate subtree z, we retain its reference-document identity, heading path, and Markdown body. These fields allow routing to match the task against both the semantic content of the candidate and the section context in which that content is organized.

Weighted deterministic matching. Each candidate subtree is assigned a deterministic relevance score based on lexical overlap between the task query and multiple candidate fields. The matching procedure considers document-level information, heading information, ancestor context, and the textual content of the candidate. Different fields receive different fixed weights so that matches to structural information, such as section headings, can contribute differently from incidental matches in longer body text. The resulting relevance scores determine the preliminary candidate ranking.

Operation-intent supplementation. Weighted term matching may fail to retrieve guidance for an operation explicitly requested by the task when the task and reference content use different surface forms. We therefore detect predefined operation intents from the task query and check whether the preliminary selection contains a candidate associated with each detected intent. When an intent is not represented and a compatible candidate is available, the router supplements the preliminary

selection subject to the remaining retrieval constraints. This step is deterministic and does not require an additional LLM call.

Budget-constrained selection. Candidate admission is subject to fixed limits on the number of reference files, the total number of selected subtrees, and the total amount of injected reference content. Starting from the relevance ranking and operation-intent supplementation, candidates are retained only while these limits and the character budget B_{ref} remain satisfied. The resulting set $\mathcal{Z}_q$ therefore contains a bounded collection of task-relevant Markdown subtrees.

Serialization and prompt injection. After selection, the router materializes the actual Markdown content of each candidate rather than exposing only its retrieval metadata. The associated heading information is retained during serialization so that the selected guidance remains interpretable in its original structural context. The serialized reference content is inserted into the task prompt before the first model call, while the serialized main skill document derived from S_3 remains persistently available to the execution agent. The model then begins reasoning and execution using this combined context.

A.5 Overall Algorithm

Algorithm 1 summarizes the complete Skill Tree pipeline, including training-time skill construction and inference-time reference routing. The training stage successively performs structure-constrained consolidation, budgeted tree shaping, and importance-aware tree reordering, producing the final main-skill tree and reference tree. At inference time, the constructed skill remains fixed and only task-relevant reference subtrees are selected and injected into the execution context.

A.6 Semantic Cohesion Analysis

We compute semantic cohesion using the same original candidate patches for all grouping strategies. Each edit body is treated as one comparison unit. To prevent the grouping metadata itself from influencing the measurement, we exclude filenames, target heading paths, patch identifiers, and Markdown heading lines from the encoded text. Explanations, procedural steps, formulas, and code are retained without additional summarization or rewriting.

Each edit body is encoded once using a fixed **Qwen3-Embedding-0.6B** model under the same encoding configuration. The resulting embeddings and pairwise cosine-similarity matrix are shared across all grouping strategies. For each non-singleton group g, we compute its semantic cohesion as the mean pairwise cosine similarity among the edits in that group:

$$\mathrm{SC}(g) = \frac{2}{|g|(|g|-1)} \sum_{i<j,\ i,j\in g} \cos(\mathbf{e}_i, \mathbf{e}_j).$$

We then macro-average over all non-singleton groups,

$$\mathrm{SC} = \frac{1}{|\mathcal{G}_{>1}|} \sum_{g\in\mathcal{G}_{>1}} \mathrm{SC}(g),$$

so that each group contributes equally regardless of size. Trace2Skill groups are constructed using its original batch-based merging procedure with batch sizes $B \in \{8, 16, 32\}$, while Skill Tree uses the structure-constrained grouping defined in Section **??**.

A.7 Detailed Experimental Setup

For WikiTableQuestions, we convert the table inputs and expected outputs into spreadsheet-compatible format and evaluate the spreadsheet skills without any further skill evolution. The resulting score measures OOD reuse of the frozen spreadsheet skill.

For DocVQA, we follow the setup used by the early Trace2Skill evaluation and subsequently adopted by SkillCAT. We use the official validation split containing 5,349 question–image pairs. The first 2,700 examples form the evolution set, from which error trajectories are collected for skill

Algorithm 1 SKILL TREE: Training-Time Construction and Inference-Time Routing

Require: Base skill S_0; training trajectories $\mathcal{T}_{S_0} = \{\tau_i\}_{i=1}^{N}$; main-skill budget B_{skill}; reference budget B_{ref}
Ensure: Constructed skill (S_3, R) and, for a task q, execution context P_q

Training-time construction

$\mathcal{P} \leftarrow \emptyset$
for $\tau_i \in \mathcal{T}_{S_0}$ **do**
  $\mathcal{P} \leftarrow \mathcal{P} \cup F_{\text{patch}}(\tau_i, S_0)$
end for
$T \leftarrow \text{PARSEMARKDOWN}(S_0)$
for $p \in \mathcal{P}$ **do**
  $t(p) \leftarrow \text{RESOLVETARGET}(p, T)$
end for
$\mathcal{G} \leftarrow \text{GROUPEDITS}(\mathcal{P}, \{t(p)\}_{p\in\mathcal{P}})$
for $G \in \mathcal{G}$ **do**
  $\widetilde{G} \leftarrow \text{DEDUPNLI}(G)$
  $T \leftarrow \text{APPLY}(T, \widetilde{G})$
end for
$S_1 \leftarrow T$
$\mathcal{U} \leftarrow \text{BUILDALLOCATIONGROUPS}(S_1)$
$\Pi \leftarrow \text{INITIALIZEPLACEMENT}(\mathcal{U}, \text{skill})$
while $L(\text{MATERIALIZESKILL}(S_1, \Pi)) > B_{\text{skill}}$ **and** an eligible group remains **do**
  $g \leftarrow \text{LOWESTRETENTIONGROUP}(\mathcal{U}, \Pi)$
  $\Pi(g) \leftarrow \text{reference}$
end while
for g in decreasing retention priority among externalized groups **do**
  if $L(\text{MATERIALIZESKILL}(S_1, \Pi[g \leftarrow \text{skill}])) \le B_{\text{skill}}$ **then**
    $\Pi(g) \leftarrow \text{skill}$
  end if
end for
$(S_2, R) \leftarrow \text{DERIVE}(S_1, \Pi)$
for $v \in V(S_2)$ **do**
  $c(v) \leftarrow \text{TRAJECTORYSUPPORT}(v)$
  $r(v) \leftarrow \text{REFERENCECONNECTIVITY}(v)$
  $I(v) \leftarrow \text{NODEIMPORTANCE}(c(v), r(v))$
end for
for $v \in V(S_2)$ **do**
  $P(v) \leftarrow \max_{u \in V(T_v),\, I(u) \text{ defined}} I(u)$
end for
$S_3 \leftarrow \text{CONSTRAINEDREORDER}(S_2, P, \mathcal{C})$

Inference-time routing

$\mathcal{Z}(R) \leftarrow \text{HEADINGSUBTREES}(R)$
$\widehat{S}_3 \leftarrow \text{SERIALIZE}(S_3)$
for new task q **do**
  $Q_q \leftarrow \text{BUILDQUERY}(q)$
  $\rho(z) \leftarrow \text{ROUTESCORE}(Q_q, z)$ for each $z \in \mathcal{Z}(R)$
  $\mathcal{Z}'_q \leftarrow \text{INTENTSUPPLEMENT}(q, \mathcal{Z}(R), \rho)$
  $Z_q \leftarrow \text{SELECT}(\mathcal{Z}'_q, \rho, \mathcal{C}_{\text{route}}, B_{\text{ref}})$
  $\widehat{Z}_q \leftarrow \text{SERIALIZE}(Z_q)$
  $P_q \leftarrow \text{COMPOSE}(\widehat{S}_3, q, \widehat{Z}_q)$
  return P_q
end for

evolution, and the remaining 2,649 examples are held out for evaluation. We report Average Normalized Levenshtein Similarity (ANLS) and Accuracy, where a prediction is considered correct when its ANLS is at least 0.5.

A.8 BASELINE-SPECIFIC PROTOCOLS

Throughout the paper, **Trace2Skill** denotes its full evolution setting: starting from the corresponding base skill, both successful and failed trajectories are analyzed and used as evolution evidence. We therefore refer to this setting simply as Trace2Skill in all tables and discussions.

For Skill Tree, Trace2Skill, SkillGrad, and SkillCAT, we use the same initial skill, evolution-task inputs, fixed trajectory pool, train/test splits, target models, and evaluation protocol. SkillOpt differs because its native optimization procedure repeatedly executes the current skill on the training set after each update and therefore recollects trajectory evidence throughout optimization. We preserve this iterative rollout protocol while matching the initial skill, task splits, target model, and scorer.

ReasoningBank is evaluated in its memory-based setting rather than as a skill-evolution method. Lessons distilled from successful and failed trajectories are stored as external memories, and the top-1 relevant memory is retrieved at inference time using Qwen3-Embedding-8B. Accordingly, ReasoningBank does not produce a standalone evolved skill.

A.9 Additional Implementation Details

For Skill Tree, the corresponding skill-author model is used as the NLI judge during semantic deduplication, with a confidence threshold of 0.75. Tree Shaping uses a main-skill budget of $B_{\text{skill}} = 1500$ Markdown lines. Tree Reordering combines trajectory-level support and reference connectivity with equal weights, $w_c = w_r = 0.5$. Inference-time routing uses a reference-content budget of $B_{\text{ref}} = 40{,}000$ characters and selects at most $K_{\text{route}} = 6$ reference subtrees per task.

For DocVQA, we switch from the language-model-only spreadsheet harness to the multimodal agent harness and corresponding document-image tools. All Skill Tree hyperparameters above remain unchanged. Full prompts, allocation priorities, workflow constraints, routing-field weights, operation-intent rules, and other implementation constants are fixed across the corresponding experiments.

A.10 Sensitivity Analysis of Skill Budget

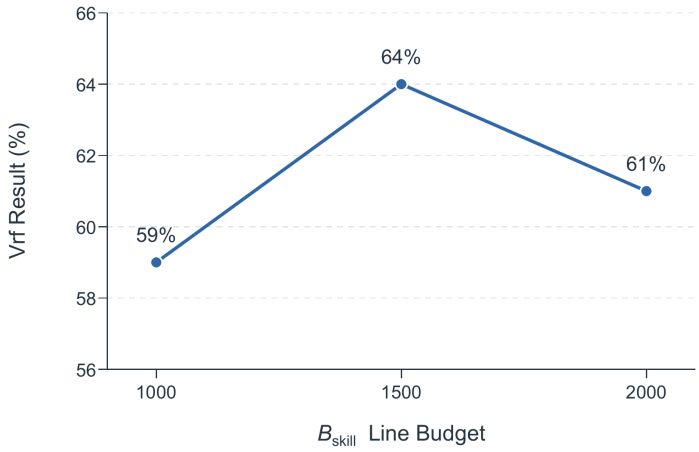

Figure 5: Sensitivity to the main-skill budget B_{skill} on Spreadsheet Bench Vrf. Performance peaks at an intermediate budget of 1500 lines.

Figure 5 shows a non-monotonic effect of the main-skill budget. Performance peaks at $B_{\text{skill}} = 1500$: a smaller budget leaves too much useful guidance in the reference tree, whereas a larger budget moves more reference content into the persistently visible main skill. This additional context can dilute the model's attention over the most relevant guidance, leading to the observed performance drop at $B_{\text{skill}} = 2000$.

A.11 Semantic Retention and Redundancy Metrics

We measure whether consolidation preserves trajectory-derived information while avoiding repeated guidance within the same structural context. Following the semantic-cohesion analysis, all text units are encoded using the same fixed Qwen3-Embedding-0.6B model, and cosine similarity is used throughout. Each trajectory-local patch p_i is associated with its resolved structural scope $g(p_i)$,

and each retained content unit s_j in the consolidated skill is associated with the corresponding Markdown scope $g(s_j)$. We restrict comparisons to matching scopes because semantically similar guidance may legitimately appear in different operational contexts.

Semantic retention. For each original patch, we identify the most semantically similar retained unit within the same structural scope. Semantic retention is defined as

$$\text{Retention} = \frac{1}{N}\sum_{i=1}^{N} \max_{\substack{j \\ g(s_j)=g(p_i)}} \cos(e(p_i), e(s_j))\,, \tag{16}$$

where $e(\cdot)$ denotes the embedding representation. If no retained unit exists in the corresponding scope, the retention score for that patch is set to zero. Higher values therefore indicate that more of the information expressed by the original trajectory-derived patches remains represented after consolidation.

Within-scope redundancy. Redundancy is measured only among retained units belonging to the same structural scope. For each retained unit s_i, we find its most similar distinct unit within that scope:

$$r_i = \max_{\substack{j \neq i \\ g(s_j)=g(s_i)}} \cos(e(s_i), e(s_j))\,. \tag{17}$$

We then average r_i over units whose structural scope contains at least two retained units:

$$\text{Redundancy} = \frac{1}{|\mathcal{S}_{>1}|}\sum_{s_i \in \mathcal{S}_{>1}} r_i, \tag{18}$$

where $\mathcal{S}_{>1}$ denotes retained units belonging to non-singleton scopes. Lower values indicate less repeated information within structurally compatible contexts. Importantly, semantic overlap across different scopes is not counted as redundancy, since similar guidance can serve distinct operations or workflow contexts.

A.12 Spreadsheet Question 2786 Example and Injected References

Spreadsheet Question (first paragraph)

In an Excel report concerning outstanding orders, I am looking for a formula that will allow me to assign available inventory stock to specific order lines on a sheet, prioritizing the allocation based on the Order No. Each Order No is associated with a customer's model, indicated by a suffix (such as A or B), and common items like 'Screw' can appear under different Order Nos. The allocation should reflect a quantity of 0 if there is no stock to fulfill an order line. Additionally, I have a complication where Order Line data is not present in Sheet 2, yet multiple lines may exist for the same Order ID on Sheet 1, each with varying requested delivery dates and the same item. How can I create such a formula?

Injected Selected Reference Subtrees form Reference File 1

When Recalculation May Not Be Available

If you cannot guarantee LibreOffice recalculation will run:

- Store computed values alongside formulas for critical cells
- For complex calculations where values must be immediately visible: compute in Python, then optionally add formula string as comment for documentation
- Always verify output environment supports formula evaluation before relying solely on formulas

Injected Selected Reference Subtrees form Reference File 2

Forward-Order Processing (COGS/FIFO)

For cost of goods sold, process layers OLDEST-FIRST:

```
# COGS: oldest purchases sold first
for po_column in ['PO-1', 'PO-2', 'PO-3', 'PO-4']:
    # Process from oldest purchase forward
    ...
```

Injected Selected Reference Subtrees form Reference File 3

✓ **CORRECT - Preserve Semantic Units**

```
# Good: Remove only leading/trailing whitespace
normalized_name = raw_name.strip()  # Keeps "SUNSHINE COAST" intact
```