Focused but Fragile: Tail Knowledge Collapse in Recursive Agent Skill Distillation

Anonymous ACL submission

Abstract

Agent skills package procedural knowledge into reusable artifacts that can improve LLM agents at inference time. Existing work shows that curated skills can materially raise average pass rates and that transferable skills can be distilled from execution traces. However, current evaluations emphasize average-case performance and systematically under-measure rare exceptions. This paper studies *tail-knowledge collapse*: under recursive rewriting of a robust skill artifact into shorter or more standardized variants, common-case utility may remain stable while rare exception handling degrades. To investigate this, we introduce the **TailSkills** benchmark, a tail-focused evaluation suite featuring 14 variant types across six categories, 208 oracle-verified exception-heavy task variants with deterministic verifiers, and an expert-guided agentic construction pipeline. We show that recursive distillation can preserve common-case utility while degrading tail-case performance, and that this degradation is accompanied by the preferential removal of short, local tail-handling instructions relative to regular workflow content.

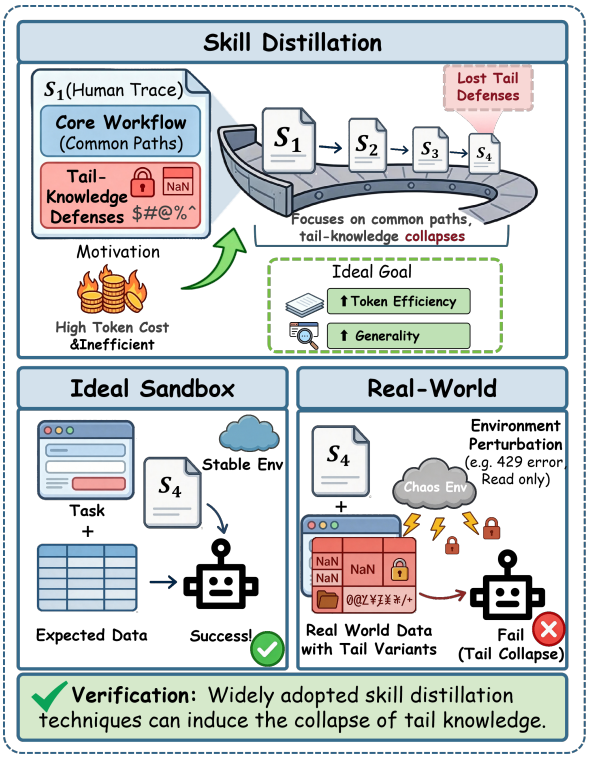

Figure 1: Recursive distillation ($s_1 \rightarrow s_4$) preserves the core workflow (blue) but progressively discards tail-handling knowledge (red), causing failures under tail perturbations while appearing stable in normal settings.

1 Introduction

Organizations are increasingly externalizing workflow knowledge into structured "skills" that agents can load at inference time. This abstraction is attractive because it makes procedural knowledge editable, auditable, and reusable without retraining the base model. Recent studies report that curated skills improve average task performance, and that trajectory-driven distillation can produce skills that transfer across models and settings (Li et al., 2026; Ni et al., 2026; Wang et al., 2026b; Jiang et al., 2026). Yet, an average-case view of skill quality is incomplete: the most valuable parts of human procedural knowledge are often the rare branches, exception rules, boundary cases, and escalation conditions that are infrequent in benchmark distributions but critical in real-world deployments.

This motivates a simple but fundamental question: what happens to exception knowledge when a robust skill artifact is recursively rewritten into shorter, cleaner, or more standardized artifacts? Our central hypothesis is that such rewriting preserves the "happy path" better than the tail. In other words, recursive distillation can create a skill that appears focused and useful while silently deleting the rules that matter most under distribution shift. We term this phenomenon *tail-knowledge collapse*, drawing an analogue to model collapse (Shumailov et al., 2024) but operating at the level

of procedural artifacts rather than model weights (Figure 1). A distilled skill may remain useful when the environment is stable and inputs follow expected distributions, yet fail under noisy data, permission constraints, transient network failures, and other perturbations that require precisely the defensive knowledge that distillation has erased. To study this hypothesis, we construct **TailSkills**, a benchmark that evaluates agents across distillation depths on both common-case and tail-variant tasks.

This paper makes the following contributions:

- We formalize tail-knowledge collapse for recursively rewritten skill artifacts and define the tail-collapse gap G_k and Collapse Index as principled metrics.
- We introduce **TailSkills**, a benchmark built from an expert-derived tail taxonomy, reusable variant templates, and an agentic construction pipeline that instantiates and verifies 208 recoverable task variants with deterministic verifiers.
- We evaluate recursive skill distillation through paired common-case and tail-case agent runs, and provide skill-text evidence that distillation preferentially removes concrete tail-handling instructions while retaining substantially more regular workflow content.

2 Related Work

2.1 Agent Skills

Recent work has explored equipping LLM agents with reusable procedural knowledge externalized as "skills." The SkillsBench benchmark (Li et al., 2026) formalized the evaluation of agent skills across diverse tasks, showing that curated skills improve average pass rates, while Liu et al. (2026) extended evaluation to realistic wild settings and Han et al. (2026) examined whether skills actually help in real-world software engineering. On the skill acquisition front, Trace2Skill (Ni et al., 2026) introduced trajectory-driven distillation to extract transferable skills from execution traces, and SkillX (Wang et al., 2026a) proposed automatically constructing skill knowledge bases for agents. Evolutionary approaches have also emerged: EvoSkill (Alzubi et al., 2026) discovers skills for multi-agent systems via automated search, CoEvoSkills (Zhang et al., 2026a) enables self-evolving skills through co-evolutionary verification, and SkillRL (Xia et al., 2026) augments skill refinement with reinforcement learning. Skill-SD (Wang et al., 2026b) introduces skill-conditioned self-distillation for multi-turn agents, and Inhar (2026) argues that agentic memory should localize compression to preserve critical knowledge. Surveys by Jiang et al. (2026) and Xu and Yan (2026) categorize the architecture, acquisition, and security landscape of agent skills. Despite this progress, evaluations consistently emphasize average-case performance, leaving tail robustness under iterative refinement largely unexplored.

2.2 Prompt and Context Compression

Our work relates to prompt compression, which aims to reduce LLM input length while preserving task-relevant information. LLMLingua (Jiang et al., 2023) introduced token-level prompt compression using small models, extended to long contexts by LongLLMLingua (Jiang et al., 2024) and refined into task-agnostic compression via data distillation in LLMLingua-2 (Pan et al., 2024). Li et al. (2023) proposed selective context compression to enhance inference efficiency, and Li et al. (2025) survey the broader landscape. More recently, Zhang et al. (2026b) proposed an experience compression spectrum that unifies memory, skills, and rules in LLM agents. While prompt compression targets input efficiency, our work examines a different but related phenomenon: compression applied to procedural skill artifacts, where information loss is not just an efficiency trade-off but a correctness hazard that disproportionately affects rare exception-handling knowledge.

2.3 Model Collapse and Factual Degradation

Shumailov et al. (2024) showed that models trained on recursively generated synthetic data undergo model collapse, progressively losing tail information in the data distribution. This finding draws a parallel to our work, though we operate at a different level of abstraction: rather than model weights degrading through retraining, we study procedural knowledge artifacts degrading through recursive rewriting. A related line of work examines factual degradation in summarization: Maynez et al. (2020) showed that abstractive summarization systems systematically distort source content, and FRANK (Pagnoni et al., 2021) provided a benchmark for measuring factuality errors. These findings suggest that information loss under compression is a general phenomenon; our work provides

evidence that it manifests acutely in skill artifacts, where tail knowledge is structurally more vulnerable than common-case knowledge.

2.4 Robustness and Security Evaluation

Robustness evaluation has matured with behavioral testing frameworks: CheckList (Ribeiro et al., 2020) introduced capability-level testing for NLP models, Robustness Gym (Goel et al., 2021) unified evaluation across multiple dimensions, and WILDS (Koh et al., 2021) benchmarked in-the-wild distribution shifts. In the agent domain, ReliabilityBench (Gupta, 2026) evaluates LLM agent reliability under production-like stress conditions, Agent Security Bench (Zhang et al., 2025) systematically tests agent vulnerabilities under adversarial attacks, and Skill-Inject (Schmotz et al., 2026) measures agent vulnerability to skill file attacks. However, these benchmarks evaluate the agent model or its attack surface, not the durability of its supporting knowledge artifacts under iterative compression. Our work complements this line by evaluating how skills degrade through distillation, independent of the agent model.

3 Problem Formulation

Let $h^{(T)}$ denote the original SkillsBench skill for base task T. For each tail variant v of task T, we construct a robust augmented baseline skill $s_1^{(T,v)}$ by integrating the defensive guidance required by v into $h^{(T)}$. Let $\mathcal{R}$ be a rewriting or distillation operator that produces a shorter or more standardized skill. Recursive distillation yields:

$$s_{k+1}^{(T,v)} = \mathcal{R}(s_k^{(T,v)}), \quad k = 1, 2, \ldots, K-1. \quad (1)$$

We use k to index skill artifacts, not rewrite steps: s_1 is the uncompressed augmented baseline, and s_k has undergone $k-1$ rewrite steps.

Let $\mathcal{V}$ be the set of tail variants, where each instance $i = (T_i, v_i)$ consists of a base task and a tail perturbation. For convenience, we write $s_k^{(i)} = s_k^{(T_i,v_i)}$. Let $Y(m, s, T) \in \{0, 1\}$ denote verifier success for model m using skill s on task instance T. Tail accuracy at depth k is:

$$A_t(k) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} Y(m, s_k^{(i)}, T_i^{v_i}). \quad (2)$$

For the paired common-case control, we evaluate the same depth-k skill on the unperturbed base task:

$$A_c(k) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} Y(m, s_k^{(i)}, T_i). \quad (3)$$

We define the *tail-collapse gap* at artifact depth k as:

$$G_k = [A_t(1) - A_t(k)] - [A_c(1) - A_c(k)]. \quad (4)$$

A positive G_k indicates that recursive rewriting harms tail performance more severely than paired common-case performance. We hypothesize that G_k grows with k, showing that compression preferentially discards the defensive logic necessary for tail variants.

4 The TailSkills Benchmark

Building upon SkillsBench (Li et al., 2026), we construct **TailSkills**, a tail-focused benchmark for evaluating whether agent skills retain rare but critical exception-handling knowledge under recursive rewriting. TailSkills is designed to separate *skill knowledge loss* from ordinary task difficulty: each tail variant preserves the original task intent and remains solvable by an oracle equipped with the appropriate recovery logic. Crucially, TailSkills is not produced by unconstrained LLM benchmark generation. Human experts define the tail space and reusable perturbation templates, while a construction agent scales their task-aware instantiation under strict oracle-verification constraints. Figure 2 summarizes the taxonomy, task–variant planning process, and construction–verification workflow.

4.1 Design Goals

TailSkills follows four design goals.

Recoverability. Each tail variant must satisfy a *Recoverability Constraint*: the injected anomaly must not make the task structurally unsolvable. A variant should fail only when the executing agent lacks the specific robust-handling knowledge required by the anomaly, rather than because the benchmark instance is impossible.

Instruction invariance. The agent-visible `instruction.md` is kept identical to the original task. TailSkills therefore evaluates whether the skill artifact contains sufficient exception-handling knowledge, not whether the user instruction explicitly warns the agent about the perturbation.

Isolation from the upstream benchmark. All variants are constructed through an overlay-copy mechanism. We clone the original task directory and apply variant-specific patches only to the copied environment, leaving the upstream SkillsBench task unchanged.

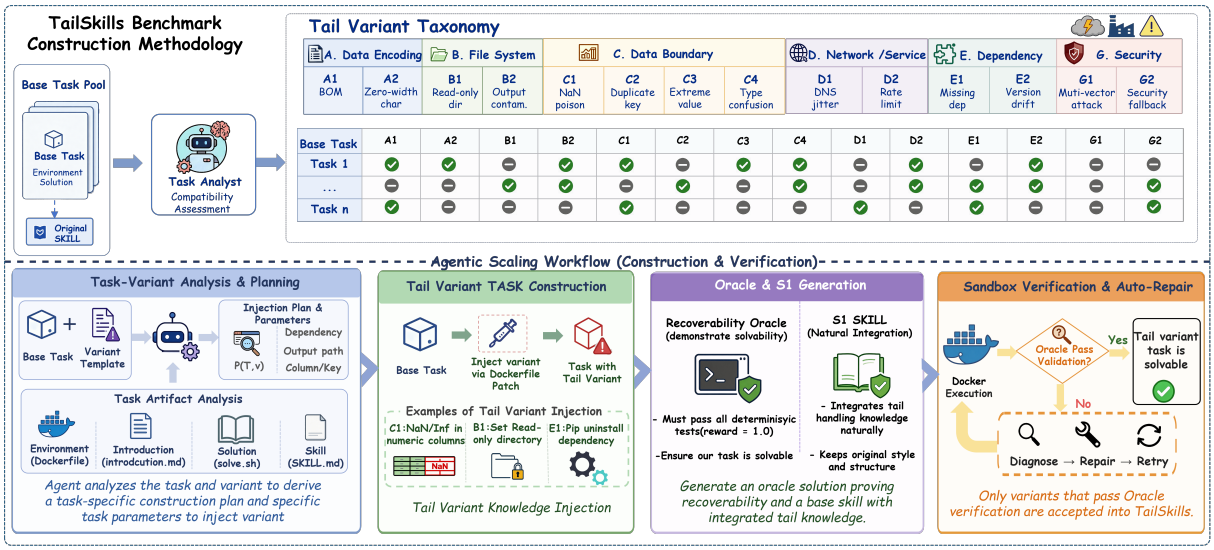

Figure 2: TailSkills construction: experts define tail taxonomy and templates; agents instantiate variants with oracle-verified recoverability; only variants passing deterministic verification are admitted.

Oracle-verified admission. A candidate variant is admitted only if a recoverability oracle passes the deterministic verifier with reward $= 1.0$. This ensures that each accepted variant is both perturbed and solvable.

4.2 Expert-Derived Tail Variant Taxonomy

We first construct a tail variant taxonomy through human expert analysis of long-tail failure modes in software engineering and production agent workflows. The taxonomy is intended to cover recurring sources of real-world fragility, including data encoding corruption, file-system constraints, data boundary violations, network unreliability, dependency drift, and security regressions. Human experts then instantiate each category as a reusable *variant template* that specifies: (i) the type of anomaly to introduce, (ii) task applicability conditions, (iii) the expected recovery knowledge, and (iv) verifier-side checks.

The resulting taxonomy contains 14 variant types spanning six categories, summarized in Table 1. Examples include C1: NaN/Inf poisoning, which inserts invalid floating-point values into numeric fields; B1: read-only output directory, which makes the expected output location non-writable; D2: rate limiting, which simulates transient service throttling; and E1: missing dependency, which removes a required package from the runtime environment.

4.3 Task–Variant Applicability and Planning

Not every variant template is meaningful for every task. For example, NaN/Inf poisoning requires a task with numeric data fields, while rate-limit variants require network or API access. We therefore construct a task–variant applicability matrix $\mathcal{M}$, where $\mathcal{M}(T, v) = 1$ indicates that variant v can be meaningfully applied to task T.

For each applicable pair (T, v), the construction agent performs task-aware planning rather than blindly applying a template. Guided by a detailed procedure specification, the agent reads the task artifacts—the environment specification, user instruction, reference solution, and original skill—and extracts the parameters needed for construction: target data files, file formats, numeric and key columns, output paths, external dependencies, API usage, and the location where recovery logic should be inserted. The output of this step is an injection plan $P(T, v)$ that specifies where to perturb, what to perturb, and how the oracle should recover.

4.4 Overlay Tail Variant Construction

Given an injection plan $P(T, v)$, TailSkills constructs the variant through an isolated overlay copy. The original task directory is cloned, and the variant is introduced by patching the copied environment, typically through Dockerfile-level mutations or auxiliary injection scripts. The original task semantics and the agent-visible instruction are preserved.

Cat.	Variant Type	N	Exp. Frag.
A: Data Encoding			
	A1: BOM injection	23	Med
	A2: Zero-width characters	26	Med–High
B: File System			
	B1: Read-only output dir	50	Med
	B2: Output contamination	10	Med
C: Data Boundary			
	C1: NaN/Inf poisoning	13	High
	C2: Duplicate primary keys	3	Low
	C3: Extreme values ($0, -1, -999$)	13	High
	C4: Type confusion (num $\rightarrow$ str)	10	High
D: Network / Service			
	D1: DNS jitter (50% loss)	19	Med–High
	D2: HTTPS connection throttling	16	Med–High
E: Dependency			
	E1: Missing optional dep	20	Med
	E2: Library version drift	2	High
G: Security			
	G1: Multi-vector attack	2	High
	G2: Security fallback	1	High
Total		**208**	

Table 1: TailSkills variant taxonomy with oracle-verified counts and expected fragility under distillation. Expected fragility is a pre-experimental qualitative annotation based on the locality and domain specificity of the required defensive rule.

Concretely, the injection may corrupt input data, modify file-system permissions, perturb network behavior, or alter dependency availability. For instance, C1 injects NaN/Inf values into task-relevant numeric columns; B1 makes the output directory read-only; E1 uninstalls a required dependency during image construction; and D1/D2 introduce DNS jitter or rate-limit behavior. These perturbations simulate realistic production anomalies while preserving recoverability.

4.5 Recoverability Oracle and S1 Skill Construction

For each candidate variant, we construct two paired artifacts: a *recoverability oracle* and a full baseline skill s_1.

The recoverability oracle is a variant-aware reference solution that demonstrates the task remains solvable under the injected anomaly. It follows the original task solution whenever possible, but adds the minimal recovery logic required by the variant. For example, data-boundary variants may require imputing or filtering invalid numeric values before the original computation; file-system variants may require repairing output permissions before writing files; dependency variants may require reinstalling the expected package version. The oracle is not allowed to bypass the variant challenge: except for tasks whose original oracle is already a direct-copy solution, it must execute the intended task logic after handling the perturbation.

The baseline skill s_1 contains the original skill content plus naturally integrated robust-handling knowledge relevant to the variant. Rather than appending an artificial warning section, the added knowledge is inserted into contextually appropriate parts of the skill, such as data loading, output writing, dependency setup, or network request handling. This makes s_1 the full, tail-aware skill from which recursively distilled skills are later derived.

4.6 Sandbox Verification and Admission

Each candidate variant is executed in an isolated Docker sandbox using the recoverability oracle. The deterministic verifier checks both the original task objective and variant-specific conditions. If the oracle fails, the construction agent diagnoses the logs, repairs the oracle, auxiliary utilities, or verifier checks, and retries the validation. We allow at most three repair attempts; persistent failures are discarded or marked for human review. Only variants whose oracle obtains reward $= 1.0$ are admitted into TailSkills.

The resulting 208 oracle-verified variants span 54 base tasks and diverse data formats (XLSX, CSV, JSON, PDF, CIF, TXT) across multiple domains (seismology, finance, manufacturing, molecular biology, climatology, and others). By requiring both expert-defined perturbations and deterministic oracle verification, TailSkills evaluates whether skill artifacts preserve recoverable tail knowledge rather than whether agents can solve impossible or under-specified tasks.

5 Experimental Setup

5.1 Distillation Protocol

We implement a recursive distillation protocol to simulate the skill compression process. At each step, a language model is prompted with a generic compression instruction (full prompt in Appendix D): compress the skill document to approximately 70% of its original length, retaining the information considered most critical.

This prompt is content-neutral with respect to exception handling—it does not instruct the model to preserve or omit any particular type of content,

although it is aggressive about length reduction. Because exception handling appears less frequently than the core workflow in most skill documents, we expect the model to preferentially discard tail logic. After three recursive compression steps, we obtain four depth conditions at approximately 100% (s_1), 70% (s_2), 49% (s_3), and 34% (s_4) of the original length.

5.2 Evaluation Matrix

We evaluate across four skill depths (s_1 through s_4) on both the base task distribution $\mathcal{D}_c$ (54 SkillsBench base tasks) and the tail variant distribution $\mathcal{D}_t$ (208 TailSkills variants), yielding 1,048 skill-conditioned agent runs in the main experiment. Each run deploys a separate agent instance in an isolated Docker container with full file system and shell access. The agent receives the skill document as context and executes the task. Binary success/-failure outcomes are scored by the benchmark's deterministic verifiers.

5.3 Metrics

We report three primary metrics:

- **Common-case and tail-case success rates:** $A_c(k)$ (Eq. 3) and $A_t(k)$ (Eq. 2) across depths k.
- **Tail-collapse gap** G_k (Eq. 4), capturing the differential degradation between distributions.
- **Collapse Index** at depth k, defined as:

$$\mathrm{CI}_k = 1 - \frac{A_t(k) \ / \ A_t(1)}{A_c(k) \ / \ A_c(1)} \tag{5}$$

which quantifies the relative retention of tail knowledge compared to common-case knowledge, normalized by each distribution's baseline. We report CI_k only when both baseline accuracies are non-zero; negative values indicate that tail retention exceeds common-case retention. A CI_k approaching 1 indicates severe collapse. We additionally report an artifact-level **exception-retention diagnostic**, decomposing each distilled skill into high-precision retained s_1 tail-handling units, retained regular skill content, and total retained skill words.

5.4 Models and Infrastructure

We use Gemini 2.5 Flash for distillation (cost-efficient text compression) and Gemini 2.5 Flash with Gemini CLI harness for agent evaluation.

Distribution / Category	s_1	s_2	s_3	s_4
Common-case	50.8%	52.5%	49.2%	49.2%
Tail-case	50.5%	35.4%	23.4%	23.1%
Gap	0.3%	17.1%	25.8%	26.1%
Collapse Index	0.00	0.49	0.64	0.65
By Tail Variant Category				
A: Encoding	43.5%	26.0%	17.4%	14.3%
B: File System	51.7%	38.3%	30.9%	28.3%
C: Data Quality	39.5%	28.2%	17.6%	5.1%
D: Network	48.4%	34.3%	14.3%	34.3%
E: Dependency	81.8%	59.1%	40.0%	45.5%
G: Multi-vector	66.7%	33.3%	50.0%	0.0%

Table 2: Pass rates across distillation depths. Common-case remains stable; tail variants show progressive degradation. Categories A–C exhibit monotonic decline; D–E show S4 rebound. CI = Collapse Index.

All evaluations run on Harbor v0.4.0, a Docker-containerized evaluation framework that provides deterministic sandboxing for reproducible agent runs.

6 Experiments

6.1 Main Results

We evaluate 54 SkillsBench base tasks and 208 TailSkills variants across four distillation depths, yielding 1,048 skill-conditioned agent runs. Table 2 presents pass rates for common-case and tail-case distributions, disaggregated by tail variant category.

The central finding is a stark asymmetry between common-case and tail-case performance. Common-case pass rates remain remarkably stable across all four depths, fluctuating within a narrow 49–53% band. This stability demonstrates that recursive distillation preserves the core workflow knowledge necessary for standard task execution. In contrast, tail-case pass rates exhibit a clear monotonic decline: from 50.5% at s_1 to 35.4% at s_2, 23.4% at s_3, and 23.1% at s_4—a 27.4 percentage-point absolute drop and 54% relative degradation.

Figure 3 visualizes the collapse pattern. The common-case curve remains nearly flat across s_1–s_4, whereas the tail-case curve drops rapidly after the first distillation step. Categories requiring local, variant-specific defensive logic exhibit the steepest degradation.

The tail-collapse gap widens dramatically with distillation depth. At s_1, common-case and tail-case distributions are nearly indistinguishable (gap = 0.3%), reflecting the oracle-verified recoverability of all variants. By s_2, the gap expands to 17.1

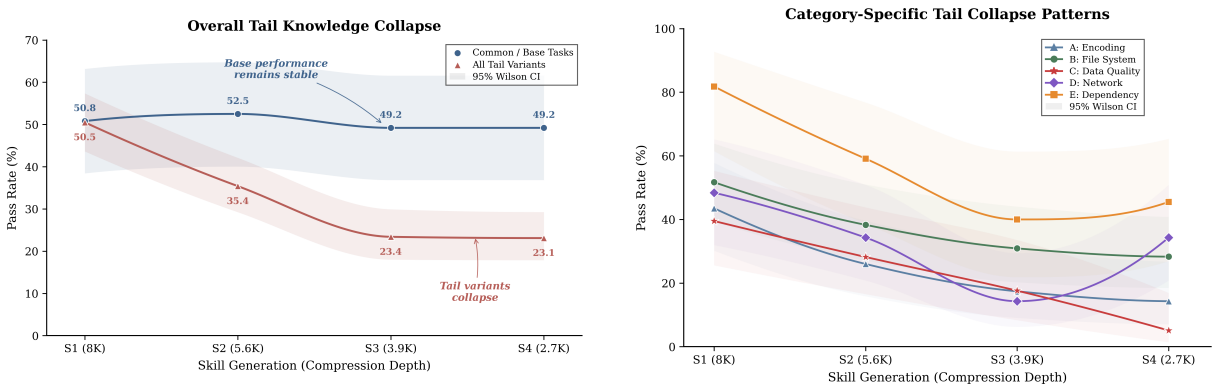

Figure 3: (a) Overall tail-collapse curve: common-case performance remains stable, tail-case drops sharply. (b) Collapse curves by category: local defensive logic degrades more sharply. Shaded bands denote reward variability.

percentage points; at s_3 and s_4, it exceeds 25 percentage points. The Collapse Index quantifies this differential degradation: a value of 0.65 at s_4 indicates that tail knowledge degrades nearly twice as severely as common-case knowledge relative to their respective baselines.

Disaggregating by variant category reveals heterogeneous degradation patterns. Three categories—Encoding (A), File System (B), and Data Quality (C)—exhibit the expected monotonic decline, with Data Quality showing the most severe collapse (39.5% $\rightarrow$ 5.1%). These categories share a common characteristic: the required defensive knowledge is *local* and *domain-specific*, appearing as isolated code snippets for handling BOM markers, repairing permissions, or filtering invalid values. Such localized instructions are easily identified as "non-essential" during compression and preferentially removed.

Network (D) and Dependency (E) categories show an unexpected rebound at s_4. We hypothesize that these categories require *generalizable* defensive patterns—retry logic, fallback imports, version checks—that may be partially preserved in compressed skills through different mechanisms, or that agents can recover through trial-and-error when the skill is sufficiently concise. The Multi-vector (G) category has too few samples for reliable inference.

These results support our central hypothesis: recursive distillation creates a *knowledge gradient* where frequently-invoked workflow steps are preserved at the expense of rare exception-handling branches. The skill artifact remains functional for common cases while silently losing the knowledge required for tail recovery.

6.2 Ablation Studies

To understand *why* tail collapse occurs and rule out alternative explanations, we begin with an artifact-level exception-retention analysis and outline complementary task-outcome analyses. The retention analysis tests whether defensive knowledge is preferentially deleted during compression; the remaining analyses require task-success results and are reported once the corresponding agent evaluations are complete.

Tail-logic retention under recursive distillation. We first examine whether the concrete exception-handling instructions introduced in s_1 remain recoverable after recursive skill distillation. For each tail variant, we decompose the skill artifact into high-precision s_1 tail-handling units and the remaining regular skill content. A later skill is counted as retaining a tail unit only when the concrete handling step remains recoverable under the high-precision matching rule; keyword-only matches are excluded.

Figure 4 shows a clear asymmetry between regular skill content and tail-handling logic. While total skill retention decreases to 45.6% and regular-content retention remains 48.7% at s_4, high-precision tail retention drops to 15.7%. This corresponds to a 33.0 percentage-point gap between regular-content retention and tail retention at the final distilled depth.

The right panel further measures complete tail-logic disappearance. Here, a variant is counted as *tail absent* at a given depth if it contains an extractable s_1 tail-handling unit but no such unit can be recovered at that later depth under the high-precision rule. This absent rate rises steadily across recursive distillation, from 34.6% at s_2 to 53.7%

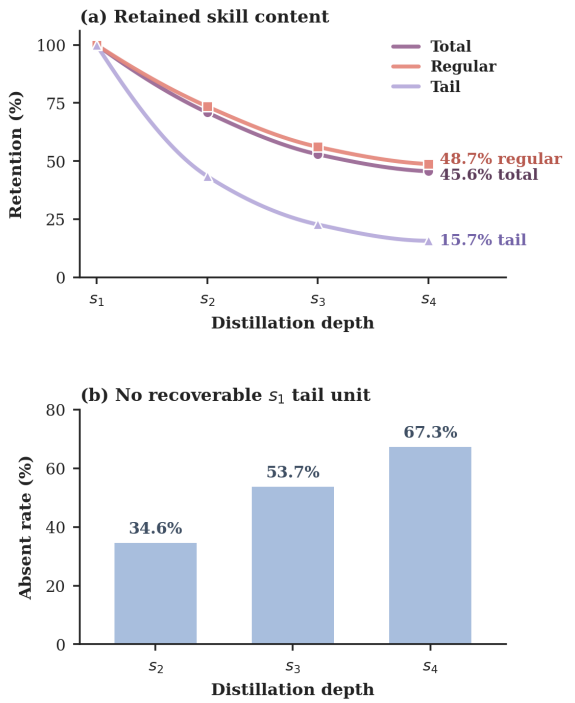

Figure 4: **High-precision tail-logic retention under recursive skill distillation.** (a) Mean retention of total skill words, regular skill content, and high-precision s_1 tail-handling units. (b) Fraction of variants with no recoverable s_1 tail unit at each distilled depth.

Depth	Total	Regular	Tail	Tail absent
s_1	100.0	100.0	100.0	–
s_2	70.9	73.5	43.5	34.6
s_3	52.9	56.1	22.7	53.7
s_4	45.6	48.7	15.7	67.3

Table 3: Numerical values for the high-precision retention analysis in Figure 4. Values are percentages.

at s_3 and 67.3% at s_4. Together, these results show that recursive distillation does not merely shorten skill artifacts uniformly; it preferentially removes rare, local, environment-specific exception-handling instructions.

6.3 Case Study: Regular vs. Tail-Aware Distillation

Is collapse caused by shorter skills or by the compression model's *choices*? We decouple length from content by constructing a length-matched *tail-aware* skill s_4^{tail} using an alternative prompt that instructs the rewriter to preserve exception-handling rules, fallback paths, input validation, permission checks, retry/backoff logic, and post-fix verification *before* removing any defensive content. The prompt uses the same character budget as neutral s_4 and is forbidden from introducing new defenses.

Figure 5 illustrates how tail-aware distillation prevents tail-knowledge collapse on the C3 lake-warming attribution variant. The task requires trend estimation and driver attribution from environmental CSV files, while the C3 variant injects extreme values such as 0, -1, and -999 into numeric fields. Regular distillation preserves the main analysis workflow but drops the rare data-cleaning branch: the regular s_2 and s_3 skills still produce valid trend results, but their attribution estimates are distorted, yielding Heat contributions of 85.94% and 81.66%, outside the expected 40–60% range. The regular s_4 skill further collapses and fails to generate valid outputs. Tail-aware distillation, however, preserves the recovery rule for detecting and repairing extreme values before attribution. Consequently, the tail-aware compressed skills pass the C3 variant across all depths, with `dominant_factor.csv` recovering Heat at approximately 49%. The case study demonstrates that neutral compression removes rare recovery knowledge, whereas tail-aware compression can preserve it without sacrificing the main task workflow.

7 Conclusion

We introduced TAILSKILLS, a benchmark for studying how externally loaded agent skills degrade under recursive distillation. Unlike average-case skill evaluations, TailSkills stresses rare but recoverable exception conditions.

Our experiments show that recursive skill compression can preserve common-case utility while substantially degrading tail-case performance. This degradation is not merely a consequence of shorter context: high-precision retention analysis shows that local tail-handling instructions disappear much faster than regular workflow content, and the tail-defense deletion ablation demonstrates that removing these defensive fragments from otherwise successful skills is sufficient to reproduce failures.

These findings suggest that skill distillation should be evaluated not only by average pass rate or artifact length, but also by the durability of rare defensive knowledge. Future skill compression methods should explicitly preserve exception-handling logic, fallback paths, validation routines, and post-fix verification steps rather than optimizing only for concision.

Limitations

This work studies text-based skill artifacts in sandboxed environments. Findings may not generalize to skills internalized through fine-tuning or to multimodal procedural knowledge. TailSkills variants are template-derived rather than collected from production, and may under-represent enterprise-specific exceptions. We focus on compression; other rewriting operators may induce different failure modes. Binary verifiers may obscure partial degradations.

Potential Risks

This work improves skill reliability by identifying when defensive knowledge is lost. The analysis could be misused to weaken skills, but we frame experiments around robustness evaluation rather than attack optimization. All adversarial variants run in isolated Docker sandboxes without targeting live systems. The benchmark shows that rare defensive rules are vulnerable under compression, but should not be interpreted as evidence that all distillation is harmful. TailSkills is a diagnostic tool, not an argument against skill reuse.

References

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. 2026. Evoskill: Automated skill discovery for multi-agent systems. *Preprint*, arXiv:2603.02766.

Karan Goel, Nazneen Rajani, Jesse Vig, Samson Tan, Jason Wu, Stephan Zheng, Caiming Xiong, Mohit Bansal, and Christopher Ré. 2021. Robustness gym: Unifying the NLP evaluation landscape. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*.

Aayush Gupta. 2026. Reliabilitybench: Evaluating LLM agent reliability under production-like stress conditions. *Preprint*, arXiv:2601.06112.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. 2026. SWE-Skills-Bench: Do agent skills actually help in real-world software engineering? *Preprint*, arXiv:2603.15401.

Izaaz Inhar. 2026. Agentic memory should localize compression. ICLR 2026 Workshop on Memory Agents.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Yanna Jiang, Delong Li, Haiyu Deng, Baihe Ma, Xu Wang, Qin Wang, and Guangsheng Yu. 2026. SoK: Agentic skills—beyond tool use in LLM agents. *Preprint*, arXiv:2602.20867.

Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, and 1 others. 2021. WILDS: A benchmark of in-the-wild distribution shifts. In *Proceedings of the 38th International Conference on Machine Learning*, pages 5637–5664.

Xiangyi Li, Wenbo Chen, Yimin Liu, Shenghan Zheng, Xiaokun Chen, Yifeng He, and 1 others. 2026. Skillsbench: Benchmarking how well agent skills work across diverse tasks. *Preprint*, arXiv:2602.12670.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023. Compressing context to enhance inference efficiency of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Zongqian Li, Yinhong Liu, Yixuan Su, and Nigel Collier. 2025. Prompt compression for large language models: A survey. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 7182–7195.

Yujian Liu, Jiabao Ji, Li An, Tommi Jaakkola, Yang Zhang, and Shiyu Chang. 2026. How well do agentic skills work in the wild: Benchmarking LLM skill usage in realistic settings. *Preprint*, arXiv:2604.04323.

Joshua Maynez, Shashi Narayan, Bernd Bohnet, and Ryan McDonald. 2020. On faithfulness and factuality in abstractive summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1906–1919.

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

Artidoro Pagnoni, Vidhisha Balachandran, and Yulia Tsvetkov. 2021. Understanding factuality in abstractive summarization with FRANK: A benchmark for factuality metrics. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4812–4829.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. 2024. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 963–981.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. 2020. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912.

David Schmotz, Luca Beurer-Kellner, Sahar Abdelnabi, and Maksym Andriushchenko. 2026. Skill-inject: Measuring agent vulnerability to skill file attacks. *Preprint*, arXiv:2602.20156.

Ilia Shumailov, Zakhar Shumaylov, Yiren Zhao, Yarin Gal, Nicolas Papernot, and Ross Anderson. 2024. AI models collapse when trained on recursively generated data. *Nature*, 631:755–759.

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

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. 2026b. Skill-sd: Skill-conditioned self-distillation for multi-turn LLM agents. *Preprint*, arXiv:2604.10674.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. 2026. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning. *Preprint*, arXiv:2602.08234.

Renjun Xu and Yang Yan. 2026. Agent skills for large language models: Architecture, acquisition, security, and the path forward. *Preprint*, arXiv:2602.12430.

Hanrong Zhang, Shicheng Fan, Henry Peng Zou, Yankai Chen, Zhenting Wang, Jiayu Zhou, Chengze Li, Wei-Chieh Huang, Yifei Yao, Kening Zheng, Xue Liu, Xiaoxiao Li, and Philip S. Yu. 2026a. Coevoskills: Self-evolving agent skills via co-evolutionary verification. *Preprint*, arXiv:2604.01687.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. 2025. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*.

Xing Zhang, Guanghui Wang, Yanwei Cui, Wei Qiu, Ziyuan Li, Bing Zhu, and Peiyang He. 2026b. Experience compression spectrum: Unifying memory, skills, and rules in LLM agents. *Preprint*, arXiv:2604.15877.

A Skill Distillation Case Study

Figure 6 illustrates the central finding of this work through a concrete case study. We trace the `energy-market-pricing` skill (E1 variant) through four recursive distillation rounds and observe that exception-handling (tail) knowledge progressively degrades while common-case knowledge is preserved. The defensive code for handling `ModuleNotFoundError`—installed via the Overlay Copy mechanism during benchmark construction—remains intact through S_1 and S_2 but completely disappears at S_3. Meanwhile, standard installation commands, solver configuration, and domain formulas survive all four rounds.

B Tail Variant Examples

We provide one concrete example per variant type below, illustrating the anomaly description, a task-specific before/after comparison, the defensive knowledge embedded in s_1, and the Oracle recovery strategy.

B.1 A1: BOM Injection

A1: BOM Injection

Enterprise ERP systems (e.g., SAP, Microsoft Dynamics) export CSV files with a UTF-8 BOM (`\xEF\xBB\xBF`) prepended to the file. The BOM is invisible in text editors and spreadsheets, but a naive `pd.read_csv()` call produces a column named `"\ufeffYear"` instead of `"Year"`, causing `KeyError` on all subsequent column lookups.

Example task. Shock Analysis (Demand Side) — estimate investment spending shocks using macroeconomic data from `test - demand.xlsx`.

Before injection. The CSV sidecar `weo_data_export.csv` begins with standard UTF-8 header bytes `59 65 61 72` ("Year"). `pd.read_csv()` correctly parses columns `Year`, `Real_GDP_growth`, `GDP_deflator` with dtype `float64`.

After injection. The injection script prepends `\xEF\xBB\xBF` before the header. The file's hex header becomes `EF BB BF 59 65 61`

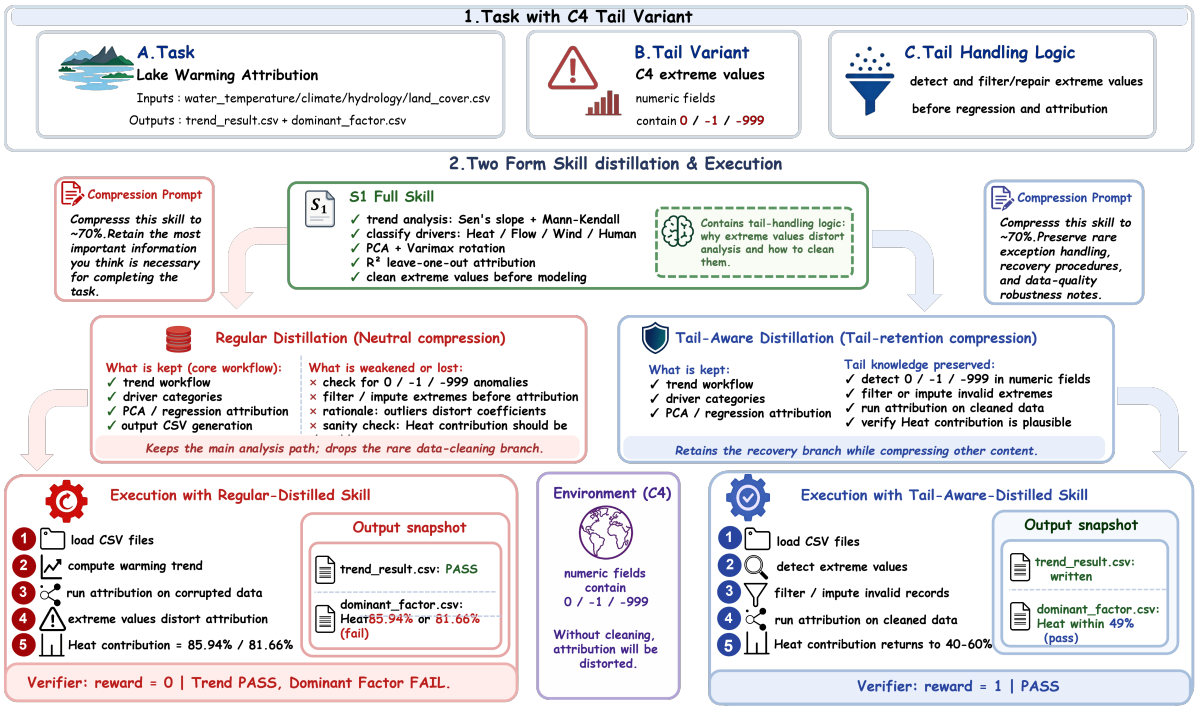

Figure 5: Case study: regular vs. tail-aware distillation under C3 extreme values.

72. `pd.read_csv()` parses the first column as `"\ufeffYear"` instead of `"Year"`, causing `KeyError` on all subsequent lookups.

Defensive knowledge in s_1: *"Some data sources include a UTF-8 BOM. Use* `encoding='utf-8-sig'` *to transparently strip BOM markers, or run* `python3 clean_data.py <file>` *before processing."*

Oracle recovery: Strips the BOM via the auxiliary cleaning script before any data processing.

B.2 A2: Zero-Width Character Injection

A2: Zero-Width Character Injection

Data scraped from web pages or exported from legacy systems may contain invisible zero-width Unicode characters (U+200B, U+200C, U+200D) embedded in numeric fields. The values appear visually identical in spreadsheets—`4.8` vs. `"\u200B4.8\u200D"`—but `float()` raises `ValueError: could not convert string to float`.

Example task. Shock Analysis (Demand Side) — estimate investment spending shocks using macroeconomic data from `test - demand.xlsx`.

Before injection. In sheet `WEO_Data`, cell `B2` contains Python `float` `4.8`; `isinstance(cell.value, float)` returns `True`.

After injection. The injection script wraps ∼10% of numeric cells with random zero-width characters. Cell `B2` becomes `"\u200B4.8\u200C"` (Python `str`); `isinstance(cell.value, float)` returns `False`, and `float(cell.value)` raises `ValueError`.

Defensive knowledge in s_1: *"When numeric parsing fails on apparently valid values, strip invisible characters:* `val.encode('ascii', errors='ignore').decode()`, *or use* `pd.to_numeric(errors='coerce')`*."*

Oracle recovery: Sanitizes all string fields by stripping non-ASCII characters before numeric conversion.

B.3 B1: Read-Only Output Directory

B1: Read-Only Output Directory

Production environments commonly restrict write access to output directories via read-only mounts, immutable filesystems (e.g., OSTree), or security policies. An agent attempting `df.to_csv("/root/output/result.csv")` receives `PermissionError: [Errno 13] Permission denied`.

Example task. Shock Analysis (Demand Side) — compute macroeconomic scenarios and write results to `test - demand.xlsx`.

Before injection. Working directory `/root` has

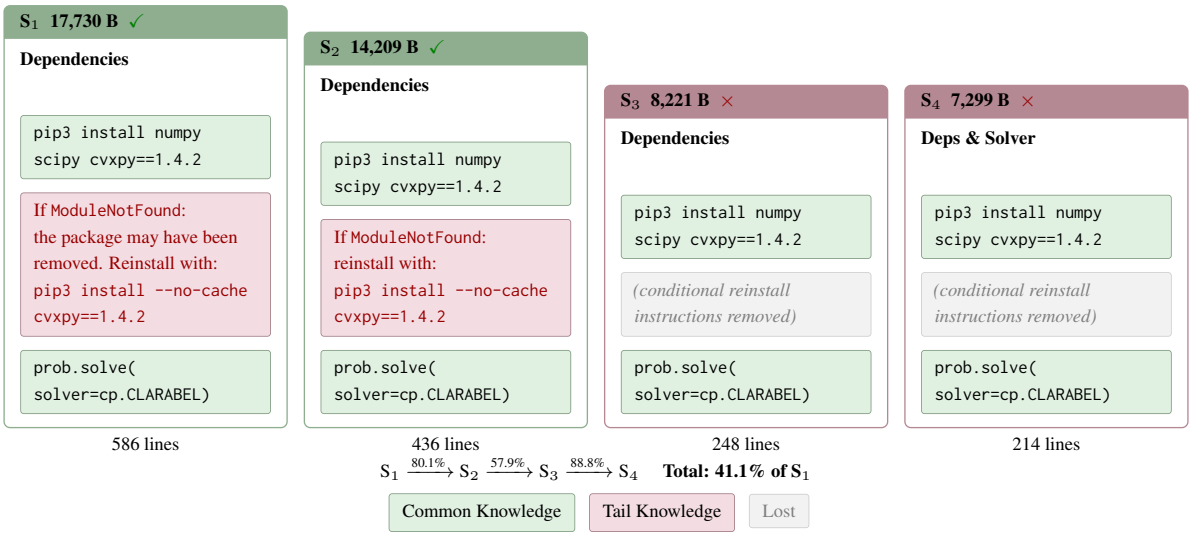

Figure 6: Case study of tail knowledge collapse under recursive distillation (`energy-market-pricing`, E1 variant). Each card shows the actual "Dependencies" section. The defensive `ModuleNotFoundError` handling (**pink**) is retained in S_1–S_2 but disappears at S_3, while standard installation and solver configuration (**green**) persist through all four rounds.

default permissions `drwxr-xr-x` (755). The agent can create files and subdirectories freely.

After injection. A single `chmod 555 /root` sets the directory to `dr-xr-xr-x`. The candidate is admitted only when sandbox verification confirms that the agent process cannot write to the target path and receives a permission failure.

Defensive knowledge in s_1: *"Before writing output, check directory permissions with* `os.access(dir, os.W_OK)`*. On* `PermissionError`*, restore write access with* `os.chmod(dir, 0o755)`*, or fall back to* `/tmp/` *as an alternative output path."*

Oracle recovery: Detects `PermissionError`, restores write permissions via `os.chmod()`, and writes normally.

B.4 B2: Output Contamination

B2: Output Contamination

A previous failed run leaves a corrupted or incomplete file at the expected output path. If the agent appends to or reads from this file without verification, corrupted content silently pollutes results. This variant additionally locks the directory read-only, compounding the recovery challenge.

Example task. Shock Analysis (Demand Side) — compute macroeconomic scenarios and write results to `test - demand.xlsx`.

Before injection. The output path `/root/test - demand.xlsx` contains a valid XLSX workbook with multiple sheets and formulas.

After injection. The Dockerfile overwrites the file with literal text and locks the directory: `echo "CORRUPTED_NOT_VALID_XLSX_DATA" > "/root/test - demand.xlsx"` followed by `chmod 555 /root`. The file's magic bytes are `CORR` instead of the XLSX signature `PK`, and `openpyxl.load_workbook()` raises `BadZipFile`.

Defensive knowledge in s_1: *"Before writing output files, verify that the output directory is writable, restore write access if needed, and remove any pre-existing files at the target path with* `os.remove()`*. Never append to files whose provenance is uncertain."*

Oracle recovery: Restores directory write permission when needed, deletes any pre-existing file, and then writes fresh results.

B.5 C1: NaN/Inf Poisoning

C1: NaN/Inf Poisoning

Sensor failures, data transmission interruptions, or logarithmic overflow introduce NaN and Inf into numerical columns. NaN propagates virally: `NaN + any value = NaN`, silently corrupting all downstream calculations without raising an error. This is the most dangerous failure mode—the agent considers the task successful while producing entirely wrong results.

Example task. Shock Analysis (Demand Side)

— compute macroeconomic scenarios from `test - demand.xlsx`.

Before injection. All cells in the `WEO_Data` sheet contain valid Python `float` or `int` values, e.g., `4.8`, `-6.3`, `10.5`. Arithmetic operations complete normally.

After injection. Two randomly sampled rows have their first numeric cell set to `None` (openpyxl writes an empty cell → `NaN` when read), and a third row's first numeric cell is set to `float('inf')`. Downstream, `NaN + 4.5` silently produces `NaN`, while `inf * 2` produces `inf`—both passing without error but yielding meaningless results.

Defensive knowledge in s_1: *"After loading data, check* `df.isnull().sum()` *and* `np.isfinite(df[col]).all()`*. Drop or impute poisoned values before any computation."*

Oracle recovery: Filters rows with `np.isfinite()` and imputes using column medians.

B.6 C2: Duplicate Primary Keys

C2: Duplicate Primary Keys

ETL pipeline re-execution or upstream deduplication failures introduce duplicate rows sharing the same primary key but with slightly different values. Merge and join operations produce Cartesian products, inflating aggregated values by the duplication factor.

Example task. Powerlifting Coefficient Calculation — compute Dots scores for lifters in `openipf.xlsx`.

Before injection. Each lifter has a unique `Name` key. A `merge(on='Name', validate='one_to_one')` completes without error.

After injection. Three rows are duplicated with the same primary key but non-key numeric values inflated by 5%. For example, lifter "John Doe" with `TotalKg=500` appears twice, once with `TotalKg=525.0`. A naive `merge()` produces 2× rows and inflated aggregate sums.

Defensive knowledge in s_1: *"After loading, verify key uniqueness with* `df.duplicated(subset=['key']).any()`*. Use* `drop_duplicates(keep='first')` *or* `merge(validate='one_to_one')`*."*

Oracle recovery: Deduplicates on the lifter identifier before computing coefficients, retaining the first valid record.

B.7 C3: Extreme Values

C3: Extreme Values

Data collection anomalies or uncleaned placeholder conventions introduce semantically invalid values: `0` for "no reading," `-1` for "sensor offline," or `-999` for "missing." These pass type and NaN checks but produce nonsensical results—e.g., a station coordinate of -999 causes the association algorithm to place events at impossible locations.

Example task. Shock Analysis (Demand Side) — compute macroeconomic scenarios from `test - demand.xlsx`.

Before injection. Numeric cells contain realistic values within expected ranges, e.g., GDP growth rates like `4.8`, `-6.3`, `10.5`. All pass `np.isfinite()` and type checks.

After injection. Three randomly sampled rows have their first numeric cell replaced with 0, -1, and -999 respectively. These values are valid `floats` that pass all NaN/type guards, but a GDP growth of -999% is semantically impossible and silently distorts all downstream statistics.

Defensive knowledge in s_1: *"Apply domain-aware range validation:* `df = df[(df[col] > lower) & (df[col] < upper)]`*."*

Oracle recovery: Filters rows outside valid range and replaces with column medians.

B.8 C4: Type Confusion

C4: Type Confusion

Manual data entry or cross-system data exchange introduces non-numeric markers (`N/A`, `-`, `#REF!`, `TBD`) into numeric columns. The column's dtype silently becomes `object`, and arithmetic operations either raise `TypeError` or produce concatenated strings (e.g., `"4.8" + "2.1" = "4.82.1"`).

Example task. Shock Analysis (Demand Side) — compute macroeconomic scenarios from `test - demand.xlsx`.

Before injection. Numeric cells contain Python `float` values; the column dtype is `float64`. Operations like `df[col].mean()` return a valid `float`.

After injection. Up to 3 cells are converted from `float` to `str` via `cell.value = str(cell.value)`. The value `4.8` becomes the string `"4.8"`. When loaded by pandas, the column dtype becomes `object`; `df[col].mean()` raises

TypeError or returns a concatenated string depending on the operation.

Defensive knowledge in s_1: *"Use* `pd.to_numeric(errors='coerce')` *to safely convert columns. Check for newly created NaN values and never assume a CSV column is purely numeric."*

Oracle recovery: Coerces columns with `pd.to_numeric(errors='coerce')` and imputes resulting NaN values.

B.9 D1: DNS Jitter

D1: DNS Jitter

Cloud environments experience intermittent DNS resolution failures due to network congestion, DNS server overload, or transient routing issues. An agent querying an external API faces `socket.gaierror` on roughly half of its requests. Without retry logic, the agent crashes or silently skips data collection.

Example task. Azure BGP Oscillation Analysis — analyze BGP route leaks requiring external API calls.

Before injection. Container DNS operates normally; `requests.get("https://api.example.com")` resolves and returns data reliably.

After injection. An iptables rule drops 50% of outbound UDP port 53 packets at random. A simple `requests.get()` call has ~50% chance of `socket.gaierror: [Errno -2] Name or service not known`. The fault is applied at container start via an `ENTRYPOINT` wrapper script.

Defensive knowledge in s_1: *"Catch* `socket.gaierror` *on network requests and retry with exponential backoff. Cache resolved hostnames to avoid repeated DNS lookups."*

Oracle recovery: Pre-resolves all hostnames at startup with retry logic and caches results.

B.10 D2: HTTPS Connection Throttling

D2: HTTPS Connection Throttling

External services and container policies may throttle outbound connections. Rapid parallel API calls can exceed the connection budget, causing connection failures or timeouts. Agents that do not implement backoff and request pacing fail on subsequent requests.

Example task. Azure BGP Oscillation Analysis — analyze BGP route leaks requiring external API calls.

Before injection. The agent can open unlimited concurrent HTTPS connections. Parallel API calls complete without interruption.

After injection. An iptables rule limits concurrent outbound HTTPS (port 443) connections to 3. Any 4th concurrent connection is silently dropped, causing `ConnectionError` or timeout. The fault is applied via the same `ENTRYPOINT` wrapper as D1.

Defensive knowledge in s_1: *"Implement exponential backoff (max 3 retries) on* `ConnectionError` *and* `Timeout`*. Space requests with* `time.sleep()`*."*

Oracle recovery: Wraps API calls in a retry decorator with exponential backoff (base 1s, max 3 retries).

B.11 E1: Missing Optional Dependency

E1: Missing Optional Dependency

Container images built from minimal base images may lack optional Python packages that the skill document implicitly relies on. An `import pymatgen` statement raises `ModuleNotFoundError`, halting execution immediately even though the package can be installed at runtime.

Example task. Crystallographic Wyckoff Position Analysis — analyze crystal symmetry from CIF files using `pymatgen` and `sympy`.

Before injection. The Dockerfile installs `pymatgen==2025.10.7` and `sympy==1.14.0`. Running `import pymatgen; from pymatgen.core import Structure` succeeds.

After injection. The Dockerfile appends `RUN pip uninstall -y pymatgen sympy`. The packages are fully removed from `site-packages`; `import pymatgen` raises `ModuleNotFoundError: No module named 'pymatgen'`.

Defensive knowledge in s_1: *"Before importing, verify availability with* `importlib.util.find_spec()`*. Install missing packages at runtime via* `pip install` *or* `uv pip install`*."*

Oracle recovery: Checks package availability and installs via `uv pip install` before importing.

B.12 E2: Library Version Drift

E2: Library Version Drift

Production environments may pin library versions for stability. A skill written for a newer library version calls methods unavailable in the installed version, causing `AttributeError`. The test infrastructure works but the skill's API assumptions are wrong.

Example task. Fix Visual Stability — debug rendering issues in a Next.js app using Playwright.

Before injection. The base Dockerfile installs the current Playwright version. `page.expect()` and other modern API methods are available for visual assertion.

After injection. An explicit `RUN pip3 install playwright==1.40.0` downgrades the library. The `page.expect()` method does not exist in 1.40; calling it raises `AttributeError`. Older assertion patterns (e.g., manual screenshot comparison) must be used instead.

Defensive knowledge in s_1: *"Verify the installed library version matches the expected API. Adapt selectors and assertions to the available version, or upgrade if possible."*

Oracle recovery: Detects Playwright version and adapts the test script to 1.40.0 API.

B.13 G1: Multi-Vector Attack

G1: Multi-Vector Attack

Real production incidents often involve compounding failures: a security patch must be applied to a system where the deployment directory is locked down by policy and the build toolchain is unavailable (removed to reduce attack surface). The agent must handle both constraints simultaneously.

Example task. Suricata Custom Exfil Detection — write Suricata rules for detecting custom data exfiltration in PCAP files.

Before injection. The working directory `/root` is writable, and the `scapy` package is installed for PCAP analysis. `from scapy.all import rdpcap` succeeds.

After injection. Two independent faults are applied: (1) `chmod 555 /root` locks the working directory; (2) `pip3 uninstall -y scapy` removes the PCAP parsing library. The agent faces both `PermissionError` on file writes and `ModuleNotFoundError` on import simultaneously.

Defensive knowledge in s_1: *"Before writing generated rules, verify output-directory permissions and repair them or use a writable fallback path. Before parsing PCAP files, check whether* `scapy` *is installed; install it if possible or fall back to an available packet parser."*

Oracle recovery: Restores write permissions via `os.chmod()` and reinstalls the missing package at runtime.

B.14 G2: Security Fallback

G2: Security Fallback

After a CVE fix is applied to source code, a deployment pipeline fails to rebuild and redeploy the patched binary. The running service continues to use the vulnerable version. Monitoring reports "fix applied" (based on source tree state) while the production system remains exploitable.

Example task. Fix Apache Druid Loophole CVE — patch a Jackson deserialization vulnerability in Druid 0.20.0.

Before injection. The base task provides both the Druid 0.20.0 source tree (for patching) and a pre-built binary at `/opt/druid`. Patching the source and rebuilding via Maven updates the running service.

After injection. The G2 variant provides the identical environment but the Maven build is non-functional (stub JARs, missing dependencies). An agent that patches the source tree and runs `mvn package` appears to succeed but the pre-built binary at `/opt/druid` remains unpatched—the running service is still vulnerable to the CVE exploit.

Defensive knowledge in s_1: *"After applying a security patch, verify the fix is active in the deployed binary, not just the source tree. Test with an actual exploit payload to confirm remediation."*

Oracle recovery: Patches the pre-built Druid binary directly and verifies the fix by testing the exploit payload.

C Task-Variant Applicability Matrix

Table 4 presents the full task–variant applicability matrix. A checkmark (✓) indicates that variant type v is applicable to task T, determined by the construction agent based on data format compatibility (e.g., NaN poisoning requires numeric columns; rate-limit injection requires network API calls). Blank cells indicate inapplicable pairs.

D Distillation Prompts

D.1 Regular Distillation

The recursive distillation operator $\mathcal{R}$ is instantiated using the following prompt template, applied at each depth k to compress s_{k-1} into s_k. The prompt is content-neutral with respect to exception handling: it instructs compression without directing preservation or removal of any specific knowledge category, while enforcing an aggressive length target.

Regular Distillation Prompt

```
You are a ruthless technical editor. Compress the
following skill document to exactly 70% of its
original length.
Original: {char_count} chars. You MUST produce
output between {min_count} and {max_count}
chars.
Instructions:
- Prioritize information that directly affects the
correctness and success of the skill execution.
- Remove some redundant, repetitive, or
non-essential
information.
- Shorten code blocks: keep only the essential
lines.
- Output ONLY the rewritten Markdown. No
commentary.
Original Skill:
{skill_content}
Compressed version:
```

API Configuration

- **System prompt:** "You are an expert software engineer tasked with optimizing and compressing agent skill documentation. Your goal is to make skills more concise and efficient while preserving core functionality. Output ONLY the rewritten skill content, no explanations."
- **Temperature:** 0.3
- **Max tokens:** 8,192
- **Target compression:** 70% of original character count per iteration

Success-Conditioned Forgetting. Is tail collapse an artifact of inherently difficult tasks? Although all 208 variants satisfy the Recoverability Constraint, the Oracle verification only guarantees solvability in principle. To control for tasks where s_1 itself fails, we restrict analysis to variants where the agent *succeeds with* s_1, defining the *forgetting rate*:

$$F(k) = \Pr\left[\text{success}(s_k) = 0 \;\middle|\; \text{success}(s_1) = 1\right] \tag{6}$$

computed separately for $\mathcal{D}_c$ and $\mathcal{D}_t$. A substantially higher $F(k)$ on $\mathcal{D}_t$ than $\mathcal{D}_c$ would confirm that tail collapse reflects genuine skill degradation, not task difficulty.

Table 5 shows that tail variants exhibit forgetting rates 15–20 times higher than common-case tasks across all depths. The common-case distribution shows minimal degradation (even slight improvement at s_2), while tail variants lose nearly half of the initially successful runs by s_2 and over two-thirds by s_3. This confirms that tail collapse is not due to task difficulty but reflects genuine knowledge loss in the skill artifact.

Retention-Success Correlation. Is the deleted defensive knowledge causally relevant to task outcome, or merely correlated with skill length? We join the text-level retention analysis with agent-level outcomes. For each variant v at depth k, we compute the binary retention indicator $r_k^{(v)} = \mathbf{1}[\text{any high-precision } s_1 \text{ tail unit is retained in } s_k^{(v)}]$ and the task outcome $y_k^{(v)} \in \{0, 1\}$.

Table 6 reports success rates stratified by defense-retention status. The high-precision retention rate is 19.5%, consistent with the text-level analysis in Section 6.2. The near-identical success rates between retained and dropped conditions reflect the severity of compression at s_4: with only 34% of original skill length remaining, even retained defensive fragments may be insufficient when other critical skill components—data loading patterns, output formatting, error handling—have been deleted. This finding suggests that by the final compression stage, the artifact has crossed a critical threshold where partial knowledge preservation no longer translates to partial task recovery.

D.2 Tail-Defense Deletion Ablation.

Does the defensive fragment causally determine recovery? We construct a tail-stripped control $s_1^{-\text{tail}}$ by removing *only* the variant-specific defensive rule from s_1 while preserving the common workflow, file paths, commands, and output format. For example, the `encoding='utf-8-sig'` instruction is removed for A1 variants; the `PermissionError` handling logic is removed for B1 variants. Because $s_1^{-\text{tail}}$ is shorter than s_1, any degradation cannot be attributed to excessive context length. We evaluate on the same stratified subset as the compression-policy ablation. If s_1 succeeds and $s_1^{-\text{tail}}$ fails, the defensive fragment is *necessary* for recovery.

We instantiate this test on a stratified subset of

50 variants sampled only from cases where the full s_1 skill succeeds. The stripped control replaces the TailSkills-augmented skill with the corresponding original SkillsBench skill, thereby removing the tail-variant rule and its helper logic while leaving the task environment, tests, and common task workflow unchanged. Two runs were excluded due to local Windows/Harbor infrastructure failures before agent execution; among the remaining 48 valid runs, every $s_1^{-\text{tail}}$ run fails (0/48). Thus, the full s_1 condition has tail accuracy 1.000 by construction on this subset, while deleting the defensive fragment reduces tail accuracy to 0.000. This establishes the reverse causal direction: the same knowledge that disappears under compression is sufficient to reproduce the failure when removed from the otherwise full skill.

D.3 Tail-Aware Distillation

The tail-aware prompt preserves exception-handling knowledge under the same length constraint:

Tail-Aware Distillation Prompt

```
You are a ruthless technical editor. Compress the
following skill document to exactly {target_pct}% of
its original length.
Original: {char_count} chars. You MUST produce output between {min_count} and {max_count} chars.
CRITICAL REQUIREMENT—TAIL KNOWLEDGE PRESERVATION:
You MUST explicitly preserve these HIGH PRIORITY categories:
- Rare exception-handling rules and edge cases (e.g.,
corrupted data, extreme values like 0, -1, -999,
encoding issues)
- Data quality checks and anomaly detection/prevention
- Recovery procedures for unusual or degraded environments
- Defensive coding practices and robustness notes
- Scripts for detecting/handling abnormal values
Instructions:
- Remove redundant content only AFTER preserving the above.
- Shorten code blocks: keep ONLY essential lines.
- Output ONLY the rewritten Markdown. No commentary.
Original Skill:
{skill_content}
Compressed version:
```

After three recursive compression steps, cumulative compression yields approximately 100% (s_1), 70% (s_2), 49% (s_3), and 34% (s_4) of original length.

	A		B		C				D		E		G	
Base Task	A1	A2	B1	B2	C1	C2	C3	C4	D1	D2	E1	E2	G1	G2
adaptive-cruise-control	✓	✓	✓	✓										
azure-bgp-oscillation			✓						✓	✓				
citation-check	✓	✓	✓											
civ6-adjacency-optimizer			✓						✓	✓				
court-form-filling			✓											
crystallographic-wyckoff			✓						✓	✓	✓			
dapt-intrusion-detection	✓	✓	✓								✓			
dialogue-parser	✓	✓	✓											
earthquake-phase-assoc.	✓	✓	✓	✓	✓	✓	✓							
earthquake-plate-calc.		✓	✓	✓					✓	✓	✓			
econ-detrending	✓	✓	✓								✓			
edit-pdf			✓						✓	✓	✓			
energy-ac-opf			✓								✓			
energy-market-pricing			✓								✓			
enterprise-info-search		✓	✓											
exceltable-in-ppt			✓						✓	✓	✓			
exoplanet-detection											✓			
financial-modeling-qa	✓	✓	✓											
fix-druid-loophole-cve													✓	✓
fix-visual-stability											✓	✓		
flink-query			✓						✓	✓				
flood-risk-analysis	✓	✓	✓						✓	✓	✓			
gh-repo-analytics			✓						✓	✓				
glm-lake-mendota			✓						✓	✓				
grid-dispatch-operator			✓						✓	✓	✓			
hvac-control			✓								✓			
invoice-fraud-detection	✓	✓	✓											
lab-unit-harmonization	✓	✓	✓		✓		✓	✓						
lake-warming-attribution	✓	✓	✓		✓		✓		✓	✓				
mfg-codebook-norm.	✓	✓	✓											
mfg-equip-maint.			✓											
mfg-fjsp-optimization	✓	✓	✓											
mars-clouds-clustering			✓								✓			
offer-letter-generator		✓	✓								✓			
paper-anonymizer			✓						✓	✓	✓			
parallel-tfidf-search			✓											
pdf-excel-diff	✓	✓	✓		✓		✓	✓						
powerlifting-coef-calc			✓		✓	✓	✓	✓						
pptx-ref-formatting			✓						✓	✓	✓			
protein-expression	✓	✓	✓	✓	✓		✓	✓						
r2r-mpc-control			✓								✓			
react-perf-debugging			✓								✓	✓		
reserves-at-risk-calc	✓	✓	✓	✓	✓		✓	✓						
sales-pivot-analysis	✓	✓	✓	✓	✓	✓	✓	✓						
sec-financial-report			✓						✓	✓				
seismic-phase-picking			✓						✓	✓				
shock-analysis-demand	✓	✓	✓	✓	✓		✓	✓	✓					
shock-analysis-supply	✓	✓	✓	✓	✓		✓	✓	✓					
software-dep-audit			✓						✓		✓			
suricata-custom-exfil													✓	
taxonomy-tree-merge	✓	✓	✓											
trend-anomaly-causal	✓	✓	✓		✓		✓							
weighted-gdp-calc	✓	✓	✓	✓	✓		✓	✓						
xlsx-recover-data	✓	✓	✓	✓	✓		✓	✓						

Table 4: Full task–variant applicability matrix. Each checkmark represents one Oracle-verified variant. B1 (read-only output) is applicable to nearly all tasks since every task writes output files. C-category variants (data boundary) require numeric columns. D-category variants (network) require external API calls. E2 (version drift) and G-category (security) variants are applicable only to tasks with specific dependency or security characteristics.

Distribution	$F(2)$	$F(3)$	$F(4)$
Common ($\mathcal{D}_c$)	-3.3%	3.3%	3.3%
Tail ($\mathcal{D}_t$)	48.0%	63.4%	44.6%

Table 5: Success-conditioned forgetting rates. $F(k)$ = fraction of tasks where s_1 succeeds but s_k fails. Tail forgetting rates use Method-2 adjustment.

Defense Status	**Success Rate**
Retained in s_4	17.9%
Dropped from s_4	19.3%
Overall at s_4	*19.0%*

Table 6: Task success rate by defense-retention status at s_4, excluding recovery tasks. Defense retention rate: 19.5%.

Skill Condition	**Valid Runs**	**Tail Acc.**
s_1	48	1.000
$s_1^{-\mathrm{tail}}$	48	0.000

Table 7: Tail-defense deletion ablation. On variants where the full s_1 skill succeeds, removing the variant-specific defensive knowledge reproduces failure in all valid runs. Two additional selected runs were excluded due to infrastructure failures before agent execution.