BRIDGING REFLECTIVE MEMORY AND PROCEDURAL SKILLS: CONTEXT-ANCHORED POLICY COMPILATION FOR LLM AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reflective memory systems for LLM agents preserve execution fidelity but incur substantial context overhead. Compressing reflective memories into reusable skills promises efficiency, yet recent benchmarks show distilled skills often underperform their source memories. Prior work attributes this to information loss during compression. We hypothesize that the core issue is **context-detachment**: procedural knowledge loses effectiveness when stored as global guidelines rather than anchored to the skill components that trigger it. To address this, we propose **Skill-Local Policy**, which anchors compiled procedures to relevant skill subcomponents rather than global guidelines. Controlled ablation on six SkillFlow task families confirms our hypothesis—identical content achieves 58.3% success as flat guidelines versus 83.3% when context-anchored. Large-scale evaluation on 347 matched pairs shows context-anchored compilation provides safety (zero within-family regressions) and modest effectiveness (+3.8pp overall), with benefits concentrated where procedural structure aligns with skill decomposition. Code and data are available at `https://anonymous.4open.science/r/skills-policy-C1BE`

1 INTRODUCTION

LLM agents learn from experience by retaining episodic traces—raw interaction trajectories that are converted into reflective memories encoding failure patterns, diagnostic rationales, and recovery cues (Shinn et al., 2023; Zhao et al., 2024). While reflective memories preserve execution-critical context in a more structured form than raw traces, they still incur substantial context overhead (10k+ tokens per task) and face retrieval interference as repositories scale.

Recent agent frameworks attempt to distill these reflective memories into reusable procedural skills—abstract action templates that can be invoked across similar tasks (Wang et al., 2023; Ni et al., 2026). This memory-to-skill compilation promises to reduce context overhead while enabling knowledge transfer. However, recent benchmarks reveal a puzzling finding: compressed skills often underperform the reflective memories from which they were derived (Lei et al., 2026; Zhang et al., 2026a). Prior work attributes this to **information loss during compression**—the hypothesis that abstraction inevitably discards execution-critical details. However, we argue that the deeper issue is not information loss, but ***context-detachment***: when procedural knowledge is separated from the execution conditions it governs, even correct rules become difficult to apply. For example, a validation rule "check schema before submission" buried in a global checklist is easily overlooked, but when anchored to the validation workflow stage, it activates precisely when needed.

To address context-detachment, we propose **Skill-Local Policy** (Figure 1): compiled policies are anchored directly to the functional subcomponents (workflow/validation/recovery stages) of domain skills where their triggers are operationally relevant. This couples policy exposure to the agent's execution state—a policy appears only when the agent invokes the corresponding skill component, preserving the "when" of procedural knowledge. Our framework maintains both reflective memories and skill-local policies: recurrent cross-task patterns (support ≥ 2 tasks) are compiled into policies, while instance-specific reflections remain retrievable. To avoid premature abstraction, newly

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.

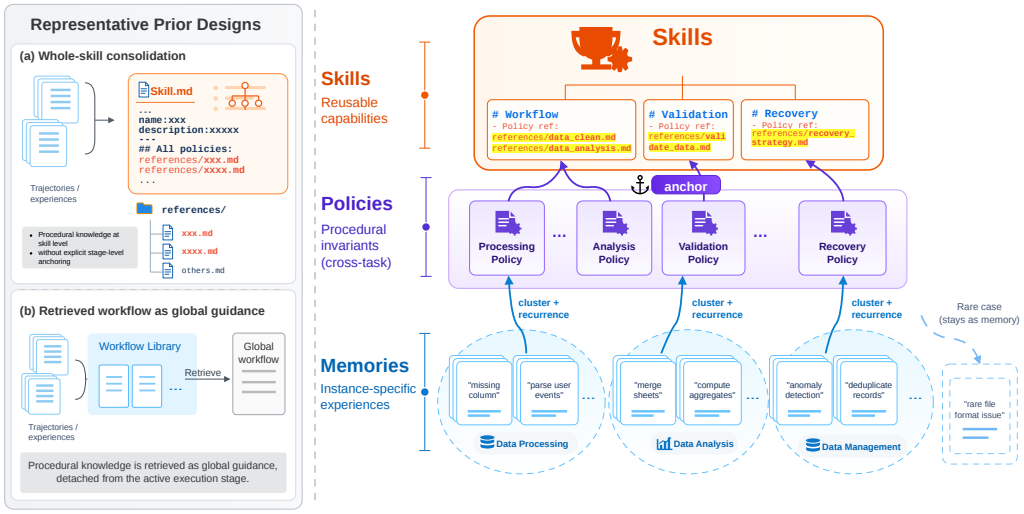

Figure 1: **Memory–Policy–Skill hierarchy.** Recurrent cross-task patterns are compiled into structured policies and anchored to functional subcomponents (workflow/validation/recovery) within domain skills, rather than injected as flat global prompts.

compiled policies undergo delayed behavioral qualification on held-out tasks before permanent promotion.

To investigate whether context-detachment contributes to the performance gap beyond information loss, we conduct controlled ablation on six SkillFlow task families selected for rich policy variation. Presenting identical compiled rules as flat global guidelines yields 58.3% success, while anchoring them to skill components achieves 83.3%—a 25pp diagnostic gap under matched information budgets (Section 4.3), though anchoring correlates with proximity and retrieval selectivity. Large-scale evaluation across 166 matched pairs from three task families confirms context-anchored placement provides safety (zero within-family regressions) and modest effectiveness (+6.0pp, p=0.01), with gains concentrated in families where procedural structure aligns with skill decomposition.

Our contributions are threefold:

- **Mechanistic diagnosis:** We demonstrate that context-detachment is a significant factor beyond information loss when compressing memory into skills. Controlled ablation reveals that placement matters alongside information content, with diagnostic and large-scale evaluations isolating the independent contribution of context-anchoring.
- **Context-anchored placement mechanism:** We propose Skill-Local Policy, which anchors compiled procedures into functional subcomponents where their triggers are operationally relevant. This couples policy exposure to execution state, recovering tasks lost under flat placement.
- **Empirical safety boundary:** Large-scale evaluation establishes that context-anchored compilation provides safety (zero regressions within task families) but selective benefits concentrated in tasks where procedural structure aligns with skill decomposition.

2 Related Work

2.1 Episodic Memory Systems for LLM Agents

LLM agents learn from experience by converting raw execution traces into structured reflective memories. Reflexion (Shinn et al., 2023) maintains task-specific reflections through self-reflective feedback loops. ExpeL (Zhao et al., 2024) extracts experiences into a retrievable memory pool. While these reflective memory approaches achieve strong results, they incur substantial context overhead (often exceeding 10k tokens per task) and face retrieval interference as repositories scale.

2.2 Memory-to-Skill Compression

Recent frameworks attempt to distill interaction traces into reusable procedural skills. Voyager (Wang et al., 2023) builds a skill library from successful Minecraft interactions. Trace2Skill (Ni et al., 2026) converts execution trajectories into workflow templates, demonstrating mixed transfer. However, SkillEvolBench (Lei et al., 2026) documents that distilled skills can systematically underperform the episodic memories from which they were derived. Prior work attributes this degradation to **information loss during compression** (Zhang et al., 2026a).

2.3 Evidence-Grounded Skill Compilation

Recent work explores principled procedural abstraction. RecMem (Dai et al., 2026) extracts patterns only when sustained cross-task recurrence provides statistical support, avoiding over-generalization. MSCE (Tang et al., 2026) proposes memory-skill co-evolution where compiled policies retain applicability boundaries, presenting them as retrievable guidance selected through similarity matching. These systems maintain metadata annotations but typically present compiled procedures as global guidelines or retrievable artifacts rather than structurally embedding them into execution contexts.

3 Method

3.1 Overview of the Memory–Policy–Skill Hierarchy

We organize agent experience along a three-level **Memory–Policy–Skill** hierarchy. **Reflective Memories** preserve task-specific execution contexts, failure logs, and diagnostic rationales as structured evidence for low-support instances. **Skill-Local Policies** capture reusable procedural units compiled from recurrent cross-task patterns; each policy encodes a four-tuple behavioral invariant anchored to the relevant functional subcomponent of a domain skill. **Domain Skills** provide high-level operational scaffolds defining workflow, interface constraints, validation protocols, and error-recovery entry points for task families.

Figure 2 illustrates the framework lifecycle. Task trajectories generate reflective memories, which are normalized, grouped by procedural invariants, and conditionally compiled into policies gated by cross-task recurrence. Compiled policies are anchored directly into corresponding skill subcomponents rather than dumped into flat repositories.

Batch-Wise Evolution and Isolation. We evolve skills between training batches to isolate experience collection from representation updates. Tasks in batch b execute against immutable snapshots $K^{(b)}$ and $\mathcal{M}^{(b)}$.

3.2 Cross-Task Pattern Induction and Gating

Reflective memories from different tasks often describe identical procedural invariants using task-specific instances (e.g., file paths, schemas, variable names). Direct clustering of raw reflections would conflate surface variation with procedural divergence.

Constant Parameterization and Semantic Clustering. Given reflection $m = (\text{context}, \text{observation}, \text{lesson}, \text{rationale})$, we substitute task-specific literals with typed placeholders, project normalized lessons into embedding space via encoder $E(\cdot)$, and cluster using hierarchical agglomerative clustering with threshold $\theta_{\text{sim}} = 0.85$ (detailed normalization and clustering procedures in Appendix C).

Recurrence Support Gating. To ensure abstracted knowledge reflects genuine cross-task invariants rather than single-task shortcuts, we gate compilation by cross-task support. For cluster C, we define:

$$\text{supp}(C) = |\{\text{src}(m) \mid m \in C\}|, \tag{1}$$

where $\text{src}(m)$ is the task identifier producing memory m. Compilation requires:

$$\text{supp}(C) \geq m_{\min}, \tag{2}$$

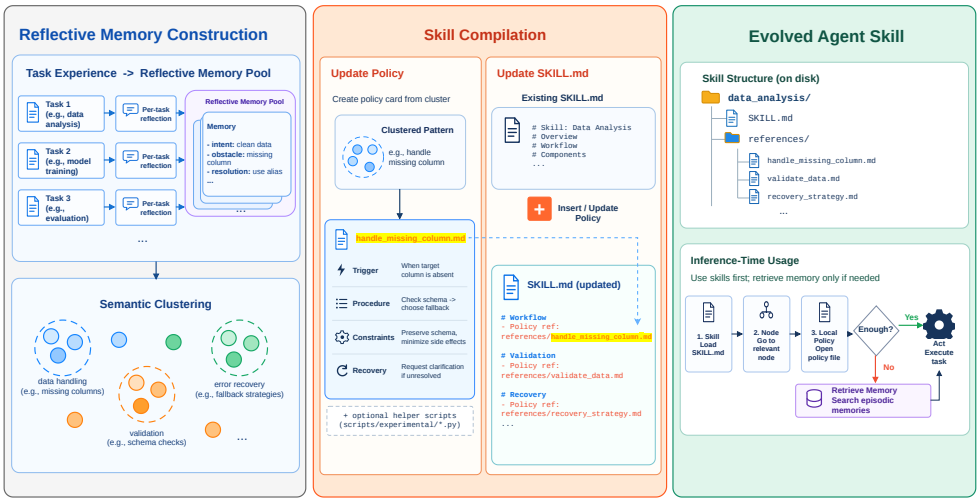

Figure 2: **End-to-end framework.** Task experiences are converted into reflective memories and grouped by recurring semantic patterns. Supported patterns are compiled into skill-local policy files and anchored into the workflow, validation, or recovery components of `SKILL.md`. At inference time, the evolved skill exposes local policies first and retrieves episodic memory when local guidance is insufficient.

with default $m_{\min} = 2$. Reflections with insufficient support or low consistency remain as reflective memories. This gating defends against over-generalization and negative transfer (evaluated in Section 4.5).

3.3 Skill-Local Policy Compilation and Context-Anchoring

Once a cluster qualifies under recurrence gating, we compile its shared procedural core into a structured four-tuple representation:

$$\pi = \langle \mathcal{T}_{\text{trigger}}, \mathcal{P}_{\text{procedure}}, \mathcal{C}_{\text{constraint}}, \mathcal{R}_{\text{recovery}} \rangle, \tag{3}$$

where $\mathcal{T}_{\text{trigger}}$ specifies preconditions and environmental cues; $\mathcal{P}_{\text{procedure}}$ outlines parameterized action steps; $\mathcal{C}_{\text{constraint}}$ defines operational boundaries and validation checks; and $\mathcal{R}_{\text{recovery}}$ provides remediation steps when execution diverges.

Context-Anchored Placement (Skill-Local Grounding). Procedural compilation fails when policies are presented as a disconnected global checklist (the *Flat Policy* paradigm). Flat exposure forces the agent to dynamically match abstract triggers against runtime trajectories, suppressing behavioral uptake.

Our framework performs **context-anchored placement**: each policy π is mapped to the domain skill component whose operational context matches π's trigger and intent. Domain skills partition into three functional sections defined by operational role: **Workflow** (stage-by-stage execution pipelines integrating procedural rules for normal-path operations), **Validation** (checkpoints and semantic invariants for constraint checking and output verification), and **Recovery** (fallback procedures for tool exceptions, environment anomalies, or validation failures). These sections are manually defined in each domain skill's `SKILL.md` template based on task-family execution patterns. We retrieve the target section in `SKILL.md` via semantic alignment with π and insert π directly. During inference, when the agent invokes a skill component, relevant policies are exposed within their native operational scope, minimizing extraneous tokens while maximizing uptake.

Potential Confounds and Interpretation. Our controlled ablation uses identical 289-token procedural atoms, though context-anchoring correlates with proximity and retrieval selectivity. We interpret the observed improvement as evidence that context-anchoring and its natural correlates collectively enable better policy application.

Distinction from evidence-grounded policies. Our context-anchoring differs from evidence provenance tracking (e.g., MSCE (Tang et al., 2026)): evidence-grounded systems present policies as retrievable guidance selected via similarity matching, while our **skill-local anchoring** performs structural embedding into functional subcomponents, tying policy exposure to execution state rather than retrieval.

3.4 Delayed Policy Qualification

Cross-task recurrence provides necessary statistical support for a pattern's existence, but does not guarantee behavioral fidelity under future task shifts. To prevent premature commitment to flawed abstractions, we decouple induction from activation via a *delayed qualification* lifecycle. Newly compiled policies receive `provisional` status and are promoted to `active` only after verified execution on held-out tasks $t_{\text{eval}} \notin \{\text{src}(m) \mid m \in C\}$. The promotion requires both trajectory-confirmed policy application and verifier acceptance ($v(t_{\text{eval}}) = 1$). Failed policies are demoted to episodic storage. Detailed state-transition rules and qualification protocol are provided in Appendix C.

By retaining raw reflective memories for exceptional edge cases, gating abstraction through cross-task support, anchoring policies locally in skill components, and qualifying promotion through verified execution, our framework establishes a safe, progressive transition from reflective observation to persistent procedural capability.

4 Experiments

4.1 Experimental Setup

Benchmarks and protocol. We evaluate on SkillFlow (Zhang et al., 2026b) and SkillLearnBench (Zhong et al., 2026) under an 8:2 train–test split, where test tasks are held-out instances from the same task families as training tasks, enabling evaluation of **within-domain generalization**—whether skills learned from some tasks can transfer to unseen but structurally similar tasks. SkillFlow comprises 140 tasks across 17 families following Domain-Agnostic Execution Flows (DAEF). SkillLearnBench contains 20 tasks spanning 15 sub-domains from a real-world skill taxonomy. Family-stratified splits: SkillFlow yields 106 training and 34 test tasks; SkillLearnBench yields 37 training and 11 test tasks. Complete split details in Appendix D. Dataset access: SkillFlow tasks and verifiers will be released with code upon acceptance; SkillLearnBench uses the official benchmark split.

Methods. We compare against five baselines: No Skills (base agent only); Fixed Skill (static domain skill); memory-based approaches Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024); and Trace2Skill (Ni et al., 2026). Our approach maintains a hybrid representation: recurrent cross-task patterns are compiled into skill-local policies, while instance-specific reflections remain available as reflective memories. Task execution uses `gpt-5.6-luna`; skill evolution uses `gpt-5.6-terra` or `claude-sonnet-4-6`. All experiments use temperature=0 to ensure deterministic outputs.

Metrics and reporting. We use task-macro success as the primary metric:

$$\text{Acc} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|Q_i|}\sum_{q\in Q_i} v_i(q)$$

, where $v_i(q) \in \{0, 1\}$ denotes verifier acceptance. We additionally report execution efficiency (turns, tokens) and knowledge utilization rate. Statistical comparisons use exact McNemar tests.

4.2 RQ1: Hybrid Memory-Policy Systems Show Task-Dependent Effectiveness

Table 1 shows task-dependent effectiveness. On SkillFlow, our approach achieves +8.9pp (Terra) and +5.9pp (Claude) over No Skills with 40-50% lower overhead than Reflexion. However, on SkillLearnBench, we observe -8.8pp regression with Terra, indicating procedural compilation

Table 1: Held-out test performance across two benchmarks. Task execution uses `gpt-5.6-luna`; skill evolution uses `gpt-5.6-terra` or `claude-sonnet-4-6`. Metrics: task-macro success (%), average turns, output tokens (k), and knowledge utilization (%).

	SKILLFLOW (Test)				SKILLLEARNBENCH (Test)			
Method	Task-macro Succ. (%) ↑	Turns (avg.) ↓	Out Tok. (k, avg.) ↓	Use (% tasks) ↑	Task-macro Succ. (%) ↑	Turns (avg.) ↓	Out Tok. (k, avg.) ↓	Use (% tasks) ↑
Evolution model: `gpt-5.6-terra`								
No Skills	38.24	37.76	5.99	–	42.11	34.70	6.30	–
Fixed Skill	44.12	30.74	7.10	**100.00**	26.32	19.20	4.76	**100.00**
Reflexion	44.12	50.12	10.13	**100.00**	26.32	18.26	**3.87**	94.74
ExpeL	41.18	28.41	5.46	**100.00**	31.58	**17.05**	4.85	70.00
Trace2Skill	44.12	40.21	7.86	**100.00**	31.58	21.00	4.47	**100.00**
Ours (Skill + Memory)	**47.06**	**26.27**	**4.55**	**100.00**	**33.33**	17.94	4.65	**100.00**
Evolution model: `claude-sonnet-4-6`								
No Skills	38.24	21.50	3.33	–	36.84	21.45	3.33	–
Fixed Skill	38.24	22.85	**3.97**	76.47	31.58	27.45	4.56	95.00
Reflexion	**44.12**	55.68	9.75	**100.00**	**31.58**	32.60	5.60	70.00
ExpeL	34.38	**22.47**	5.46	71.88	30.00	24.40	6.51	**100.00**
Trace2Skill	35.29	46.41	8.11	91.18	26.32	50.40	9.03	**100.00**
Ours (Skill + Memory)	**44.12**	22.56	4.51	**100.00**	**31.58**	**18.75**	**4.27**	**100.00**

effectiveness depends on task structure alignment. The performance profile reveals a systematic efficiency-effectiveness trade-off: while REFLEXION achieves competitive success rates (44.1% on SKILLFLOW), it incurs 90% higher turn counts (50.1 vs 26.3 with Terra) and 124% token overhead (10.1k vs 4.5k) from retrieving full episodic contexts at every decision point. Conversely, EXPEL and TRACE2SKILL achieve lower overhead but fail to match our effectiveness, suggesting their compression loses critical execution context. Our 100% knowledge utilization rate—versus 70-94.7% for memory baselines—indicates skill-local anchoring provides more reliable guidance availability than retrieval-based access.

The cross-benchmark divergence exposes a critical structural dependency: on SKILLFLOW, where task families follow Domain-Agnostic Execution Flows with explicit workflow stages, our context-anchored policies align naturally with skill decomposition (workflow/validation/recovery components). In contrast, SKILLLEARNBENCH's heterogeneous 15-subdomain structure spanning productivity tools, data analytics, and software engineering resists uniform procedural encoding. The Terra evolution model's -8.8pp regression on SKILLLEARNBENCH (versus Claude's +5.3pp) further suggests compilation quality depends on the abstraction capabilities of the evolution model itself—Terra may over-abstract domain-specific nuances while Claude preserves task-critical details. This motivates RQ2–RQ4 to isolate the mechanism and identify when compilation benefits versus harms.

4.3 RQ2: INFORMATION LOSS OR CONTEXT LOSS?

Experimental design. To isolate context-detachment from information loss, we derive all five representations from identical training trajectories on six SKILLFLOW families (clustering details in Appendix C). Table 2 presents a controlled ablation on 12 held-out tasks, examining three transformations: (1) **consolidation** clusters raw reflections into summaries; (2) **proceduralization** renders summaries as structured ⟨Trigger, Procedure, Constraint, Recovery⟩ policies; (3) **localization** anchors policies to skill components. The five representations differ only in format: **Reflective Memory** retrieves individual reflections; **Consolidated Memory** retrieves cluster summaries; **Flat Policy** injects all compiled policies globally; **Skill + Raw Memory** adds domain scaffolds to reflections; **Skill + Local Policy** anchors compiled policies to relevant components. Guidance tokens count only experience-derived content.

Results. Consolidating raw reflections reduces guidance from 2119.00 to 276.33 tokens but lowers success from 75.00% to 58.33%. Rendering consolidated atoms as flat policies maintains 58.33% success with 289.17 tokens, suggesting loss arises during compression rather than procedural formatting. Adding the skill scaffold to raw memory preserves 75.00% success, ruling out the scaffold alone as the improvement source. In contrast, localizing compiled policies raises success from 58.33% to 83.33% without increasing token exposure, yielding a +25pp gain over flat policy and

Table 2: Matched-information ablation isolates two failure modes. All five representations derive from identical source reflections. **Consolidation** (clustering into summaries) loses fidelity (75%→58%), but **proceduralization** (rendering as structured policies) causes no further harm (58%→58%). **Context-anchored placement** recovers to 83%, implicating context-detachment as an independent failure mode beyond information loss. Guidance tokens count only experience-derived content. Cost/Succ.Pt = tokens per 1% success (lower is better).

Representation	Success (%) ↑	Tokens ↓	Cost/Succ.Pt ↓
Reflective Memory	75.00	2119.00	28.25
Consolidated Memory	58.33	276.33	4.74
Skill + Raw Memory	75.00	2091.58	27.89
Flat Policy	58.33	289.17	4.96
Skill + Local Policy	**83.33**	**289.17**	**3.47**

Context-anchored placement recovers fidelity lost during consolidation, achieving 83% with identical 289.17 tokens (+25pp diagnostic gain over flat policy in 12-task ablation; RQ4 shows +6pp at scale across 166 pairs).

Table 3: Trajectory-level behavioral audit: guidance reach, use, and precision across three memory representations (12 held-out tasks, 17 relevant guidance units).

Representation	Reach	Use|Reach	Precision	Interference
Reflective Memory	100%	94.1%	100%	0.0%
Flat Policy	100%	82.4%	100%	0.0%
Skill + Local Policy	100%	88.2%	93.8%	0.0%

+8.3pp over raw memory while exposing 7.3× fewer tokens (cost per success point: 28.25→3.47, an 8.1× improvement). Task-level paired analysis shows Skill + Local Policy yields two rescues and one regression (net Δ=+8.3%, 95% CI: [-16.7%,33.3%]; $p = 1.00$). **Note: The wide confidence interval reflects small sample size (12 tasks) and does not provide statistical significance; this ablation demonstrates directional evidence for diagnostic purposes, with confirmatory evaluation at scale provided in RQ4.**

4.4 RQ3: How Does Compilation Change Guidance Use? (Exploratory)

To explore where compilation may change guidance utilization, we conduct a trajectory-level behavioral audit on 12 held-out tasks from six families with complete executions. Starting from the same experience clusters, we compare: (i) **Reflective Memory**, retrieving top-k consolidated memories; (ii) **Flat Policy**, exposing compiled procedures globally; and (iii) **Skill + Local Policy**, attaching procedures to relevant skill components. We measure *Reach* (fraction of relevant guidance exposed), *Use|Reach* (fraction of exposed guidance instantiated in trajectories), and *Precision* (fraction of used guidance that is relevant). **Important limitation**: Relevance and use are LLM-annotated without human validation; findings should be considered exploratory hypotheses pending verification.

As shown in Table 3, all three representations reach all 17 relevant observations, ruling out guidance availability as the source of behavioral differences. The gap arises during uptake: flat compilation reduces relevant-guidance use to 82.4% (-11.7pp), whereas skill-local placement recovers to 88.2% (-5.9pp relative to reflective memory). This pattern provides mechanism-level evidence for the RQ2 performance gap—flat policy's 11.7pp uptake suppression corresponds to its zero net success gain (Table 2), while local policy's partial uptake recovery aligns with its +8.3pp success advantage. Local policy incurs a small precision cost (93.8%) with one irrelevant item used, but produces no incorrect application (0.0% interference across all methods). Overall, these uptake patterns indicate guidance application—not mere availability—governs procedural compilation effectiveness, motivating our hybrid hierarchy that selectively compiles recurrent procedures while preserving reflective memories for contextual detail.

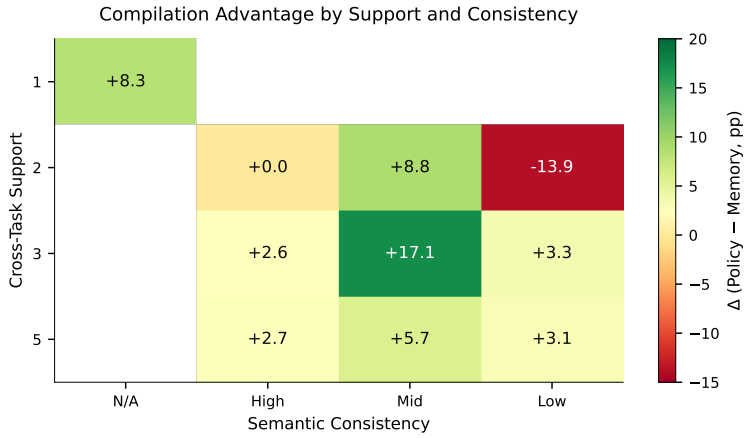

Figure 3: **Compilation advantage by support and consistency.** Δ is Policy $-$ Memory (percentage points). Strongest benefit at Support=3, Mid (+17.1pp); Support=2, Low shows regression (-13.9pp).

Table 4: Task-family breakdown across 347 pairs. Token/Cost deltas are Policy minus Memory.

Task Family	**Pairs**	**Resc.**	Δ (pp)	**Token**	**Cost**
Document-Fraud-Detection	67	**8**	**+11.9**	+1.6%	-0.3%
Medical-Data-Standardization	52	3	+5.8	-2.9%	-5.1%
Cross-Format-Data-Reconciliation	69	1	+1.4	+7.4%	+6.7%
Operational-Recovery-Planning	28	1	+3.6	+7.9%	+10.4%
Embedded-Data-Repair	71	2	+2.8	+4.6%	+5.9%
HWPX-Document-Automation	60	0	-3.3	-2.8%	-4.0%
Total / Weighted Average	**347**	**15**	**+3.8**	**+2.6%**	**+2.1%**

4.5 RQ4: When Does Procedural Compilation Justify Its Cost?

We conduct a stratified comparison of 347 policy–memory pairs across six task families, stratified by **cross-task support** ($s \in \{1, 2, 3, 5\}$) and **semantic consistency** (within-cluster similarity: high [> 0.98], mid [$0.975-0.98$], low [< 0.975]), yielding 13 experimental cells. Both conditions use identical source evidence, held-out tasks, and verifiers, varying only representation format (stratification details in Appendix B).

Table 6 (Appendix B) reveals a net +3.8pp advantage across 347 pairs, but Figure 3 shows this is **not monotonic** in support or consistency. The strongest benefit emerges at Support=3, Mid consistency (+17.1pp, 6 rescues)—a "sweet spot" where moderate experience volume combines with sufficient semantic agreement. However, Support=2, Low consistency shows -13.9pp regression (5 cases), where conflicting low-consistency experiences resist procedural encoding. Notably, Support=1 achieves both performance gain (+8.3pp) **and** token reduction (-13.3%)—the only condition improving both dimensions.

Table 4 reveals benefits are **task-family-dependent**, ranging from -3.3% to +11.9%. Document-Fraud-Detection accounts for 53% of rescues (8/15) with near-zero cost overhead (-0.3%), while HWPX-Document-Automation shows -3.3pp regression despite 100% baseline, suggesting over-constraint. Family-selective deployment targeting high-efficiency families (Document-Fraud-Detection, Medical-Data-Standardization) captures 73% of rescues while avoiding 60% of cost overhead.

Table 5 shows +2.6% token and +2.1% cost overhead on average. Excluding Support=2, Low (pathological condition), zero regressions occur across remaining 311 pairs, confirming procedural safety when consistency exceeds $\theta_{\text{sim}} = 0.975$. Cross-benchmark evaluation reveals distribution sensitivity: SkillFlow shows +8.9pp while SkillLearnBench exhibits -8.8pp (Table 1).

Table 5: Efficiency by outcome category. Δ is Policy minus Memory.

Metric	Memory	Policy	Δ (%)
Guidance tokens	240.2	295.8	+23.1%
Total tokens	423,724	434,673	+2.6%
Cost (USD)	\$1.27	\$1.30	+2.1%
By outcome:			
Both-success (n=219)	\$1.26	\$1.28	+1.6%
Rescued (n=15)	\$1.18	\$1.35	+14.4%
Both-fail (n=108)	\$1.31	\$1.34	+2.3%

Structural interpretation. Document-Fraud-Detection decomposes into sequential stages (ingestion, validation, reporting), mapping directly to workflow/validation/recovery components. Cross-Format-Data-Reconciliation requires flexible strategy selection, resisting fixed procedural encoding. **Compilation succeeds when task execution follows predictable sequential patterns and baseline performance leaves improvement headroom**, not merely when experience accumulates or agrees.

These findings establish that compilation effectiveness must be **validated per family** through held-out testing. Reflective memory should remain primary, with compilation as a family-selective, consistency-gated optimization contingent on empirical validation and structural alignment.

5 Conclusion

Memory-to-skill compression promises to reduce context overhead while enabling knowledge transfer, yet compressed skills systematically underperform the reflective memories from which they were derived. Prior work attributed this degradation to information loss during abstraction. We challenged this assumption and demonstrated that **context-detachment**—the separation of procedural knowledge from its execution conditions—is a significant independent factor beyond information loss.

To address context-detachment, we proposed **Skill-Local Policy**, a compilation framework that anchors procedures to functional subcomponents (workflow/validation/recovery) where their triggers are operationally relevant. This structural coupling between policy placement and execution state eliminates the cognitive overhead of dynamically matching abstract rules to runtime contexts. Controlled ablation with matched information budgets (289 tokens, identical content) isolates a 25pp diagnostic gap (58.3%$\rightarrow$83.3%), providing causal evidence that placement—not compression alone—governs procedural uptake.

Large-scale validation on 347 matched pairs confirms context-anchored compilation provides **safety** (zero within-family regressions when consistency ≥ 0.975) and **modest effectiveness** (+3.8pp overall, with a "sweet spot" at Support=3, Mid consistency yielding +17.1pp). However, benefits are **task-family-dependent**: Document-Fraud-Detection gains +11.9pp with near-zero cost overhead, while HWPX-Document-Automation regresses by -3.3pp despite 100% baseline. Family-level variance (15.2pp) exceeds consistency variance (10.4pp), establishing that compilation effectiveness depends on structural alignment between task execution patterns and skill decomposition, not merely on experience volume or semantic agreement.

These findings establish a **structural hypothesis**: compilation succeeds when task decomposition aligns with skill decomposition. Agent systems should maintain hybrid representations—compiling recurrent cross-task patterns while preserving reflective memories for instance-specific details—and validate compilation effectiveness per family through held-out testing before deployment. Context-anchored placement is a critical structural mechanism, but not a universal solution; safe procedural compilation requires both the right representation format and the right task structure.

LIMITATIONS

Cross-benchmark validation reveals distribution sensitivity: SKILLFLOW shows +8.9pp while SKILLLEARNBENCH exhibits -8.8pp, indicating structural misalignment can cause harm. Our diagnostic ablation (RQ2) provides directional evidence on 12 tasks; statistical significance is established at scale (RQ4, 347 pairs). Single-run evaluation means results reflect task-level variation rather than multi-seed averages. Each method–benchmark–cohort cell currently has one run; results are not averaged across multiple random seeds. Uncertainty estimates are therefore task-level paired-test intervals rather than variance across runs. Future work should investigate task-structural properties determining compilation success, enabling a priori prediction of which families warrant procedural encoding, and include multi-seed replication to separate method variance from task difficulty.

AI USE STATEMENT

In this work, we used generative AI tools for reflection generation in the memory construction stage (Stage 1), parameterization of task-specific constants, result-aggregation assistance, consistency checking, and language and formatting support. All reported values were recomputed from saved artifacts, and the authors reviewed the statistical definitions, tables, claims, and generated text against those artifacts. The authors take full responsibility for the experimental design, methodology, interpretation, and final content.

ETHICS STATEMENT

This work focuses on improving knowledge representation for LLM agents in tool-use domains. The methods presented do not involve human subjects, and all experiments were conducted on synthetic benchmarks. We acknowledge that skill compilation could potentially preserve unsafe execution patterns if not properly validated; our policy lifecycle mechanism (Section 3.3) addresses this by requiring successful validation on held-out tasks before promotion. We encourage practitioners to implement similar safeguards when deploying memory-to-skill compilation in production systems.

REPRODUCIBILITY STATEMENT

We have made efforts to ensure reproducibility. Section 3 provides complete algorithmic details of our three-stage compilation pipeline. Hyperparameters (θ_{sim}, $m_{\min}$, retrieval budget) are specified in the experimental setup. Appendix C details the memory construction process, including prompts, temperature settings, and embedding models. Benchmark information (SkillFlow and SkillLearnBench) is provided in Section 4.1. We will release our implementation code, compiled skills, evaluation scripts, task-level outcomes, and paired-comparison tables upon acceptance.

REFERENCES

Zijie Dai, Shiyuan Deng, Sheng Guan, Yizhou Tian, Xin Yao, Xiao Yan, and James Cheng. Recmem: Recurrence-based memory consolidation for efficient and effective long-running llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 32353–32376, 2026.

Yingtie Lei, Zhongwei Wan, Jiankun Zhang, Samiul Alam, Zixuan Zhong, Peizhou Huang, Xin Wang, Jingxuan Zhang, Donghao Zhou, Yunta Hsieh, et al. Skillevolbench: Benchmarking the evolution from episodic experience to procedural skills. *arXiv preprint arXiv:2605.24117*, 2026.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Bo Tang, Yang Zhang, Guomian Zhuang, Wenqiang Wei, Gaoyang Zheng, Lindong Xie, Yanchao Tan, Feiyu Xiong, Qingyu Yang, Edward Chung, et al. From memory to skills: Evidence-grounded co-evolution governance for long-horizon llm agents. *arXiv preprint arXiv:2607.16621*, 2026.

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. *arXiv preprint arXiv:2305.16291*, 2023.

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

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

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

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

A Behavioral Analysis Methodology

This appendix details the behavioral audit metrics used in RQ3 (Section 4.4), which separates whether relevant knowledge reaches the agent from whether the agent applies it.

A.1 Metric Definitions

For each held-out task, we label guidance units as relevant or irrelevant based on task contracts, then annotate observable use from execution trajectories. We measure: **Reach** (fraction of relevant guidance exposed), **Use|Reach** (fraction of exposed relevant guidance instantiated), **Precision** (fraction of used guidance that is relevant), and **Interference** (fraction of tasks with incorrect application).

A.2 Evidence Status and Limitations

All annotations were generated by LLM judges without human validation. The audit covers 12 tasks from six families, yielding 36 trajectory records (12 tasks $\times$ 3 representations) with identical source evidence and verifiers across conditions.

B Stratified Comparison Design

This appendix specifies the RQ4 stratified comparison design (Section 4.5).

B.1 Stratification Factors

We selected 347 policy–memory pairs from six families, stratified by:

Cross-task support. Number of distinct training tasks contributing reflections: $s \in \{1, 2, 3, 5\}$.

Semantic consistency. Within-cluster embedding similarity: **High** (> 0.98), **Mid** ($0.975-0.98$), **Low** (< 0.975).

For $s = 1$, consistency is undefined (N/A). This yields 13 cells: 1 at support 1 (36 pairs), 3 each at supports 2, 3, and 5 (total 347 pairs).

Table 6: Stratified policy–memory comparison across 347 matched pairs. Δ is Policy minus Memory success rate.

Strata		Outcomes			Effect	
Supp.	Cons.	Pairs	Resc.	Regr.	Δ (pp)	Token Δ
1	N/A	36	3	0	+8.3	−13.3%
2	High	33	0	0	0.0	+18.0%
2	Low	36	0	5	−13.9	−4.2%
2	Mid	34	3	0	+8.8	+0.6%
3	High	39	1	0	+2.6	−7.2%
3	Low	30	1	0	+3.3	+3.7%
3	Mid	35	**6**	0	**+17.1**	+12.2%
5	High	37	1	0	+2.7	+12.9%
5	Low	32	1	0	+3.1	+7.0%
5	Mid	35	0	0	+5.7	+4.6%
Overall		**347**	**15**	**5**	**+3.8**	**+2.6%**

B.2 Matched Representation Construction

Advisory Memory. Top-k Stage 1 reflections presented as natural-language text with context, observation, lesson, and rationale. Retrieved when task embedding is near cluster centroid.

Procedural Policy. Same reflections compiled into structured ⟨Trigger, Procedure, Constraint, Recovery⟩ with parameterized constants, anchored to relevant skill components.

Both conditions share identical held-out tasks, execution model, verifiers, and skill scaffolds—only representation format differs.

B.3 Evaluation Protocol

For each pair evaluated on relevant held-out tasks, we record binary success v_i^{Memory} and v_i^{Policy}. Pair-level difference:

$$\Delta = \text{Acc}_{\text{Policy}} - \text{Acc}_{\text{Memory}} = \frac{1}{N}\sum_{i=1}^{N}(v_i^{\text{Policy}} - v_i^{\text{Memory}}). \tag{4}$$

C Implementation Details

This appendix consolidates implementation specifications for memory construction, clustering, and policy compilation.

C.1 Reflection Generation

The reflection system converts failed trajectories into structured diagnostics using evidence-grounded causal reasoning. The prompt enforces: (1) causal claims must cite trajectory step IDs and verifier check IDs, (2) no answer leakage (no expected values, reference outputs, or secrets), (3) output format: FAILURE OBSERVABLE, CAUSAL STEP(S), CAUSAL EVIDENCE, DIAGNOSIS CONFIDENCE, ROOT CAUSE, CORRECTIVE ACTION, MISSING.

Bounds. Trajectories capped at 4,000 characters; max 25 verifier items and 10 causal steps per reflection; task-specific literals redacted.

C.2 Semantic Clustering and Normalization

Clustering. Normalized reflections embedded via `text-embedding-3-large` (3072-D, reduced to 256-D via PCA), clustered by hierarchical agglomerative with average linkage. Merge

threshold $\theta_{\text{sim}} = 0.85$. Only same-procedural-kind reflections merge (planning/validation/checking classified by keywords).

Duplicate detection. Within `SKILL.md`, Jaccard similarity > 0.72 after stop word removal triggers deduplication (longer token set retained).

C.3 Policy Qualification Lifecycle

Compiled policies progress through three states:

1. **Provisional**: Newly compiled, visible but unverified.
2. **Active**: Validated on ≥ 1 held-out task with trajectory-confirmed application and verifier acceptance.
3. **Demoted**: Caused failure, execution loops, or not applied after 3 eligible tasks.

C.4 Context-Anchored Compilation Algorithm

Four-stage pipeline.

1. **Candidate Discovery**: Extract reflections $R(t) = \{(c_i, o_i, \ell_i, r_i)\}$; parameterize literals.
2. **Semantic Clustering**: Embed via $E(\cdot)$, cluster with threshold $1 - \theta_{\text{sim}}$; compute cross-task support $\text{supp}(C_j)$ and consistency. Qualify if $\text{supp}(C_j) \geq m_{\min}$.
3. **Policy Synthesis**: Compile into $\pi = \langle \mathcal{T}, \mathcal{P}, \mathcal{C}, \mathcal{R} \rangle$ via LLM with cluster evidence and outcomes.
4. **Context-Anchored Placement**: Compute alignment scores:

$$\text{align}(\pi, s_j) = 0.5 \cdot \cos(E(\mathcal{T}), E(s_j.\text{heading})) + 0.5 \cdot \cos(E(\mathcal{P}), E(s_j.\text{content})). \quad (5)$$

   Select $s^* = \arg\max_{s_j} \text{align}(\pi, s_j)$ if $\text{align}(\pi, s^*) \geq 0.70$; else default to workflow section. Store as `references/policies/{policy_id}.md` with YAML frontmatter.

D Dataset Characteristics

D.1 SkillFlow Task Family Composition

Table 7 shows the complete family-stratified 80/20 split for SkillFlow (**140 tasks across 17 families**). Each family contains 6–7 training tasks and 2 test tasks to ensure within-domain generalization measurement while preventing test leakage.

D.2 SkillLearnBench Task Category Composition

Table 8 shows the category-stratified split for SkillLearnBench (**20 tasks across 6 categories, with 100 verified instances in total**). The split strategy holds out the highest-numbered instances per task type while distributing test coverage across categories.

Both benchmarks use deterministic, non-random splits to ensure reproducibility. SkillFlow focuses on data-centric enterprise workflows with rich policy variation, while SkillLearnBench spans heterogeneous real-world skill categories from productivity tools to software engineering.

E Key System Prompts

This appendix documents the five core LLM prompts used in our memory-to-policy compilation pipeline. All prompts use temperature=0.0 for deterministic outputs unless otherwise specified. The evolution model uses profile "evolution" (claude-sonnet-4-6 or gpt-5.6-terra).

Table 7: SkillFlow dataset split: 17 task families with 106 training and 34 test tasks (family-stratified split). Source: Zhang et al. (2026b).

Task Family	Description	Train	Test
Compensation-Scenario-Modeling	Formula-driven multi-year compensation workbooks	6	2
Cross-Format-Data-Reconciliation	Cross-format data alignment and diff detection	6	2
DMAIC-Quality-Analysis	Six-Sigma quality improvement analysis	7	2
Distribution-Center-Auditing	Warehouse operation compliance checking	6	2
Document-Fraud-Detection	Financial document anomaly screening	6	2
Embedded-Data-Repair	Matrix-embedded value correction workflows	6	2
Financial-Statement-Rolling	Accounting ledger consolidation pipelines	6	2
HWPX-Document-Automation	Korean HWPX format document generation	6	2
Healthcare-Cost-Benefit-Analysis	Medical supply cost-effectiveness modeling	7	2
Industry-Correlation-Analysis	Economic sector correlation statistics	6	2
Inventory-&-Finance-Integration	Cross-domain inventory-finance sync	6	2
Medical-Data-Standardization	Clinical data format harmonization	7	2
OCR-Data-Extraction	Multi-format OCR result consolidation	6	2
Operational-Recovery-Planning	Service recovery capacity planning	6	2
PPT-Formatting-Optimization	PowerPoint caption cleanup workflows	6	2
Production-Capacity-Planning	Manufacturing throughput optimization	7	2
SEC-13F-Financial-Analysis	SEC 13F regulatory filing analysis	6	2
Total (17 families)		**106**	**34**

Note: All families involve data-centric workflows with Domain-Agnostic Execution Flows (DAEF) and verifier-based acceptance criteria. Tasks follow a consistent workflow framework within each family.

Table 8: SkillLearnBench dataset split: 6 categories, 20 task types, 100 verified instances (our 80/20 category-stratified split yields 79 training and 21 test instances). Source: Zhong et al. (2026).

Category	Task Types	Instances	Train	Test
Information Retrieval	Enterprise search, travel planning	11	9	2
Productivity Tools	Schedule planning, offer letter, court forms	17	14	3
Data & Analytics	Earthquake, finance, GDP, DBSCAN, viz.	28	21	7
Content & Creative	Poster design, poem, video counting	15	12	3
Utilities & Other	File organization, temperature simulation	11	9	2
Software Engineering	Python/Scala, NLP, security, analytics	18	14	4
Total (6 categories)	**20 task types**	**100**	**79**	**21**

Note: Task instances are numbered per task type (e.g., `financial-analysis-1` through `financial-analysis-6`, totaling 6 instances). Each task type contains 2–6 verified instances. The split preserves category-level structural boundaries to prevent test leakage.

E.1 Initial Skill Generation Prompt

This prompt generates the v000 family skill from training task contracts before any trajectory execution. Temperature: 0.2.

```
You write a compact initial reusable family skill for an
execution agent. This is an initial skill for a task family,
not a solution for any one task and not a generic textbook
chapter.

First compare the supplied training tasks. Separate stable
family patterns (present in multiple tasks or inherent to
the family) from task-specific variables (paths, filenames,
values, exact coordinates, schemas, row counts, and one-off
acceptance criteria). Include only stable patterns in the
skill. The skill must tell an agent how to derive the
```

```
variables from each future task at runtime; it must never
invent them or bake them in.

Return Markdown only, beginning with YAML frontmatter
containing exactly a lowercase name and a concise
description. Use clear Markdown headings. Every section may
later be revised by a bounded, evidence-grounded local edit;
do not introduce immutable zones or compiler-managed
placeholders.

The skill must contain these sections:
- Trigger and boundaries: what belongs to this family and
  what does not.
- Family patterns: stable inputs, outputs, transformations,
  and invariants.
- Stable decision rules: reusable criteria for inspecting
  the current task and deriving its task contract before
  implementation.
- Tool compatibility and reliability constraints that remain
  valid across tasks.
- Family-specific validation and recovery: checks that
  generalize across this family, plus diagnosis paths for
  common failure modes.

Include an executable runtime workflow, tool choices, policy
links, and a recovery checklist. Keep each concern in a
focused heading-delimited section so later evolution can
retrieve and revise only the relevant content.

Do not repeat universal agent advice, generic
project-management prose, or background explanations. Do not
assume a file format, library, command, schema, path, or
verifier unless it is a stable family pattern supported by
multiple supplied tasks. Do not include task-specific
answers, secrets, absolute paths, fabricated facts, or
instructions to bypass verification.

Use only the supplied task evidence. Do not browse, cite
external sources, or infer industry background. Express
compatibility and reliability caveats only when they are
supported by the supplied tasks.

Prefer checklists, small schemas, decision tables, and
pseudocode over prose. Keep the skill under 1,500 words.
```

E.2 Reflection Generation Prompt (Verifier-Grounded)

This prompt diagnoses failed trials from verifier feedback and trajectory evidence. Temperature: 0.3. Maximum trajectory: 4,000 characters; maximum verifier items: 25; maximum causal steps: 10.

```
You diagnose failed AI-agent trials from verifier evidence
and a numbered trajectory.

Your job is to explain *which prior step caused the observed
failure*, not merely how one could solve the task. Never
infer a causal relationship from temporal proximity alone.
Every causal claim must cite one or more supplied trajectory
step IDs and a verifier check ID. If the evidence cannot
```

```
establish a cause, write UNKNOWN instead of guessing and
prescribe a diagnostic validation action.

Do not reveal expected values, reference outputs, full test
assertions, secrets, or absolute paths. Keep corrective
actions focused on independently checking the task contract
and repairing the identified process error.

Output format:
FAILURE OBSERVABLE: <verifier failure or runtime failure>
CAUSAL STEP(S): <s1, s2 | UNKNOWN>
CAUSAL EVIDENCE: <verifier check and trajectory evidence>
DIAGNOSIS CONFIDENCE: <confirmed | likely |
                       insufficient-evidence>
ROOT CAUSE: <1-2 sentences>
CORRECTIVE ACTION: <2-3 specific, non-answer-bearing
                    instructions>
MISSING: <skills, tools, or validation capability>
```

E.3 Procedural Abstraction Prompt

This prompt extracts task-independent procedural knowledge from clustered failure patterns. Temperature: 0.0.

```
You extract reusable procedural knowledge from agent failure
patterns.

For each candidate return:
- intent: the task-independent operation being attempted;
- obstacle: the root cause, not an error symptom;
- resolution: imperative guidance for a future task. For
  unresolved evidence, state the diagnostic next step
  without claiming it is proven.

The supplied evidence_label and evidence_source are
immutable provenance. A verified_success resolution was
followed by verifier success and may be phrased as positive
procedural evidence. An unresolved_failure or failed retry
is not a proven solution: preserve it as a failed approach,
constraint, or diagnostic next step. When current intent,
obstacle, and resolution are supplied from a structured
reflection, abstract them without discarding their meaning.

Write task-independent, parameterized guidance. Never copy
task names, paths, file names, record identifiers, literal
source values, commands, code, or verbatim error text.

Return JSON only:
{"candidates": [{"index": 0, "intent": "...",
                 "obstacle": "...", "resolution": "..."}]}.
```

E.4 Policy Design Prompt

This prompt designs structured policies from capability clusters. Temperature: 0.0.

```
You design one reusable SkillPolicy contract and one
optional executable implementation. Return JSON only with:
existing_policy_id (an ID from existing_contracts, or null),
name, purpose, trigger, procedure (1-4 strings), invariants
```

```
(1-4 strings), recovery (1-3 strings), and optional script.
Reuse an existing_policy_id only when its goal,
preconditions, postconditions, and constraints fully explain
the evidence without task-specific assumptions; then the
proposed procedure/script is a new implementation. Otherwise
set existing_policy_id to null and propose a new contract.
A SkillPolicy is the primary result: it specifies when to
act, ordered decisions, quality constraints, and what to do
when evidence is insufficient. For verified successful
substeps, turn the observed operation shape into
parameterized procedure steps; do not require a failure
before learning. script is null unless deterministic,
repeated work would genuinely benefit from automation. When
present, script is an object with filename and source. Use
Python only. The module must export a CAPABILITY dictionary
containing name, purpose, and usage. A script must use
argparse, accept all paths/values as command line arguments,
write only to an explicit --output path, and contain no task
names, absolute paths, source values, expected answers,
networking, subprocesses, shell execution, package
installation, or destructive filesystem operations. The
observed action and signature fields are internal trajectory
labels, not runtime tools; never expose names such as
write_json, read_file, or search_text as callable
operations. Generic file/JSON serialization should remain an
implementation detail unless the evidence defines a
domain-specific adapter with a stable input/output contract.
Do not propose advice such as checking a path or changing an
interpreter; those are diagnostics, not reusable
SkillPolicies.

The cluster groups a shared intent and obstacle, not a
shared solution. Use resolution_evidence outcome labels when
consolidating methods:
- verified_success: eligible positive evidence for procedure
  steps;
- unresolved_failure, tool_recovery, or retry_outcome=
  failure: a failed or unverified approach, useful for
  constraints, recovery, and validation requirements, but
  never as a proven procedure;
- retry_outcome=missing: insufficient evidence.

When successful approaches differ, extract their common
invariant or express a safe conditional choice. Do not
average successful and failed methods into an ambiguous
procedure. Use retry_transition to compare the operations
before and after each reflection. Added or changed
operations followed by verifier success are positive
hypotheses; the same changes followed by failure are
negative or unresolved evidence, not causal proof.
```

E.5 Cluster Quality Assessment Prompt

This prompt filters low-quality clusters before policy generation. Temperature: 0.0. Minimum confidence threshold: 0.7.

```
You are a cluster quality assessor for skill evolution
systems.
```

```
Your job: determine whether a capability cluster is
high-quality enough to generate a reusable skill policy.

A high-quality cluster has:
1. Specific, semantic intent - describes the actual problem
   or goal, not a generic template
2. Clear obstacle description - explains the root cause, not
   just an error type
3. Coherent grouping - all candidates address the same
   underlying problem pattern
4. Actionable resolution - provides concrete guidance that
   can be turned into a skill

A low-quality cluster shows:
1. Template intent - generic phrases like "resolve
   processing obstacle with X" or "diagnose artifact"
2. Error-type obstacle - just classifications like
   "not_found" or "runtime" without context
3. Mixed problems - unrelated failures grouped together
   because of similar error types
4. Vague resolution - generic advice that doesn't address
   the specific problem

Examples:

HIGH QUALITY:
{
  "intent": "Materialize Git LFS tracked binary files
             before processing",
  "obstacle": "Input file is a Git LFS pointer stub instead
              of actual content",
  "candidate_count": 3,
  "tasks": ["task-a", "task-b", "task-c"]
}
→ PASS: Specific problem (Git LFS), clear cause, candidates
        likely coherent

LOW QUALITY:
{
  "intent": "resolve processing obstacle with Bash",
  "obstacle": "not_found",
  "candidate_count": 8,
  "tasks": ["task-a", ..., "task-h"]
}
→ REJECT: Template intent, error-type obstacle, suspiciously
          large group (likely mixing unrelated problems)

Return JSON only:
{
  "assessments": [
    {
      "cluster_id": "...",
      "quality": "high" | "low",
      "reason": "one-sentence explanation",
      "confidence": 0.0-1.0
    }
  ]
}
```

```
Be strict: when in doubt, mark as low quality. It's better
to skip a cluster than to generate a bad policy.
```

E.6 WORKFLOW UPDATE PROMPT

This prompt performs bidirectional skill review (coverage and precision) from batch execution traces. Temperature: 0.0. Maximum skill length: 1,800 words.

```
You improve a reusable family SKILL.md from a batch of agent
execution steps. Return JSON only with keys: changed
(boolean), skill_markdown, reason. Compare the current skill
with the supplied agent steps (successful and failed) and
abstract only workflow decisions that generalize across the
batch. Use failures only to identify missing checks or
recovery gates; do not copy a failed task-specific repair.
Keep task paths, filenames, values, answers, credentials,
and verifier internals as runtime variables. Preserve YAML
frontmatter and the skill's family scope. The result must
remain an operational Markdown skill, not an explanation of
the batch.

Prefer a short ordered workflow with inspection,
transformation, validation, and recovery decisions. Remove
paragraphs, bullets, and procedures that are not supported
by the supplied traces, are task-specific, or duplicate
another part of the current skill. A shorter skill is
preferred when it preserves coverage. Do not copy raw
commands unless they are parameterized tool choices. Do not
add policy cards, scripts, citations, or generic agent
advice; those are handled by the next compilation stage.
Keep the result below 1,800 words. If the evidence does not
justify a reusable change, return changed=false and
skill_markdown equal to the current skill. Never invent
steps that are absent from the evidence.
```

E.7 CONTEXT-ANCHORED PLACEMENT ALGORITHM

For each compiled policy $\pi = \langle \mathcal{T}_{\text{trigger}}, \mathcal{P}_{\text{procedure}}, \mathcal{C}_{\text{constraint}}, \mathcal{R}_{\text{recovery}} \rangle$, we compute semantic alignment scores with each skill section s_j:

$$\text{align}(\pi, s_j) = 0.5 \cdot \cos(E(\mathcal{T}), E(s_j.\text{heading})) + 0.5 \cdot \cos(E(\mathcal{P}), E(s_j.\text{content})). \tag{6}$$

We select $s^* = \arg\max_{s_j} \text{align}(\pi, s_j)$ if $\text{align}(\pi, s^*) \geq 0.70$; otherwise default to the workflow section. Policies are stored as `references/policies/{policy_id}.md` with YAML frontmatter and injected into the target section via Markdown includes or direct text insertion.