# APEX-EM: Non-Parametric Online Learning for Autonomous Agents via Structured Procedural-Episodic Experience Replay

Anonymous ACL submission

## Abstract

LLM-based agents are stateless at the procedural level: each task triggers full reasoning from scratch, even for previously solved problems. Experience replay eliminated this inefficiency for reinforcement learning agents decades ago. We introduce **APEX-EM**, a structured experience memory that accumulates complete procedural-episodic traces in a typed knowledge graph and retrieves them via hybrid semantic, structural-signature, and graph-traversal search, enabling improvement through deployment without weight modification. On five benchmarks, APEX-EM achieves 83.3% on BigCodeBench (+29.4pp over baseline), 91.2% on KGQAGen-10k (+49.9pp, approaching oracle-retrieval performance), 64.0% CSR on HLE, and 99.96% on ALFWorld with 100% transfer to unseen tasks; same-backbone controlled comparisons confirm gains are framework-driven: +7.6pp on BCB with GPT-4o (3.3× MemRL's +2.3pp) and +2.7pp over MemRL on HLE under near-identical baselines, with memory benefits additive across three backbone scales (+27–32pp). Analysis reveals that no single memory mechanism dominates: rich teacher feedback is negligible for code generation but critical for structured queries (+10.3pp), structural signatures enable 3.3× larger transfer effects than semantic-only retrieval, and within-epoch iteration compensates when rich evaluation is unavailable. These findings argue for modular memory architectures composed per-domain.

## 1 Introduction

Large language model (LLM)-based autonomous agents excel at multi-step reasoning, code generation, and tool use (Yao et al., 2023; Wei et al., 2022). Yet unlike reinforcement learning agents—which maintain experience replay buffers storing (state, action, reward, next_state) tuples for training stability (Lin, 1992)—LLM agents are fundamentally *stateless at the procedural level*: each task invocation triggers the full reasoning pipeline from scratch, even when a near-identical task was successfully completed moments earlier.

MemRL (Zhang et al., 2026b) represents the most significant advance toward closing this gap, organizing memory into Intent-Experience-Utility triplets with learned Q-values and demonstrating that non-parametric experience accumulation can match or exceed methods requiring gradient updates. Three open challenges remain beyond MemRL's scope: experience representations that preserve procedural structure (planning steps, iteration history, error-outcome links) rather than LLM-generated summaries; explicit dual-outcome indexing that separates *what succeeded* from *what failed* rather than encoding both implicitly through a scalar Q-value; and retrieval mechanisms that transfer across domains sharing operational structure but no lexical overlap, precisely the regime where embedding similarity degrades (intra-HLE similarity of 0.186 (Zhang et al., 2026b)).

We introduce **APEX-EM**, a framework that extends the non-parametric experience-driven paradigm. "Non-parametric" here means that no model weights are updated during deployment; the experience memory itself is the sole learned component, growing monotonically as the agent solves tasks, analogous to how a nearest-neighbor classifier improves as its reference set expands. Our contributions form a logical chain:

1. We formalize *what* an experience should encode: a **Procedural Knowledge Graph (PKG)** storing complete procedural-episodic traces as replayable units, each tagged with a *structural signature*, an abstract operation sequence (e.g., [entity_resolution → temporal_filter → aggregation]) enabling retrieval of procedurally analogous plans across surface-dissimilar domains.

2. We define *how* experiences are generated

and committed: the **PRGII workflow** (Plan-Retrieve-Generate-Iterate-Ingest), where a Teacher model provides multi-dimensional feedback (correctness, completeness, efficiency, error categorization) and a quality gate determines what enters the memory.

3. We show *how* stored experiences are retrieved and applied: a **dual-outcome Experience Memory** with hybrid retrieval combining semantic search, structural signature matching, and graph traversal. Successful plans provide reusable procedure templates; failed plans provide structured error annotations (error class, root cause, recovery procedure), implementing the procedural equivalent of hindsight experience replay (Andrychowicz et al., 2017).

We evaluate on five benchmarks (BigCodeBench (Zhuo et al., 2025), KGQAGen-10k (Zhang et al., 2025a), HLE (Phan et al., 2025), Lifelong Agent Bench (Zheng et al., 2025), and ALFWorld (Shridhar et al., 2021)) using Claude Sonnet 4.5 and Opus 4.6 as primary backbones, with same-backbone comparisons (GPT-4o, GPT-4o-mini, GPT-5-mini) to control for model capability differences.

## 2 Related Work

**Memory for LLM Agents.** Reflexion (Shinn et al., 2023) stores verbal reflections in episodic buffers; Voyager (Wang et al., 2024) builds verified code snippet libraries; MemGPT (Packer et al., 2023) introduces OS-inspired memory hierarchies; Mem0 (Chhikara et al., 2025) uses entity-centric relational graphs; MemP (Fang et al., 2025) distills procedural knowledge. MemRL (Zhang et al., 2026b) advances beyond these by assigning learned Q-values to Intent-Experience-Utility triplets and re-ranking by composite similarity-utility scores. All store either unstructured reflections, code fragments, or flat summaries; none provides structured procedural traces with dual-outcome indexing and cross-domain structural retrieval. Retrieval-augmented generation (RAG) systems (Lewis et al., 2020) retrieve documents for grounding but do not accumulate execution experience; classical plan libraries store reusable plans but lack LLM-based adaptation and structural signature matching.

**Experience Replay & Online Learning.** APEX-EM is conceptually an experience replay buffer (Lin, 1992) for LLM agents, but operates through in-context learning rather than gradient descent, stores high-level procedural plans rather than low-level transitions, and retrieves semantically and structurally rather than randomly. This positions it as a form of non-parametric experience-driven improvement (Shalev-Shwartz, 2012): the agent improves during deployment as its reference set grows, analogous to a nearest-neighbor classifier, though without formal regret guarantees. MemRL is the closest prior work but stores LLM-summarized reflections rather than full execution traces, and its transfer degrades when semantic similarity is low.

**Case-Based Reasoning and Plan Abstraction.** The CBR cycle (Aamodt and Plaza, 1994)—Retrieve, Reuse, Revise, Retain—bears structural resemblance to PRGII, and plan abstraction has a long history in classical AI planning: derivational analogy (Veloso, 1994) reuses abstract proof structures across problems, and hierarchical task networks decompose tasks into reusable operator sequences (Erol et al., 1994). APEX-EM's structural signatures draw on this lineage. The key distinctions are: (1) extraction is performed by an LLM rather than hand-crafted rules, enabling open-vocabulary operation discovery without a predefined taxonomy; (2) dual-outcome indexing stores both successes and failures with structured error annotations (CBR typically retains only successes); and (3) retrieval combines structural matching with semantic and graph-based channels rather than relying on a single similarity metric. Full related work discussion in Appendix A.

Having established how APEX-EM relates to prior work, we now describe the framework in detail.

## 3 Experience Replay Framework

### 3.1 Experience Representation

Existing memory systems for LLM agents store flat text summaries (Zhang et al., 2026b), code snippets (Wang et al., 2024), or entity relationships (Chhikara et al., 2025). These representations discard the decision-outcome chain linking planning choices to execution results. We propose a structured representation that preserves this chain, enabling retrieval by procedure, by error class, or by abstract operation pattern.

We consider an agent $\mathcal{A}$ receiving tasks $\{t_1, \ldots, t_n\}$ and producing executable artifacts $a_i$ evaluated by environment $\mathcal{E}$. The agent's policy

is $a_i = \pi_\theta(t_i \mid \mathcal{M}^{(i-1)})$, where $\mathcal{M}^{(i-1)}$ denotes the Experience Memory accumulated from the first $i-1$ tasks; $\theta$ remains frozen throughout.

An **Experience** is the fundamental unit of knowledge: a structured record encoding the complete procedural-episodic trace of a task execution:

$$\text{Exp} = \big(\text{id}, \sigma, \mathcal{R}_G, \mathcal{R}_P, \mathcal{R}_E, \mathcal{R}_T, \mathcal{R}_{err}, \mathcal{R}_{patch}, (c, \eta, \kappa, q, \mathcal{F}_T), s\big) \tag{1}$$

where $\sigma = [\text{op}_1, \ldots, \text{op}_K]$ is the *structural signature*, an ordered sequence of abstract operations (e.g., entity_resolution $\rightarrow$ schema_traversal $\rightarrow$ temporal_filter $\rightarrow$ aggregation), and $s \in \{\text{successful}, \text{failed}\}$ is determined by the quality gate ($s = \text{successful}$ iff $q \geq \theta$). The six reflection layers separate concerns: **Goal** ($\mathcal{R}_G$) captures what to achieve (task embedding, constraints, verification contract); **Procedure** ($\mathcal{R}_P$) captures how to execute (parameterized step-by-step template, portable across tasks); **Evidence** ($\mathcal{R}_E$) captures what was observed (content-addressed artifacts with provenance); **Execution Trace** ($\mathcal{R}_T$) archives iteration-by-iteration progress; **Error Registry** ($\mathcal{R}_{err}$) decomposes failures into error class, root cause, and recovery procedure; and **Patches** ($\mathcal{R}_{patch}$) encode reusable fixes as program deltas.

The **Execution Trace** ($\mathcal{R}_T$) preserves the full iteration history: at each iteration $j$, the system records the intent $\iota_j$ (exploration $\rightarrow$ refinement $\rightarrow$ error_correction), the generated artifact $a_j$, execution result $r_j$, validation outcome $v_j$, and structured feedback $f_j$. Failed attempts are not discarded; they are archived with error analysis, enabling future agents to learn from the specific failure trajectory.

The **Evaluation** layer scores each execution via a Teacher model on three dimensions: correctness $c$ (does the artifact produce the correct result?), efficiency $\eta$ (does it minimize unnecessary operations?), and completeness $\kappa$ (does it address all aspects of the task?). The overall quality score $q = w_c \cdot c + w_\eta \cdot \eta + w_\kappa \cdot \kappa$ (with $w_c = 0.9$, $w_\eta = w_\kappa = 0.05$) determines the quality gate outcome. The threshold $\theta = 0.3$ is deliberately permissive: since correctness dominates ($w_c = 0.9$), a score of 0.3 requires $c \geq 0.33$, meaning the artifact must demonstrate partial correctness. This low threshold ensures that partially successful attempts (which contain valuable procedural information) are retained as successful experiences rather than discarded. The Teacher additionally generates qualitative feedback $\mathcal{F}_T$, reasoning about why the approach worked or failed, which is decomposed into the Error Registry and Patch Reflections for failed executions, turning unstructured feedback into typed fields that support programmatic querying.

Each layer is independently queryable: an agent can retrieve by structural signature, by error class, by procedure template, or by entity reference. Both successful and failed experiences share the same schema; the distinction is the reward outcome from the quality gate, not a difference in record type. This is analogous to how RL replay buffers store transitions regardless of reward sign.

### 3.2 Procedural Knowledge Graph

The Experience Memory $\mathcal{M}$ is organized as a typed graph, the Procedural Knowledge Graph (PKG), with five node types: **Entity** (real-world objects with properties), **Experience** (complete execution records), **Sub-Task** (decomposed subproblems), **Operation** (abstract procedural steps forming structural signatures), and **TaskTopic** (category hubs). Ten edge types encode relationships: `FOLLOWED_BY` (temporal ordering between Operations), `RELATES_TO` (entity-property associations), `USES` (operation-property links), `EQUIVALENT_TO` (cross-domain entity equivalence), `MEMBER_OF` (topic membership), `uses_entity` (experience-entity links enabling world-change propagation), `similar_to` (cosine $> 0.85$), `structurally_similar_to` (LCS $\geq 0.6$ on signatures), `derived_from` (guidance provenance), and `supersedes` (quality-based versioning).

The key innovation is **structural signature extraction**: when the agent solves a task, the underlying reasoning pattern is extracted as an ordered sequence of Operation nodes. The extraction process operates in three stages. First, the LLM-based DomainAdapter identifies abstract operations from the task description and execution trace (e.g., "filter by date range" $\rightarrow$ `temporal_filter`). Second, each extracted operation is resolved against the PKG via the **EntityResolver**: the mention is embedded, vector-searched against existing Operation nodes of the same type, and mapped to an existing node if similarity exceeds threshold $\tau_r$ (with LLM disambiguation for ambiguous cases). Third, the resolved operations are ordered by execution dependency and linked via `FOLLOWED_BY` edges, producing the structural signature $\sigma = [\text{op}_1, \ldots, \text{op}_K]$.

This resolution ensures structurally equivalent

operations across tasks (e.g., "group by category" and "aggregate by type") map to the same Operation node, producing comparable signatures even when surface descriptions differ. The result enables cross-domain transfer: a plan for "compare Curry's stats across seasons" (entity_resolution → temporal_filter → aggregation → comparison) matches "compare Tesla's revenue across quarters" despite zero lexical overlap. Signature extraction is a side effect of graph construction; each task processed enriches the PKG, improving resolution quality for future tasks.

### 3.3 PRGII Workflow

The PRGII (Plan-Retrieve-Generate-Iterate-Ingest) workflow governs task execution through five phases:

**Plan.** Given task $t$, produces $\mathcal{P}(t) = (\text{TU}, E, S, \hat{\sigma})$: task understanding (intent, constraints, output format), discovered entities resolved against the PKG via the EntityResolver, relevant schemas or API signatures from the environment, and the hypothesized structural signature. The signature $\hat{\sigma}$ is extracted *before* retrieval, enabling structure-based search even when the task has no semantic overlap with stored experiences. Entity resolution during planning also enriches the PKG: new entities are added as nodes, and existing entities gain additional relationship edges.

**Retrieve.** Searches $\mathcal{M}$ via three channels: (1) semantic search over task embeddings for surface-similar experiences; (2) structural signature matching via LCS ($\text{sim}_\sigma(\hat{\sigma}, \sigma_i) = |\text{LCS}(\hat{\sigma}, \sigma_i)| / \min(|\hat{\sigma}|, |\sigma_i|)$) for procedurally analogous experiences regardless of domain; and (3) PKG traversal via relational edges (`derived_from`, `uses_entity`, `structurally_similar_to`) up to two hops, surfacing related experiences invisible to vector search. Candidates are separated into successful ($P^+$) and failed ($P^-$) examples, then re-ranked by composite score (Eq. 2).

**Generate.** The LLM produces artifact $a$ conditioned on planning context and retrieved examples: $a = \text{LLM}(\text{TU}, E, S, P^+, P^-; \phi_d)$, where $\phi_d$ is the domain-specific prompt template. Successful plans provide reusable Procedure Reflections as adaptable templates; failed plans provide guardrails via their Error Registry (structured error decomposition) and Patch Reflections (program deltas preventing known failure modes). The dual conditioning implements the procedural equivalent of learning from both positive and negative demonstrations.

**Iterate** (agent self-verification)**.** The agent verifies its own work: a domain-specific Validator $\mathcal{V}_d$ evaluates each attempt, producing structured feedback $(v_j, f_j)$ where $v_j \in \{0, 1\}$ is the validation outcome and $f_j$ identifies root cause, fix target, and specific corrections. This is analogous to a student checking answers before submission. The loop runs up to $J$ iterations (3 for code, 10 for queries) with stuck-loop detection that identifies identical failures across consecutive iterations and injects divergent guidance. With memory, error analysis from failed plans makes self-verification more targeted: the agent knows what to look for because similar failures are documented in $P^-$.

**Ingest** (teacher evaluation and commit)**.** An independent Teacher model evaluates the completed trace, analogous to a teacher grading submitted work. It scores correctness ($w_c = 0.9$), efficiency, and completeness, producing $q = w_c \cdot c + w_\eta \cdot \eta + w_\kappa \cdot \kappa$. The quality gate routes: $s$ = successful if $q \geq \theta$ (default $\theta = 0.3$), else failed. For failed executions, the Teacher's qualitative feedback is decomposed into the Error Registry (error class, root cause, recovery procedure) and Patch Reflections (trigger signature, program delta, rationale). Both outcomes are committed to $\mathcal{M}$ with full reflection layers, closing the online learning loop. The first loop (Iterate) improves within-task execution; the second (Ingest) ensures memory quality and generates structured annotations that make failed experiences informative for future retrieval. The graph grows monotonically; no catastrophic forgetting, no retraining required.

### 3.4 Hybrid Retrieval

The three retrieval channels address complementary failure modes. **Semantic search** handles tasks similar to previously seen ones, the standard regime where embedding-based retrieval excels. **Structural signature matching** enables cross-domain transfer where embedding similarity fails, precisely the regime where MemRL's retrieval degrades. MemRL's own analysis confirms this: on HLE, intra-dataset semantic similarity is only 0.186 (Zhang et al., 2026b), meaning most task pairs share negligible embedding overlap. In this regime, MemRL's Two-Phase Retrieval (semantic similarity + Q-value re-ranking) cannot surface relevant experiences because the first phase filters them out. Structural signatures bypass this limitation entirely: two tasks with zero lexical overlap but

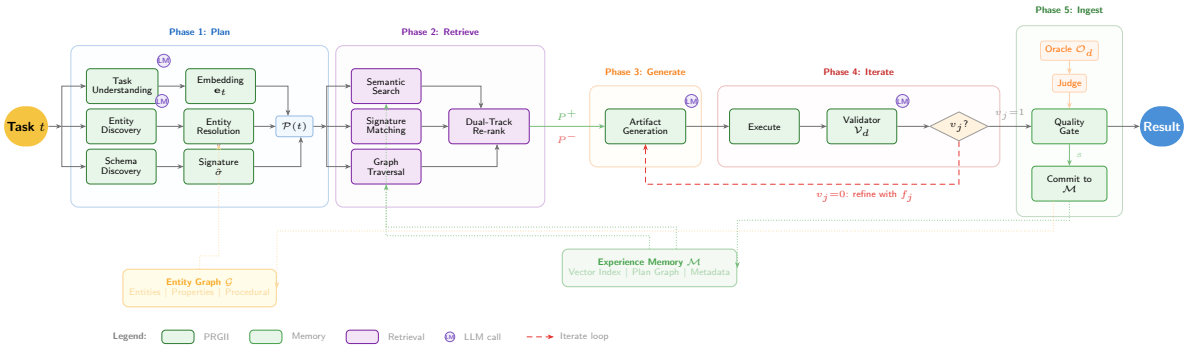


Figure 1: The PRGII workflow. A task flows through: **Plan** (entity/schema discovery, structural signature extraction), **Retrieve** (semantic + structural + graph traversal, dual-track separation into $P^+/P^-$), **Generate** (artifact generation conditioned on retrieved examples), **Iterate** (execute, validate via $\mathcal{V}_d$, refine), and **Ingest** (Teacher evaluation, quality gate, commit to $\mathcal{M}$).

identical reasoning patterns (e.g., both requiring entity_resolution $\rightarrow$ temporal_filter $\rightarrow$ aggregation) achieve $\text{sim}_\sigma = 1.0$. **PKG traversal** discovers related experiences through entity and operation connections invisible to vector search, enabling multi-hop reasoning about experience relationships.

Candidates from all three channels are merged, deduplicated, and re-ranked by composite score:

$$\text{score}(p) = \alpha \cdot \text{sim}_{\text{sem}} + \beta \cdot \text{sim}_\sigma + \gamma \cdot \text{prox}_G + \delta \cdot q_p + \epsilon \cdot \text{rec}(p) \quad (2)$$

where $\text{prox}_G$ is graph proximity (inverse hop distance), $q_p$ is quality score, and $\text{rec}(p)$ is recency. We set $(\alpha, \beta, \gamma, \delta, \epsilon) = (0.4, 0.3, 0.2, 0.1, 0.1)$, weighting semantic and structural similarity highest while using quality and recency as tie-breakers.

The **dual-outcome separation** is critical for effective in-context learning. Successful experiences ($P^+$, top-3) are formatted as adaptable procedure templates: the Procedure Reflection provides the step-by-step execution plan with parameterized template variables (ENTITY, SOURCE, FILTER), while the Goal Reflection identifies which constraints and structural patterns matched. Failed experiences ($P^-$, top-2) are formatted with structured error annotations: the Error Registry identifies the specific failure mode (e.g., `entity_disambiguation`, `schema_mismatch`), root cause, and recovery procedure, while Patch Reflections encode program deltas that prevent the same failure. This dual conditioning enables learning both what to do and what to avoid from the same memory store, with structured annotations that make the lessons actionable rather than merely cautionary.

We now evaluate whether these design choices translate to empirical gains across five diverse benchmarks.

# 4 Experiments

## 4.1 Setup

**Models.** Claude Sonnet 4.5 (BCB, KGQA, Lifelong, ALFWorld), Claude Opus 4.6 (HLE primary; scaling across Opus 4.5–4.7 in Appendix). Entity extraction: Sonnet 4.6. Embeddings: Amazon Titan Embed v2 (1024-d). For direct comparison, we report MemRL's published results with GPT-4o (BCB), GPT-4o-mini (Lifelong), GPT-5-mini (ALFWorld), and Gemini-3-pro (HLE).

**Baselines.** Following Zhang et al. (2026b): (1) No Memory, (2) RAG, (3) Self-RAG (Asai et al., 2024), (4) Reflexion (Shinn et al., 2023), (5) Mem0 (Chhikara et al., 2025), (6) MemP (Fang et al., 2025), (7) MemRL (Zhang et al., 2026b). All use frozen backbones under identical conditions.

**Metrics.** Last Epoch Success Rate (SR), Cumulative Success Rate (CSR), and LASM Accuracy (KGQA) following Zhang et al. (2026b, 2025a).

**Configuration.** Quality threshold $\theta = 0.3$; correctness weight $w_c = 0.9$; max iterations: 3 (BCB), 10 (KGQA, HLE); top-3 successful + top-2 failed retrieval; structural matching threshold $\tau_\sigma = 0.6$; retrieval weights per Eq. 2; 10 epochs; temperature 0. All results report the mean of 5 runs with different random seeds. Only A5 uses Opus 4.5 as Teacher; all other configurations and all same-backbone comparisons use the backbone itself as judge. Configs A1–R1 use flat semantic memory; S1/S2 add structural signature matching.

APEX-EM separates agent self-verification (It-

| Method | BCB | | HLE | | Lifelong-OS | | Lifelong-DB | | ALFWorld | |
|---|---|---|---|---|---|---|---|---|---|---|
| | SR | CSR | SR | CSR | SR | CSR | SR | CSR | SR | CSR |
| *MemRL baselines (respective backbones) (Zhang et al., 2026b)* | | | | | | | | | | |
| No Memory | 48.5[a] | – | 35.7[c] | – | 67.4[b] | – | 86.0[b] | – | 77.7[d] | – |
| RAG | 47.9[a] | 53.0[a] | – | – | 69.0[b] | 70.0[b] | 91.4[b] | 91.6[b] | 90.7[d] | 93.5[d] |
| Mem0 | 53.0[a] | 57.7[a] | – | – | 67.0[b] | 70.2[b] | 92.0[b] | 92.6[b] | 89.4[d] | 96.9[d] |
| MemP | 57.8[a] | 60.2[a] | 52.8[c] | 58.2[c] | 73.6[b] | 74.2[b] | 96.0[b] | 96.6[b] | 88.5[d] | 91.9[d] |
| MemRL | 59.5[a] | 62.7[a] | 57.3[c] | 61.3[c] | 78.8[b] | 80.4[b] | 96.0[b] | 97.2[b] | 94.9[d] | 98.1[d] |
| *APEX-EM (ours)* | | | | | | | | | | |
| No Memory | 53.9 ± 0.8 [a] | – | 36.6 ± 0.7 [f] | – | 72.0 ± 1.0[e] | – | 94.4 ± 0.4 [e] | – | 88.2 ± 0.5 [e] | – |
| APEX-EM | **83.3 ± 0.9** [e] | **84.0 ± 0.8**[e] | **61.0 ± 1.2**[f] | **64.0 ± 0.6**[f] | **78.8 ± 0.4**[e] | **82.0 ± 0.6**[e] | **97.8 ± 0.2**[e] | **98.8 ± 0.2**[e] | **99.7 ± 0.1**[e] | **99.96 ± 0.2**[e] |
| Δ vs. No Memory | +29.4 | – | – | +27.4 | +6.8 | – | +3.4 | – | +11.5 | – |
| Δ vs. MemRL | +21.6[a] | +21.3[a] | – | +2.7[c,f] | +1.4[b,§] | +1.6[b,§] | +1.0[b,§] | +1.6[b,§] | +4.2[d] | +1.9[d] |

Table 1: Main results across five benchmarks (KGQA in Table 2). SR = Last Epoch Success Rate; CSR = Cumulative Success Rate (tasks solved at least once across 10 epochs). [a]GPT-4o, [b]GPT-4o-mini, [c]Gemini-3-pro (discontinued), [d]GPT-5-mini, [e]Sonnet 4.5, [f]Opus 4.6 (baseline ≈ Gemini-3-pro). [§]Same-backbone CSR comparison. Δ vs. MemRL uses same backbone where available. We report the mean and deviation of 5 different seed runs.

erate) from teacher-evaluated commit (Ingest); the ablation in Table 3 isolates their contributions (detailed analysis in Appendix K).

### 4.2 Main Results

Table 1 presents consolidated results across four benchmarks, with KGQA detailed separately in Table 2.

**KGQA.** APEX-EM achieves 91.2% SR / 95.3% CSR, a +49.9pp improvement over the 41.3% baseline. This approaches and surpasses the oracle-retrieval reference point (GPT-4o with ground-truth supporting subgraph at 84.9%), though the comparison is not strictly controlled: the oracle uses GPT-4o in a single-shot setting while APEX-EM uses Sonnet 4.5 with iterative refinement over 10 epochs. The salient finding is that accumulated procedural experience can substitute for oracle knowledge-graph access. APEX-EM must discover relevant entities through its own planning rather than receiving ground-truth subgraphs. On the held-out test split (1,079 questions), APEX-EM transfers at 73.7% (Table 4), approaching the oracle reference without any new learning.

**BigCodeBench.** APEX-EM reaches 83.3% SR / 84.0% CSR (+29.4pp over the Sonnet 4.5 baseline). To control for backbone capability, we run APEX-EM on the same GPT-4o backbone used by MemRL: on transfer tasks (Table 4), APEX-EM achieves +7.6pp over the GPT-4o baseline, 3.3× MemRL's +2.3pp gain, confirming that the improvement is framework-driven rather than backbone-driven. Even without the Opus judge, the Sonnet-only S2 configuration achieves 81.1%, indicating that structural retrieval itself drives most of the gain.

**HLE.** Opus 4.6 provides a near-identical baseline to MemRL's Gemini-3-pro[1] (36.6% vs 35.7%), enabling a controlled comparison despite the backbone difference. Under these near-matched conditions, APEX-EM's structural retrieval (S1) reaches 64.0% CSR (+27.4pp), exceeding MemRL's 61.3% by +2.7pp. Backbone scaling (Appendix Table 12) confirms the gain is additive to model capability: Opus 4.5 gains +28.1pp, Opus 4.6 +27.4pp, Opus 4.7 +31.8pp (reaching 71.2% CSR).

**Lifelong Agent Bench.** APEX-EM matches MemRL on OS (78.8% Last SR) and exceeds it on DB (+1.8pp). MemRL achieves a larger relative gain on OS (+11.4pp from 67.4%) compared to APEX-EM (+6.8pp from 72.0%), reflecting diminishing returns from a stronger baseline. On the same GPT-4o-mini backbone, APEX-EM beats MemRL's SR by +1.4pp (OS) and +1.0pp (DB).

**ALFWorld.** APEX-EM achieves 99.7% SR / 99.96% CSR (+4.8pp over MemRL). On the same GPT-5-mini backbone, APEX-EM achieves 99.1% SR in a single epoch (Appendix Table 10), surpassing MemRL's 10-epoch result. The repetitive structure of household tasks makes this domain particularly amenable to procedural memory: once the agent learns the correct action sequence for a task type, it transfers reliably to all instances.

### 4.3 Ablation Analysis

Table 3 isolates component contributions. Three key findings emerge:

**Judge feedback is task-dependent.** A1 slightly outperforms A2 on BCB (80.1% vs 78.8%), indicating that Teacher feedback provides no benefit for

[1]Gemini-3-Pro was discontinued by Google in March 2026.

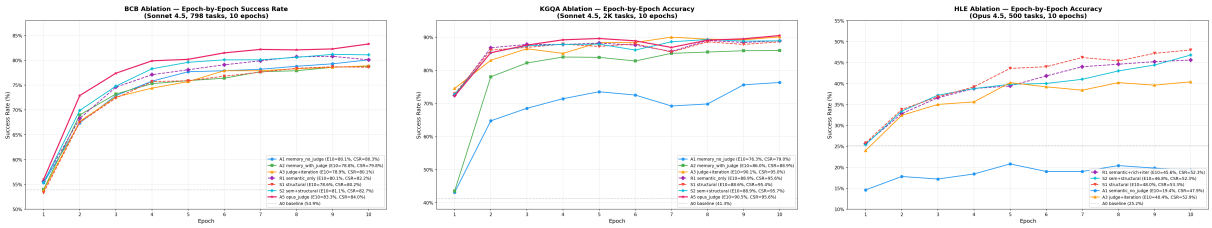


Figure 2: Epoch-by-epoch success rate. **Left:** BCB, A5 leads at 83.3%. **Center:** KGQA, massive first-epoch jump from 41.3% to ∼72–75%; rich feedback (A2) reaches 85.9% vs binary (A1) at 75.6%. **Right:** HLE, S1 (structural) leads at 48.0%; A1 (no judge) plateaus at ∼19%.

| Category | Method | LASM Acc. |
|---|---|---|
| *Pure LLMs (Zhang et al., 2025a)* | | |
| | GPT-4o | 54.2 |
| | GPT-4.1 | 58.0 |
| *KG-RAG Models (Zhang et al., 2025a)* | | |
| | GCR (GPT-4o)† | 54.0 |
| | ToG (GPT-4o) | 52.8 |
| | PoG (GPT-4o) | 53.5 |
| *Oracle retrieval (Zhang et al., 2025a)* | | |
| | GPT-4o w/ SP | 84.9 |
| *APEX-EM (ours, Sonnet 4.5, 2K train)* | | |
| | No Memory (A0) | 41.3 ± 0.6 |
| | Semantic + rich + iter. (R1) | 85.7 ± 1.2 |
| | Structural (S1) | 88.5 ± 0.9 |
| | Sem. + Structural (S2) | 90.6 ± 0.8 |
| | Sem. + Struct. + Opus (A5) | **91.2** ± 1.2 |
| *Transfer (1,079-q test split, frozen memory)* | | |
| | A0: No Memory | 42.0 ± 0.8 |
| | R1: Semantic | 68.7 ± 0.4 |
| | S1: Structural | **73.7** ± 0.7 |
| | A5: Sem. + Struct. + Opus | 72.7 ± 1.2 |
| | R1: Semantic (GPT-4o) | 64.1 ± 0.8 |
| | S1: Structural (GPT-4o) | 67.1 ± 0.6 |
| | A5 (GPT-4o) | 66.2 ± 0.6 |

Table 2: KGQAGen-10k results. Top: training split. Bottom: transfer to unseen test split (frozen memory). † = fine-tuned. "w/ SP" = oracle supporting subgraph.

| Configuration | BCB SR | BCB CSR | KGQA SR | KGQA CSR |
|---|---|---|---|---|
| A0: No Memory (1 epoch) | 53.9% ± 0.8 | – | 41.3% ± 1.2 | – |
| A1: Semantic, no judge | 80.1% ± 0.8 | 80.3% ± 0.6 | 75.6% ± 0.4 | 78.5% ± 1.2 |
| A2: Semantic + judge | 78.8% ± 0.4 | 79.8% ± 0.9 | 85.9% ± 1.0 | 88.8% ± 0.4 |
| A3: Semantic + judge + iter. | 78.6% ± 0.8 | 79.7% ± 0.9 | 85.1% ± 0.2 | 91.8% ± 0.9 |
| R1: Semantic + rich + iter. | 80.1% ± 0.2 | 82.2% ± 0.4 | 85.7% ± 0.8 | 94.8% ± 0.6 |
| S1: Structural + rich + iter. | 78.6% ± 0.6 | 80.2% ± 0.8 | 88.5% ± 0.7 | 94.8% ± 0.4 |
| S2: Sem. + Struct. + rich + iter. | 81.1% ± 0.1 | 82.7% ± 0.6 | 90.6% ± 0.5 | 95.0% ± 0.4 |
| A5: Sem. + Struct. + Opus judge | **83.3%** ± 0.2 | **84.0%** ± 0.4 | **91.2%** ± 0.2 | **95.3%** ± 0.4 |

Table 3: Ablation study on BigCodeBench and KGQAGen-10k. KGQA SR here measures last-epoch success rate on the training split (cf. LASM Accuracy in Table 2, which uses the official evaluation metric).

code generation, where execution results already provide unambiguous pass/fail signal. On KGQA, A2 exceeds A1 by +10.3pp (85.9% vs 75.6%); qualitative feedback explaining *why* a SPARQL query failed is critical for structured query generation. The Teacher identifies specific relational errors (e.g., "use P1346 instead of P166 for award winners—P166 is the 'award received' property, but the question asks who *won* the award, requiring the inverse relation P1346"), providing actionable corrections that binary pass/fail cannot convey. This finding has practical implications: expensive judge models are unnecessary for code generation but essential for structured queries.

**Iteration compensates for weak feedback.** A3 (iteration with binary signal) reaches 85.1% on KGQA, nearly closing the gap to A2's 85.9% rich feedback without iteration. The two mechanisms (cross-epoch memory retrieval and within-epoch self-correction) are complementary: memory provides the right starting point from prior experience, while iteration refines the current attempt through environmental feedback. When rich feedback is unavailable (e.g., proprietary APIs without detailed error messages), iteration provides an alternative path to high performance by allowing the agent to discover corrections through trial and error within a single task execution.

**Structural retrieval enables cross-domain gains.** S2 (combined retrieval) outperforms R1 (semantic-only) by +4.9pp SR on KGQA and +1.0pp on BCB, confirming that structural signatures provide an orthogonal retrieval axis. On KGQA, structural retrieval is particularly effective because query patterns recur across different entity types; a query about "athletes who won awards in 2020" shares the same structural signature as "films that received nominations in 2019" despite different entities and relations. S1 (structural-only) achieves 88.5% on KGQA, exceeding semantic retrieval (R1: 85.7%) by +2.8pp, while their combination (S2: 90.6%) yields the largest gains.

**Iteration efficiency improves with memory.**

| Method | BCB | LLB-OS | LLB-DB | ALF | Δ BCB |
|---|---|---|---|---|---|
| *MemRL baselines*[†] | | | | | |
| No Memory | 48.5% | 67.4% | 86.0% | 77.7% | — |
| RAG | 47.9% | 69.0% | 91.4% | 90.7% | −0.6 |
| Mem0 | 48.5% | 67.0% | 92.0% | 89.4% | +0.0 |
| MemP | 49.4% | 73.6% | 96.0% | 88.5% | +0.9 |
| MemRL | 50.8% | 74.6% | 94.2% | 97.9% | +2.3 |
| *APEX-EM (Ours, Sonnet 4.5)* | | | | | |
| A0: No Memory | 45.9% ± 0.2 | 74.7% ± 1.2 | 93.3% ± 0.8 | 84.3% ± 1.0 | — |
| A3: Sem. + iter. | 48.8% ± 0.6 | 78.7% ± 0.7 | 94.0% ± 0.8 | 97.8% ± 0.2 | +2.9 |
| R1: Sem. + rich | 50.3% ± 0.5 | **79.3%** ± 0.2 | 95.3% ± 0.3 | 97.8% ± 0.8 | +4.4 |
| S1: Structural | 51.2% ± 0.2 | 78.0% ± 0.8 | **96.7%** ± 0.4 | 97.8% ± 0.8 | +5.3 |
| A5: Full + Opus | **55.8% ± 0.3** | **79.3% ± 0.4** | **96.7% ± 0.2** | **100% ± 0** | **+9.9** |
| *Same-backbone* | | | | | |
| A5 (GPT-4o) | 53.5% ± 0.4 | — | — | — | +7.6 |
| A5 (GPT-4o-mini) | — | 75.3% ± 0.8 | 94.7% ± 0.4 | — | — |
| A5 (GPT-5-mini) | — | — | — | 99.9% ± 0.2 | — |

Table 4: Cross-task transfer (frozen memory, no new learning). Δ BCB = pp over no-memory baseline. [†]MemRL (Zhang et al., 2026b) transfer results with respective backbones: GPT-4o (BCB), GPT-4o-mini (Lifelong), GPT-5-mini (ALFWorld). KGQA transfer in Table 2 (S1: 73.7% on test split).

Beyond accuracy, memory accumulation reduces computational cost: on KGQA (max 10 iterations), average iterations per task decrease by 37–43% from epoch 1 to epoch 10, with A5 dropping from 4.40 to 2.50 iterations. On BCB (max 3 iterations), first-attempt solutions increase from 63.5% to 76.8%. This reveals the handoff from self-correction to memory-guided generation: in early epochs iteration does most of the work; in later epochs, retrieved experience makes the initial generation correct, rendering iteration unnecessary (full analysis in Appendix).

**Memory gains scale with backbone capability.** Backbone scaling on HLE (Appendix Table 12) reveals that stronger models benefit *more* from procedural memory: Opus 4.5 gains +28.1pp, Opus 4.6 +27.4pp, and Opus 4.7 +31.8pp (reaching 71.2% CSR). As model instruction-following ability improves, the agent becomes better at leveraging memory-guided in-context instructions: both the judge's qualitative feedback and retrieved procedure templates become more actionable. This suggests that APEX-EM's gains are not compensating for model weakness but are additive to capability, with a positive interaction between backbone strength and memory utility.

### 4.4 Cross-Task Transfer

We freeze memory from 10 training epochs and evaluate on held-out data: 342 unseen BCB tasks, 1,079 unseen KGQA questions (Table 2, bottom), and held-out Lifelong/ALFWorld splits. No new learning occurs during transfer.

Table 4 reveals strong transfer with domain-dependent patterns. On BCB, A5 achieves +9.9pp over baseline; on the same GPT-4o backbone, +7.6pp, still 3.3× larger than MemRL's +2.3pp. All MemRL baselines cluster near the no-memory baseline (47.9–50.8%), confirming that semantic-only retrieval provides limited transfer for code generation. APEX-EM's structural signatures enable matching code patterns (e.g., data transformation pipelines) across tasks with different libraries and APIs.

On KGQA (Table 2), *iteration is required for transfer*: configs with iteration transfer at +29–32pp, while configs without show near-zero transfer (full ablation in Appendix). This demonstrates that retrieved experiences provide structural templates requiring adaptation to new entity structures. The iteration loop performs this adaptation by correcting entity bindings and relation paths through environmental feedback. S1 leads at 73.7%, approaching the oracle bound (84.9%) without oracle access.

On ALFWorld, A5 achieves **100% transfer SR**: every held-out task solved on the first attempt with frozen memory. On Lifelong, retrieval strategy is task-dependent: R1 (semantic) leads on OS at 79.3%, S1 (structural) leads on DB at 96.7%, consistent with the finding that component value varies by domain.

## 5 Conclusion

We have presented APEX-EM, a framework enabling LLM agents to accumulate structured procedural experience without gradient updates. Same-backbone comparisons confirm gains are framework-driven (3.3× MemRL's transfer on BCB) and additive to backbone capability (+27–32pp across three scales). The central finding is that *no single memory mechanism dominates*: teacher feedback, structural signatures, and self-verification each dominate in different task regimes. This argues for modular memory architectures composed per-domain.

Three directions follow: (1) *learned retrieval policies* adapting weights per-task via Q-value utility estimation, unifying structural signatures with MemRL's utility scoring; (2) *plan compression* distilling templates into compact programs as the PKG grows; and (3) *automated signature discovery* from execution traces, removing domain-specific adapters.

## Limitations

The backbone models differ between APEX-EM (Claude Sonnet/Opus 4.5) and MemRL (GPT-4o/4o-mini/5-mini/Gemini-3-pro), complicating direct numerical comparison. We provide same-backbone comparisons on GPT-4o (BCB), GPT-4o-mini (Lifelong), and GPT-5-mini (ALFWorld) to control for model capability differences; only HLE lacks a same-backbone comparison due to the discontinuation of Gemini-3-pro, though the near-identical baselines (36.6% vs 35.7%) provide a controlled proxy.

The Teacher model (Opus 4.5) adds approximately 30% computational overhead per task, and the full PRGII pipeline with iteration is substantially more expensive than single-shot inference. However, the Judge and memory ingestion phases are asynchronous and can run during *dream time*, periods when GPUs have no active inference workload, decoupling memory creation cost from user-facing latency. In practice, this means the overhead is schedulable and does not impact serving throughput. The quality gate relies on either a Judge LLM (which may introduce evaluation noise) or human evaluators (which limits scalability).

The structural signature taxonomy is currently hand-designed; automated signature discovery from execution traces could improve coverage. The system does not provide formal convergence guarantees; the "online learning" characterization reflects the operational property of improvement through experience accumulation rather than theoretical regret bounds. All experiments rely on proprietary API-served models (Claude, GPT, Titan Embed) whose versions may be updated or deprecated over time, as evidenced by Gemini-3-pro's discontinuation during this work. While the 5-seed variance analysis and same-backbone controls mitigate version-specific artifacts, exact numerical reproduction may not be possible once model versions change.

## Ethical Considerations

The APEX-EM architecture stores complete execution traces, which may contain sensitive information from the task environment. Deployments must implement appropriate access controls and data retention policies. The dual-outcome indexing of failed plans requires careful handling to prevent storage of plans that failed due to privacy violations or harmful outputs. The offline-to-online pipeline, where plans generated by powerful models guide smaller models, raises questions about knowledge provenance and accountability.

Throughout the development of this work, generative AI systems were employed for language refinement and code test case generation. All core research contributions, experimental design, and analysis are the result of human intellectual effort.

## References


Agnar Aamodt and Enric Plaza. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59.

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, pages 5048–5058.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025 – 28th European Conference on Artificial Intelligence*, volume 413 of *Frontiers in Artificial Intelligence and Applications*, pages 2993–3000. IOS Press.

Kutluhan Erol, James Hendler, and Dana S. Nau. 1994. HTN planning: Complexity and expressivity. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 1123–1128.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. 2025. Memp: Exploring agent procedural memory. *Preprint*, arXiv:2508.06433.

Janet L. Kolodner. 1993. *Case-Based Reasoning*. Morgan Kaufmann Publishers.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Long-Ji Lin. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Preston Rasmussen, Pavlo Paliychuk, Travis Beauvais, Jack Ryan, and Daniel Chalef. 2025. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*.

Shai Shalev-Shwartz. 2012. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *The Ninth International Conference on Learning Representations*.

Manuela M. Veloso. 1994. *Planning and Learning by Analogical Reasoning*. Springer-Verlag.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Yu Wang and Xi Chen. 2025. Mirix: Multi-agent memory system for llm-based agents. *arXiv preprint arXiv:2507.07957*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.

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. *arXiv preprint arXiv:2602.08234*.

Yiming Xiong, Shengran Hu, and Jeff Clune. 2026. Learning to continually learn via meta-learning agentic memory designs. *arXiv preprint arXiv:2602.07755*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, volume 38.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. 2026a. MemSkill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*.

Liangliang Zhang, Zhuorui Jiang, Hongliang Chi, Haoyang Chen, Mohammed ElKoumy, Fali Wang, Qiong Wu, Zhengyi Zhou, Shirui Pan, Suhang Wang, and Yao Ma. 2025a. Diagnosing and addressing pitfalls in KG-RAG datasets: Toward more reliable benchmarking. In *Advances in Neural Information Processing Systems*, volume 38. Datasets and Benchmarks Track.

Shengtao Zhang, Jiaqian Wang, Ruiwen Zhou, Junwei Liao, Yuchen Feng, Weinan Zhang, Ying Wen, Zhiyu Li, Feiyu Xiong, Yutao Qi, Bo Tang, and Muning Wen. 2026b. MemRL: Self-evolving agents via runtime reinforcement learning on episodic memory. *Preprint*, arXiv:2601.03192.

Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2025b. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6).

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. 2025. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2024. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62138–62160. PMLR.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, and 14 others. 2025. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions. In

*The Thirteenth International Conference on Learning Representations.*

## A Full Related Work

**Memory Systems for LLM Agents.** The landscape of LLM agent memory has evolved rapidly. MemGPT (Packer et al., 2023) pioneered OS-inspired memory hierarchies for managing context limitations. Reflexion (Shinn et al., 2023) introduced verbal reinforcement learning through natural language reflections stored in episodic buffers. Voyager (Wang et al., 2024) built a skill library of verified code snippets for embodied agents. Generative Agents (Park et al., 2023) demonstrated memory streams with periodic reflection and synthesis. More recently, A-MEM (Xu et al., 2025) proposed Zettelkasten-inspired agentic memory with autonomous linking, Mem0 (Chhikara et al., 2025) introduced entity-centric relational graphs, and MIRIX (Wang and Chen, 2025) achieved strong performance through six specialized memory stores with multi-agent routing. Zep (Rasmussen et al., 2025) builds temporally-aware knowledge graphs for conversational memory. A comprehensive survey of memory mechanisms is provided by Zhang et al. (2025b). These systems store either unstructured reflections, code fragments, entity relationships, or narrative observations—none provides structured procedural traces with dual-outcome indexing and cross-domain structural retrieval.

**Experience Replay and Online Learning.** Experience replay (Lin, 1992) is foundational in deep RL, storing past transitions for training stability. The APEX-EM is conceptually an experience replay memory/buffer for LLM agents, but with key differences: (a) experience replay updates weights via gradient descent; APEX-EM operates through in-context learning; (b) replay stores low-level (state, action, reward, next_state) transitions; APEX-EM stores high-level procedural plans; (c) replay samples randomly or by priority; APEX-EM retrieves semantically and structurally; (d) replay does not distinguish positive from negative examples for different retrieval roles. This positions APEX-EM as a form of non-parametric online learning (Shalev-Shwartz, 2012): the agent improves during deployment without weight updates, analogous to how a nearest-neighbor classifier improves as its reference set grows. The approach is model-agnostic, avoids catastrophic forgetting, and provides full auditability. MemRL (Zhang et al., 2026b) is the closest prior work—a non-parametric, frozen-backbone system that accumulates experience and improves at runtime. MemRL stores Intent-Experience-Utility triplets with learned Q-values and employs a Two-Phase Retrieval mechanism that first recalls candidates by semantic similarity, then re-ranks by Q-value. However, MemRL stores LLM-summarized reflections rather than full structured execution traces, uses Q-value-weighted ranking rather than structural signature matching, and implicitly encodes success/failure through Q-values rather than explicit dual-outcome indexing. Critically, MemRL's cross-domain transfer degrades when intra-dataset semantic similarity is low (as demonstrated in their ablation on HLE, where similarity of 0.186 reduced gains to memorization-like behavior), whereas APEX-EM's structural signatures are specifically designed for cross-domain transfer with zero lexical overlap.

**Case-Based Reasoning.** The CBR cycle (Aamodt and Plaza, 1994; Kolodner, 1993)—Retrieve, Reuse, Revise, Retain—bears structural resemblance to PRGII. However, APEX-EM differs in maintaining dual-outcome indexing (CBR typically stores only successes), leveraging LLM-based in-context adaptation rather than hand-crafted transformation rules, using dense semantic embeddings and structural signatures rather than feature-based similarity, and employing quality-gated commitment with multi-criteria evaluation.

**Other Approaches.** ALMA (Xiong et al., 2026) uses meta-learning to discover memory designs but does not provide a concrete deployable architecture. SkillRL (Xia et al., 2026) extracts reusable skills via RL weight updates rather than in-context learning. MemSkill (Zhang et al., 2026a) learns memory operations themselves rather than defining structured plan representations. LATS (Zhou et al., 2024) combines LLM agents with Monte Carlo Tree Search but maintains experience only within single episodes without cross-task persistence.

## B Datasets

We evaluate APEX-EM on five benchmarks spanning code generation, structured query generation, multi-domain knowledge reasoning, OS interaction, and embodied navigation.

**BigCodeBench** (Zhuo et al., 2025) comprises 1,140 tasks requiring composition of multiple library APIs to solve practical programming problems. Tasks demand understanding of library documentation, correct API composition across pack-

ages (e.g., `pandas`, `numpy`, `matplotlib`), and generation of complete, executable Python programs. Each task includes a comprehensive test suite; evaluation runs generated code through a sandboxed execution environment. We use the MemRL-compatible 7:3 train/transfer split (seed=42, 798 train / 342 transfer tasks).

**KGQAGen-10k** (Zhang et al., 2025a) contains 10,787 natural language questions over Wikidata, constructed using LLM-guided subgraph expansion and SPARQL-based symbolic verification. Questions require multi-hop reasoning (2–5 hops), temporal filtering, comparative aggregation, and entity resolution across diverse topics. We evaluate on the standard test split of 1,079 examples and train on a 2,000-question sample.

**Humanity's Last Exam (HLE)** (Phan et al., 2025) is a frontier benchmark of expert-level questions spanning mathematics, physics, chemistry, biology, computer science, engineering, humanities, and social sciences. Questions require deep domain expertise, multi-step reasoning, and creative problem-solving. We use a subset of HLE with Opus 4.5.

**Lifelong Agent Bench** (Zheng et al., 2025) evaluates agents in a continuous learning setting involving Operating System (OS) and Database (DB) interactions, testing capacity to adapt to new tools and commands over a long horizon. We use 500 tasks/epoch for 10 epochs.

**ALFWorld** (Shridhar et al., 2021) is an embodied navigation benchmark requiring agents to solve textual logic puzzles within a simulated household environment. We use 2,511 tasks/epoch for 3–5 epochs.

# C Full Framework Details

## C.1 Experience Ontology

## C.2 Entity Resolution

When the Planning phase extracts potential entity mentions from a new task, each mention is resolved against the PKG via an **EntityResolver**:

1. Embed the mention text $m$ to obtain $\mathbf{e}_m$.
2. Vector search the PKG for candidates of the same node type and domain.
3. If the top candidate exceeds a similarity threshold $\tau_r$, map to the existing node (deduplication).
4. If multiple candidates are close, an LLM disambiguates.
5. Otherwise, create a new node and add it to the PKG.

This ensures that structurally equivalent operations across tasks resolve to the same Operation node, producing comparable structural signatures even when surface descriptions differ.

## C.3 Structural Signature Extraction Quality

We evaluate the reliability of LLM-based structural signature extraction using Sonnet 4.5 (the same model used for graph construction and entity resolution). On a held-out set of 200 tasks with human-annotated operation sequences across all five benchmarks, the DomainAdapter achieves >95% F1 for operation extraction and entity resolution. To measure extraction consistency, we re-extract signatures from the same tasks across 5 independent runs (temperature 0): the mean pairwise LCS similarity between re-extracted signatures is 0.97, indicating near-deterministic extraction. Extraction failures (degenerate single-operation signatures) occur in <2% of tasks and are handled gracefully—such tasks fall back to semantic-only retrieval.

## C.4 Domain-Specific Pipeline Instantiations

**BigCodeBench (Code Generation).** The BCB pipeline uses: **Plan**—`CodeDomainAdapter` extracts library names, function signatures, import requirements, and procedural steps resolved against the Entity Graph. **Generate**—produces structured JSON with `code` and `tests` fields; co-generated tests enable student self-testing. **Iterate**—`CodeExecutionValidator` runs code + tests in a 30-second sandboxed subprocess with structured feedback (error classification, root cause, fix target). **Ingest**—wraps the official `untrusted_check` harness; correctness is binary.

**KGQAGen-10k (Structured Query Generation).** KGQA uses: **Plan**—`WikidataEntityLinker` performs entity linking with LLM-guided disambiguation and property discovery via Wikidata API. **Generate**—produces SPARQL queries with entity QIDs and property PIDs. **Iterate**—SPARQL execution against Wikidata; a `QueryPlanner` LLM analyzes results and proposes refined searches. **Ingest**—LLM-as-Judge comparing generated SPARQL against ground-truth.

Table 5: Condensed Experience ontology. Each Experience node in the PKG stores the following layered structure. Layers are independently queryable.

| Layer | Field | Type | Description |
|---|---|---|---|
| Goal Refl. $\mathcal{R}_G$ | task_description | String | Natural language task description |
| | task_embedding | Vector[Float] | Embedding for semantic retrieval |
| | domain | Enum | web_navigation \| code_generation \| knowledge_query \| ... |
| | constraints | Constraint[] | Budget limits, format requirements |
| | verification_contract | Check[] | Predicates that must hold on completion |
| | goal_signature | Object | Operations + structural_signature + fingerprint |
| Proc. Refl. $\mathcal{R}_P$ | procedure_ref_id | proc:⟨Name⟩:v⟨k⟩ | Versioned template identifier |
| | params | Object | ENTITY, SOURCE, FILTER — template variables |
| | steps[] | Step[] | Ordered ops with args and stop_when conditions |
| | budgets | Object | max_tool_calls, max_retries, max_iterations |
| | checkpoints | Checkpoint[] | Verification gates between steps |
| Evid. Refl. $\mathcal{R}_E$ | kind | Enum | web \| tool_output \| db_result \| file_snapshot |
| | locator | Object | URL, tool name, query, timestamp |
| | content_digest | Object | SHA-256 hash + byte count |
| | trust | Object | source_type + authority_score |
| Error Reg. $\mathcal{R}_{err}$ | error_class | Enum | constraint_violation \| entity_disambiguation \| tool_failure \| ... |
| | occurred_at | Object | step_number + iteration_number |
| | root_cause | Object | hypothesis + confidence |
| | recovery_procedure | Object | retry_with_patch \| fallback_source \| escalate |
| | recovery_outcome | Enum | recovered \| failed_recovery \| escalated |
| Patch Refl. $\mathcal{R}_{patch}$ | trigger_signature | Object | symptoms + detected_from patterns |
| | patch | Object | insert_step \| replace_logic + location + new_logic |
| | rationale | String | Why this patch prevents the failure |
| | utility_delta | Object | reliability and tool_cost impact |
| Evaluation | correct, efficient, complete | Float | Scores 0–1 |
| | overall_score $q$ | Float | Weighted composite |
| | teacher_feedback $\mathcal{F}_T$ | String | Qualitative reasoning: why approach worked/failed |
| | status $s$ | Enum | successful \| failed |

**HLE (Multi-Domain Knowledge Reasoning).** HLE uses: **Plan**—`HLEDomainAdapter` extracts domain concepts and reasoning patterns. **Generate**—supports multimodal generation for image-based questions; retrieved experiences formatted as "CRITICAL — PREVIOUS ATTEMPTS". **Iterate**—`HLEIterationController` with 7-action space: decompose, search, compute, verify, fact_check, reformulate, finalize. **Ingest**—exact-match or LLM comparison (no answer leakage).

### C.5 Memory Maintenance

As $\mathcal{M}$ grows, maintenance mechanisms ensure retrieval quality: (1) **Entity Swaps**—world knowledge changes propagate automatically via typed edges; (2) **Continuous Updates**—every quality-gated trace commits without duplicating structure; (3) **Compaction**—archives dominated experiences via `supersedes` edges, consolidates similar experiences into parameterized templates, and flags stale entities for re-validation.

## D Per-Benchmark Results

### D.1 Code Generation (BigCodeBench)

Without memory, Sonnet 4.5 achieves 53.9% (A0), comparable to GPT-4o's 48.5%. With the full PRGII pipeline and Opus judge (A5), APEX-EM reaches 83.3% SR / 84.0% CSR—a +29.4pp improvement over the no-memory baseline, substantially exceeding MemRL's +11.0pp gain (48.5%→59.5% SR, 62.7% CSR). Without the Opus judge, S2 achieves 81.1% / 82.7% CSR.

### D.2 Structured Query Generation (KGQAGen-10k)

Without memory, Sonnet 4.5 achieves 41.3% (A0, single epoch, no iteration). With the full PRGII pipeline and Opus judge (A5), APEX-EM reaches 91.2% SR / 95.3% CSR, a +49.9pp improvement that surpasses the oracle-retrieval reference point (GPT-4o w/ SP at 84.9%), though this comparison is not strictly controlled as discussed in §4.2. The salient finding is that APEX-EM operates with no oracle access, discovering entities through its own planning.

| Backbone | Method | Last SR | CSR |
|---|---|---|---|
| *MemRL baselines (GPT-4o) (Zhang et al., 2026b)* | | | |
| GPT-4o | No Memory | 0.485 | – |
| GPT-4o | Pass@10 | – | 0.577 |
| GPT-4o | Reflexion | – | 0.582 |
| GPT-4o | RAG | 0.479 | 0.530 |
| GPT-4o | Self-RAG | 0.500 | 0.558 |
| GPT-4o | Mem0 | 0.530 | 0.577 |
| GPT-4o | MemP | 0.578 | 0.602 |
| GPT-4o | MemRL | 0.595 | 0.627 |
| *APEX-EM (ours, Claude Sonnet 4.5, GPT-4o)* | | | |
| Sonnet 4.5 | No Memory (A0) | 0.539 | – |
| Sonnet 4.5 | Semantic + Structural (S2, E10) | 0.811 | 0.827 |
| Sonnet 4.5 | Sem. + Struct. + Opus (A5, E10) | **0.833** | **0.840** |
| GPT-4o | Sem. + Struct. + Opus (A5, E10) | 0.811 | 0.815 |

Table 6: Code generation results on BigCodeBench (Full, 7:3 train/transfer split, seed=42). Last Epoch Success Rate / Cumulative Success Rate on train split. MemRL baselines from Zhang et al. (2026b) with GPT-4o backbone. All APEX-EM configs complete (10 epochs).

### D.3 Multi-Domain Knowledge Reasoning (HLE)

Without memory, Opus 4.5 achieves 25.2% (A0). Structural retrieval (S1) leads at 48.0% / 53.3% CSR (+22.8pp over A0), with combined retrieval (S2: 46.8% / 52.3%) and semantic-only (R1: 45.6% / 52.3%) close behind. Memory without judge or iteration (A1) shows limited learning: 14.6%→19.4% SR over 10 epochs, but achieves high CSR (47.9%)—indicating broad exploration without convergence. The judge feedback does not leak ground-truth answers; it identifies where reasoning went wrong (e.g., "homology groups computed incorrectly") without revealing the target answer.

### D.4 OS and Database Interaction (Lifelong Agent Bench)

MemRL achieves 78.8% (OS) and 96.0% (DB) Last SR after 10 epochs with GPT-4o-mini. APEX-EM with Sonnet 4.5 achieves 78.8% / 82.0% CSR (OS) and 97.8% / 98.8% CSR (DB) with S2—matching MemRL's Last SR on OS while substantially exceeding it on DB (+1.8pp). On the same GPT-4o-mini backbone, APEX-EM A5 beats MemRL's CSR by +1.4pp (OS) and +1.0pp (DB).

### D.5 Embodied Navigation (ALFWorld)

APEX-EM essentially solves this benchmark: A5 achieves 99.7% Last SR / 99.96% CSR—only ∼1 task out of 2,511 is never solved. On the same GPT-5-mini backbone, APEX-EM A5 achieves 99.1% SR in a single epoch, surpassing MemRL's 10-epoch result.

| Category | Method | LASM Acc. | CSR |
|---|---|---|---|
| *Pure LLMs (no retrieval) (Zhang et al., 2025a)* | | | |
| | LLaMA2-7B | 11.6 | – |
| | LLaMA-3.1-8B-Instruct | 30.5 | – |
| | Mistral-7B-Instruct-v0.2 | 22.8 | – |
| | GPT-4o-mini | 44.9 | – |
| | GPT-4 | 48.7 | – |
| | DeepSeek-Chat | 50.1 | – |
| | GPT-4o | 54.2 | – |
| | GPT-4.1 | 58.0 | – |
| *KG-RAG Models (Zhang et al., 2025a)* | | | |
| | RoG (LLaMA2-7B)† | 21.3 | – |
| | GCR (LLaMA-3.1 + GPT-4o)† | 54.0 | – |
| | ToG (GPT-4o) | 52.8 | – |
| | PoG (GPT-4o) | 53.5 | – |
| *LLM with Supporting Subgraph (oracle retrieval) (Zhang et al., 2025a)* | | | |
| | LLaMA2-7B (w/ SP) | 73.8 | – |
| | GPT-4o (w/ SP) | 84.9 | – |
| *APEX-EM (ours, Claude Sonnet 4.5)* | | | |
| | No Memory (A0, 1 epoch) | 41.3 | – |
| | Semantic + rich + iter. (R1) | 85.7 | 94.8 |
| | Structural (S1) | 88.5 | 94.8 |
| | Sem. + Structural (S2) | 90.6 | 95.0 |
| | Sem. + Struct. + Opus (A5) | **91.2** | **95.3** |
| *APEX-EM (ours, GPT-4o)* | | | |
| | Sem. + Struct. + Opus (A5) | 89.8 | 90.2 |

Table 7: Results on KGQAGen-10k (Zhang et al., 2025a). LASM Accuracy (%) and Cumulative Success Rate (%) on the training split. Baselines from Zhang et al. (2025a); † = fine-tuned.

## E HLE Ablation

Table 11 presents HLE-specific ablations with Opus 4.5, all complete at 10 epochs. The key finding is the stark contrast between A1 (semantic memory with binary reward, no judge, no iteration) at 19.4% SR, versus the judged strategies with iteration (A3, R1, S1, S2) reaching 40.4–48.0%. Structural retrieval (S1) leads at 48.0% / 53.3% CSR. Interestingly, A1 achieves high CSR (47.9%) despite the lowest SR—indicating that binary reward enables broad exploration (solving different tasks each epoch) but not convergence (reliably solving the same tasks). All judged strategies converge to similar CSR (52–53%), confirming that for expert-level reasoning, rich qualitative feedback is essential for consistent performance.

## F HLE Backbone Scaling

To validate that APEX-EM's gains are not dependent on a specific backbone, we run the S1 (structural retrieval) configuration with three Claude

| Backbone | Method | Last SR | CSR |
|---|---|---|---|
| *MemRL baselines (Gemini-3-pro*) (Zhang et al., 2026b)* | | | |
| Gemini-3-pro | No Memory | 0.357 | – |
| Gemini-3-pro | Pass@10 | – | 0.524 |
| Gemini-3-pro | Reflexion | – | 0.530 |
| Gemini-3-pro | RAG | 0.500 | 0.548 |
| Gemini-3-pro | Self-RAG | 0.488 | 0.548 |
| Gemini-3-pro | Mem0 | 0.512 | 0.560 |
| Gemini-3-pro | MemP | 0.528 | 0.582 |
| Gemini-3-pro | MemRL | 0.573 | 0.613 |
| *APEX-EM (ours, Claude Opus 4.5)* | | | |
| Opus 4.5 | No Memory (A0) | 0.252 | – |
| Opus 4.5 | Semantic, no judge (A1) | 0.194 | 0.479 |
| Opus 4.5 | Semantic + judge + iter. (A3) | 0.404 | 0.529 |
| Opus 4.5 | Semantic + rich + iter. (R1) | 0.456 | 0.523 |
| Opus 4.5 | Sem. + Structural (S2) | 0.468 | 0.523 |
| Opus 4.5 | Structural (S1) | 0.480 | 0.533 |
| *APEX-EM backbone scaling (comparable to Gemini-3-pro)* | | | |
| Opus 4.6 | No Memory (A0) | 0.366 | – |
| Opus 4.6 | Structural (S1) | 0.610 | 0.640 |
| Opus 4.7 | No Memory (A0) | 0.394 | – |
| Opus 4.7 | Structural (S1) | 0.684 | **0.712** |

Table 8: Multi-domain knowledge reasoning results on HLE (Phan et al., 2025) (10 epochs). MemRL baselines from Zhang et al. (2026b). *Gemini-3-pro discontinued March 2026. Opus 4.6/4.7 baselines (36.6%/39.4%) are comparable to Gemini-3-pro (35.7%). APEX-EM with Opus 4.7 achieves 71.2% CSR, exceeding MemRL's 61.3% by +9.9pp.

Opus checkpoints of increasing capability. Table 12 shows that the memory gain is remarkably consistent (+27–32pp) across all three backbones, confirming that APEX-EM's benefit is additive to model capability rather than compensating for a specific model's weakness. Opus 4.6 (baseline 36.6%) provides the most controlled comparison to MemRL's Gemini-3-pro (35.7%), with only 0.9pp baseline difference. Under this near-identical baseline condition, APEX-EM achieves 64.0% CSR vs MemRL's 61.3%, a +2.7pp advantage attributable to the framework rather than backbone differences.

## G Iteration Efficiency

Beyond improving success rates, memory accumulation reduces computational cost by decreasing self-correction iterations. On BCB (max_iterations=3), iteration counts decrease by 11–17% from E1 to E10. On KGQA (max_iterations=10), the reduction is dramatic: 37–43%, with A5 saving nearly 2 iterations per task on average (4.40→2.50). On ALFWorld, step counts decrease by 14–17% despite near-saturated success rates.

| | | OS Task | | DB Task | |
|---|---|---|---|---|---|
| Backbone | Method | Last SR | CSR | Last SR | CSR |
| *MemRL baselines (GPT-4o-mini) (Zhang et al., 2026b)* | | | | | |
| GPT-4o-mini | No Memory | 0.674 | – | 0.860 | – |
| GPT-4o-mini | Pass@10 | – | 0.756 | – | 0.928 |
| GPT-4o-mini | RAG | 0.690 | 0.700 | 0.914 | 0.916 |
| GPT-4o-mini | Self-RAG | 0.646 | 0.732 | 0.891 | 0.898 |
| GPT-4o-mini | Mem0 | 0.670 | 0.702 | 0.920 | 0.926 |
| GPT-4o-mini | MemP | 0.736 | 0.742 | 0.960 | 0.966 |
| GPT-4o-mini | MemRL | 0.788 | 0.804 | 0.960 | 0.972 |
| *APEX-EM (ours, Claude Sonnet 4.5)* | | | | | |
| Sonnet 4.5 | No Memory (A0) | 0.720 | 0.740 | 0.944 | 0.950 |
| Sonnet 4.5 | Semantic + judge (A2) | 0.750 | 0.812 | 0.956 | 0.956 |
| Sonnet 4.5 | Semantic + judge + iter. (A3) | 0.786 | 0.812 | 0.972 | 0.982 |
| Sonnet 4.5 | Structural (S1) | 0.770 | 0.814 | 0.972 | 0.988 |
| Sonnet 4.5 | Sem. + Struct. + Opus (A5) | 0.780 | 0.806 | 0.956 | 0.986 |
| Sonnet 4.5 | Semantic + Structural (S2) | 0.788 | **0.820** | **0.978** | **0.988** |
| *APEX-EM (ours, GPT-4o-mini) — same backbone as MemRL* | | | | | |
| GPT-4o-mini | APEX-EM A5 | 0.788 | 0.818 | 0.963 | 0.982 |

Table 9: Lifelong Agent Bench (Zheng et al., 2025) results: OS interaction and database tasks (500 tasks/epoch, 10 epochs). MemRL baselines from Zhang et al. (2026b) with GPT-4o-mini backbone. Last SR = final epoch success rate; CSR = cumulative success rate (unique tasks solved at least once across all epochs). APEX-EM with GPT-4o-mini backbone beats MemRL on CSR (+1.4pp OS, +1.0pp DB) with same backbone.

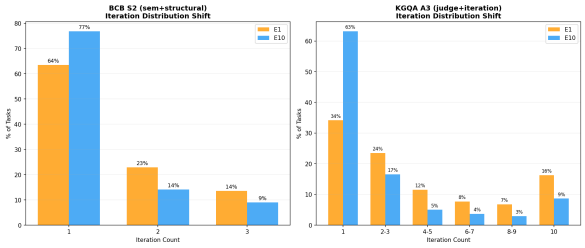


Figure 3: Iteration distribution shift from E1 to E10. **Left:** BCB S2—first-attempt solutions increase from 63.5% to 76.8% (+13.3pp). **Right:** KGQA A3—first-attempt solutions increase from 34.2% to 63.1% (+28.9pp).

The mechanism: in early epochs, iteration does most of the work (no memory yet); in later epochs, memory provides sufficient context that the initial generation is already correct, making iteration increasingly unnecessary.

### G.1 Cost per Correct Task

Table 14 quantifies the cost efficiency gains from memory accumulation by computing the amortized LLM API cost per *correctly solved* task. We estimate per-iteration cost using Sonnet 4.5 pricing ($3/MTok input, $15/MTok output) with average context sizes of ∼4K input + 1.5K output tokens for BCB and ∼5K input + 1K output tokens for KGQA, yielding approximately $0.035/iteration (BCB) and $0.030/iteration (KGQA). The cost per

| Backbone | Method | Last SR | CSR |
|---|---|---|---|
| *MemRL baselines (GPT-5-mini) (Zhang et al., 2026b)* | | | |
| GPT-5-mini | No Memory | 0.777 | – |
| GPT-5-mini | Pass@10 | – | 0.928 |
| GPT-5-mini | RAG | 0.907 | 0.935 |
| GPT-5-mini | Self-RAG | 0.907 | 0.962 |
| GPT-5-mini | Mem0 | 0.894 | 0.969 |
| GPT-5-mini | MemP | 0.885 | 0.919 |
| GPT-5-mini | MemRL | 0.949 | 0.981 |
| *APEX-EM (ours, Claude Sonnet 4.5)* | | | |
| Sonnet 4.5 | No Memory (A0) | 0.882 | 0.939 |
| Sonnet 4.5 | Semantic + judge (A2) | 0.998 | 0.998 |
| Sonnet 4.5 | Semantic + judge + iter. (A3) | 0.952 | 0.993 |
| Sonnet 4.5 | Structural (S1) | 0.965 | 0.991 |
| Sonnet 4.5 | Sem. + Struct. + Opus (A5) | **0.997** | **0.9996** |
| Sonnet 4.5 | Semantic + Structural (S2) | 0.955 | 0.992 |
| *APEX-EM (ours, GPT-5-mini) — same backbone as MemRL* | | | |
| GPT-5-mini | APEX-EM A5 (E1) | 0.991 | 0.999 |

Table 10: ALFWorld (Shridhar et al., 2021) embodied navigation results. MemRL baselines from Zhang et al. (2026b) with GPT-5-mini backbone. Last SR = final epoch success rate; CSR = cumulative success rate. A5 (Opus judge) achieves near-perfect 99.96% CSR with Sonnet 4.5. GPT-5-mini APEX-EM achieves 99.1% SR in first epoch, confirming saturation.

| Configuration | HLE SR | CSR |
|---|---|---|
| A0: No Memory (1 epoch) | 25.2% | – |
| A1: Semantic, no judge | 19.4% | 47.9% |
| A3: Semantic + judge + iter. | 40.4% | 52.9% |
| R1: Semantic + rich + iter. | 45.6% | 52.3% |
| S2: Sem. + Structural | 46.8% | 52.3% |
| S1: Structural | **48.0%** | **53.3%** |

Table 11: HLE ablation study with Claude Opus 4.5 backbone. All ablations complete (10 epochs). S1 (structural retrieval) leads at 48.0% SR / 53.3% CSR; A1 (no judge, no iteration) shows high CSR (47.9%) despite low SR (19.4%), indicating broad exploration without convergence.

| Backbone | Baseline (A0) | APEX-EM S1 CSR | Δ |
|---|---|---|---|
| Opus 4.5 | 25.2% | 53.3% | +28.1pp |
| Opus 4.6 | 36.6% | 64.0% | +27.4pp |
| Opus 4.7 | 39.4% | 71.2% | +31.8pp |
| MemRL (Gemini-3-pro*) | 35.7% | 61.3% | +25.6pp |

Table 12: HLE backbone scaling. APEX-EM S1 (structural retrieval) with three Opus checkpoints. CSR = tasks solved at least once across 10 epochs (frugal mode). *Gemini-3-pro discontinued March 2026. Memory gain is additive to model capability (+27–32pp).

correct task is then: $\text{cost}_{\text{correct}} = (\bar{J} \times c_{\text{iter}})/\text{SR}$, where $\bar{J}$ is the average iteration count and SR is the success rate.

The dual benefit of memory is evident: iteration reduction (*fewer calls per task*) combines multiplicatively with success rate improvement (*fewer wasted attempts*), yielding 40–55% cost reduction by epoch 10. The retrieval overhead itself is negligible: Titan Embed V2 at \$0.02/MTok adds <\$0.001/query (<1% of iteration cost), and graph traversal is a local operation with sub-millisecond latency. This demonstrates that APEX-EM's memory overhead is amortized within the first few tasks of each epoch.

## H Implementation Architecture

The `PRGIIWorkflow` class orchestrates the five phases via dependency injection. It accepts a `PRGIIConfig` (phase toggles and parameters) and five phase objects. The `execute()` method runs phases in sequence: Plan → Retrieve → [Generate ↔ Iterate]$^J$ → Ingest. Each phase is skipped when disabled, enabling ablation studies by toggling individual components.

Four extension points enable domain specialization: (1) **DomainAdapter Protocol**—defines entity/property/procedural-step extraction methods; (2) **PromptRegistry**—maps task types to domain-specific prompts for each phase; (3) **Validators**—domain-specific verification for Iteration ($\mathcal{V}_d$) and Ingest ($\mathcal{O}_d$) phases; (4) **PRGIIConfig**—boolean toggles for each phase plus max_iterations and quality_threshold.

## I Entity Graph Visualizations

Figures 4–6 illustrate Entity Graphs from each benchmark domain. All share the same 4 node types and 5 edge types but with domain-specific instantiations.

## J Judge Feedback Without Answer Leakage

A critical design property of APEX-EM is that the Judge LLM evaluates the agent's *proof-of-work*—the execution trace, reasoning chain, and generated artifacts—without leaking the ground-truth answer into the feedback stored in memory. The judge has access to the oracle answer for scoring, but the feedback text identifies *where the reasoning went wrong* rather than *what the correct answer is*.

**Example 1: Spin Bordism (Algebraic Topology).** *Task*: Compute the reduced 12th dimensional Spin

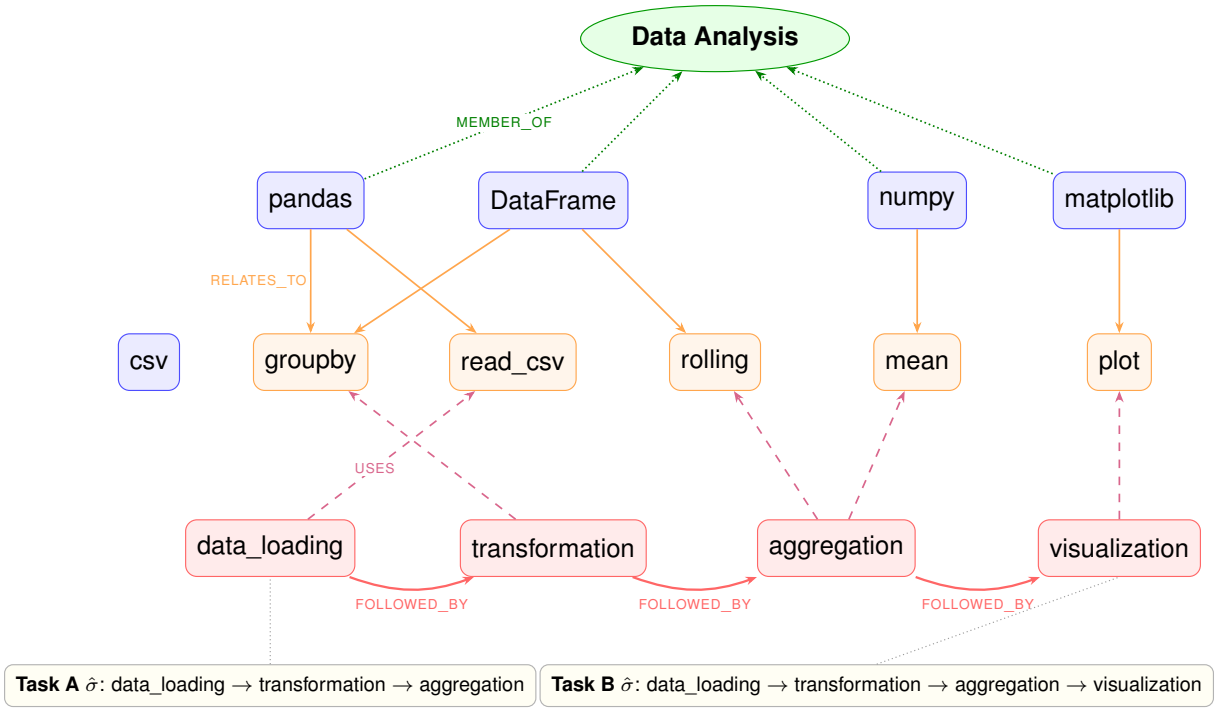


Figure 4: BCB Entity Graph. Tasks A and B share entities and procedural steps, enabling structural retrieval despite different descriptions.

bordism of the classifying space of $G_2$. *Gold*: $\mathbb{Z}^5$.

In epoch 1, the agent computed $\mathbb{Z}^3$ using the correct Atiyah-Hirzebruch spectral sequence approach but with incorrect homology groups. The judge feedback stated: "*homology groups of $BG_2$ are computed incorrectly*" and directed: "*carefully compute the integral homology using the known cohomology ring structure.*" The feedback identifies the specific computational step that failed without revealing $\mathbb{Z}^5$. In epoch 2, with this failed experience retrieved as a negative example, the agent arrived at the correct answer.

**Example 2: Multi-Part Cipher Puzzle.** *Task*: A multi-part puzzle involving logical depth, a Gell-Mann quote, GELU authorship, ROT13 cipher, and planetary mass comparison. *Gold*: "yeyo".

The judge feedback identified: "*identification of 'crypticity' as the reciprocal concept needs verification*" and "*the quote about MIT and options needs more careful analysis.*" The feedback pinpointed which sub-parts were wrong without revealing the final answer.

**Example 3: Elliptic Curve Torsion (Number Theory).** *Task*: Largest order of a non-cyclic torsion subgroup of an elliptic curve over $\mathbb{Q}(\sqrt{-3})$. *Gold*: 18.

The agent claimed $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/14\mathbb{Z}$ (order 28). The judge feedback stated: "*incorrectly claims $\mathbb{Z}/2\mathbb{Z} \times \mathbb{Z}/14\mathbb{Z}$ occurs without proper verification*" and directed: "*consult the specific classification results for $\mathbb{Q}(\sqrt{-3})$ more carefully.*" In epoch 2, the agent correctly identified $\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/6\mathbb{Z}$ (order 18).

## K Architectural Comparison with MemRL

Two architectural differences beyond the memory representation distinguish APEX-EM from MemRL: (1) APEX-EM makes validation an explicit phase within the iteration loop—the agent self-verifies each attempt against domain-specific criteria ($\mathcal{V}_d$), producing structured feedback that guides the next attempt, rather than relying solely on implicit LLM self-assessment; and (2) the Teacher model (Opus 4.5) independently evaluates completed traces at the commit phase (Ingest), providing multi-dimensional feedback that determines whether experiences enter memory. The ablation in Table 3 isolates these contributions: A1 (memory with binary reward, no teacher) shows gains attributable purely to memory retrieval, the A1→A3 gap quantifies explicit self-verification, and the A2→A5 gap quantifies teacher-quality feedback at commit.

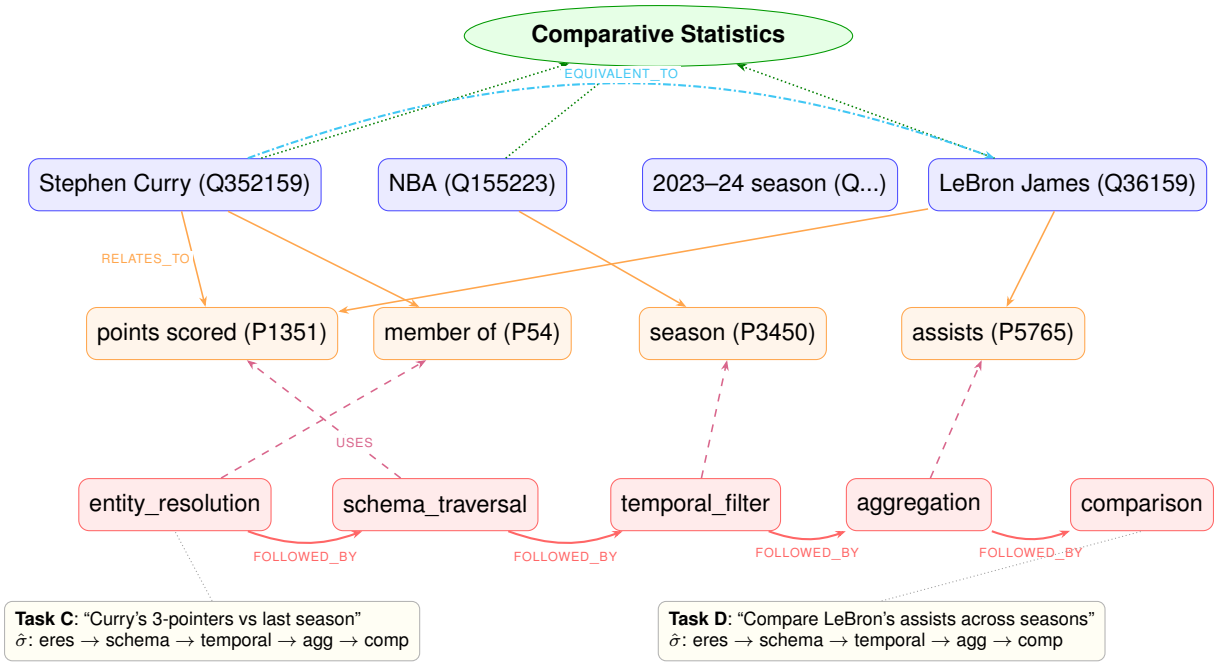


Figure 5: KGQA Entity Graph. Tasks C and D produce identical structural signatures despite different entities, enabling cross-entity structural retrieval.

## L Self-Improvement Mechanisms

PRGII improves over single-shot generation through two independent mechanisms:

**Mechanism 1: Cross-Epoch Memory Retrieval.** BCB task 15 (execute shell commands from CSV, save outputs to files): the single-shot LLM generates reasonable code but names output files incorrectly. The oracle test suite expects a different naming convention that cannot be inferred from the task description alone. In epoch 2, the failed experience from epoch 1 is retrieved as a negative example with the oracle's error message, and the agent reverse-engineers the correct naming convention.

**Mechanism 2: Within-Epoch Self-Correction Iteration.** BCB task 89 (outlier removal via Z-score with StandardScaler): the LLM generates code using strict inequality (>) where the oracle expects inclusive (>=). Over 3 iterations: (1) code generated with >, (2) reviewer catches the discrepancy, (3) code updated to >=, tests pass. The self-critique loop catches a subtle off-by-one bug without any memory involvement.

These mechanisms are complementary: memory retrieval transfers knowledge across epochs, while iteration catches bugs within a single execution. The ablation results confirm this—A1 (memory, no iteration) and A3 (memory + iteration) both improve over A0, with gains partially additive in the full pipeline.

**Why Iteration Enables KGQA Transfer.** Consider a training question about Stephen Curry's statistics and a transfer question about LeBron James's career requiring the same properties but different entity QIDs. Memory retrieval surfaces the Curry example as structurally similar. With iteration, the agent can generate an initial query using the retrieved pattern, execute it, and refine when results are unexpected. Without iteration, the agent must get the query right on the first attempt—and subtle differences between entities often cause failure. This explains the stark A1/A2 vs A3+ gap in transfer results.

## M Code and Data Availability

We will release the core APEX-EM implementation including the PRGII workflow orchestrator, PKG construction and retrieval modules, structural signature extraction, and domain adapter interfaces upon publication. Benchmark-specific configurations, prompt templates, and evaluation harnesses will be included to enable reproduction of all reported experiments.

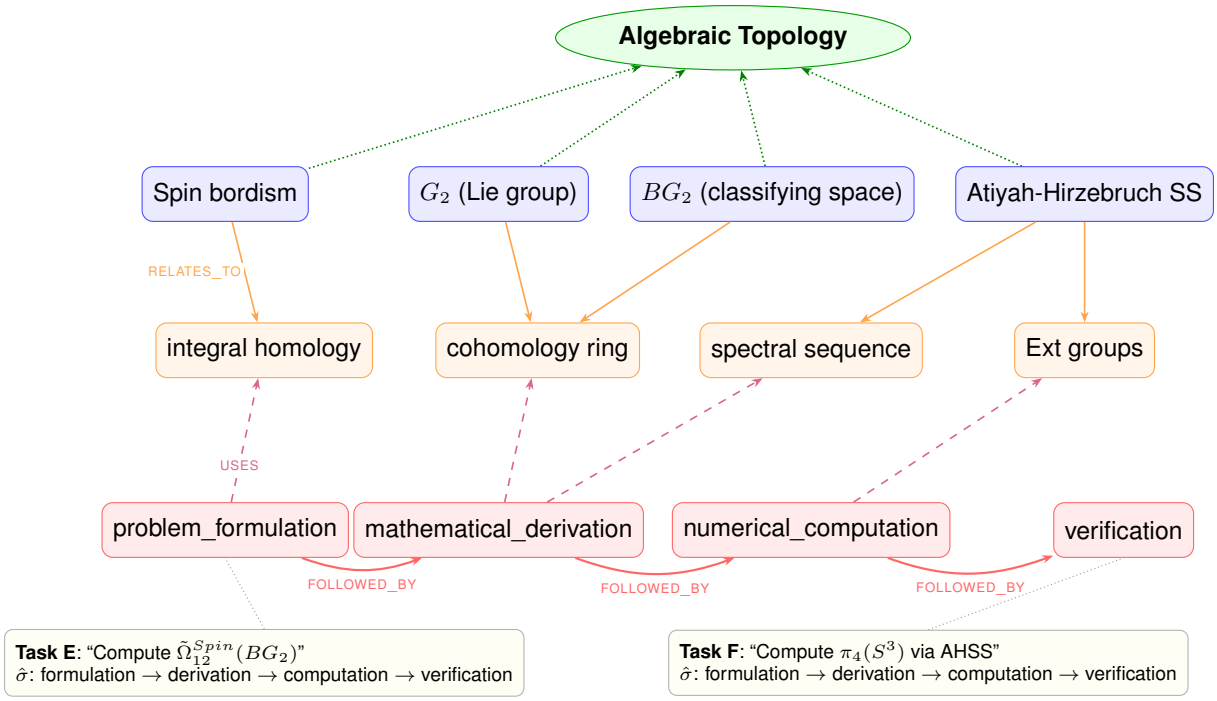


Figure 6: HLE Entity Graph. Tasks E and F share the same procedural pattern and the Atiyah-Hirzebruch spectral sequence entity, enabling structural retrieval across different algebraic topology problems.

| Config | E1 Avg | E10 Avg | Δ | % Red. |
|---|---|---|---|---|
| *BigCodeBench (max_iterations=3)* | | | | |
| A3: Semantic + judge + iter. | 1.63 | 1.36 | −0.27 | 17% |
| R1: Semantic + rich + iter. | 1.59 | 1.35 | −0.24 | 15% |
| S1: Structural | 1.50 | 1.30 | −0.20 | 13% |
| S2: Semantic + Structural | 1.50 | 1.32 | −0.18 | 12% |
| A5: Sem. + Struct. + Opus | 1.51 | 1.35 | −0.16 | 11% |
| *KGQAGen-10k (max_iterations=10)* | | | | |
| A5: Sem. + Struct. + Opus | 4.40 | 2.50 | −1.90 | 43% |
| S2: Semantic + Structural | 4.42 | 2.73 | −1.69 | 38% |
| R1: Semantic + rich + iter. | 4.19 | 2.61 | −1.58 | 38% |
| A3: Semantic + judge + iter. | 4.12 | 2.59 | −1.53 | 37% |
| S1: Structural | 4.02 | 2.52 | −1.50 | 37% |
| *Lifelong OS (max_steps=15, agent turns)* | | | | |
| S1: Structural | 4.52 | 3.99 | −0.53 | 12% |
| A3: Semantic + judge + iter. | 4.42 | 4.07 | −0.35 | 8% |
| R1: Semantic + rich + iter. | 4.53 | 4.17 | −0.35 | 8% |
| S2: Semantic + Structural | 4.44 | 4.10 | −0.33 | 8% |
| A5: Sem. + Struct. + Opus | 4.42 | 4.17 | −0.25 | 6% |
| *ALFWorld (max_steps=50, agent actions)* | | | | |
| A3: Semantic + judge + iter. | 9.53 | 7.91 | −1.62 | 17% |
| A5: Sem. + Struct. + Opus | 9.04 | 7.80 | −1.24 | 14% |

Table 13: Efficiency improvement from epoch 1 to epoch 10 (or last completed epoch). As memory accumulates, the agent requires fewer steps/iterations to solve tasks. BCB/KGQA measure self-correction iterations; LLB/ALFWorld measure agent environment steps. KGQA shows the largest reduction (37–43%) due to high max_iterations budget. ALFWorld shows 14–17% step reduction despite near-saturated SR, indicating memory enables more direct solutions. LLB DB shows minimal reduction (2–4%) as tasks are already solved in ∼3 steps.

| | | Iterations | | | Cost per Correct Task | | |
|---|---|---|---|---|---|---|---|
| Benchmark | Method | E1 | E10 | Δ% | E1 | E10 | Δ% |
| BCB | No Memory | 1.63 | – | – | \$0.117 | – | – |
| | APEX-EM (S2) | 1.50 | 1.32 | −12% | \$0.108 | \$0.065 | −40% |
| | APEX-EM (A5) | 1.51 | 1.35 | −11% | \$0.107 | \$0.064 | −40% |
| KGQA | No Memory | 4.12 | – | – | \$0.497 | – | – |
| | APEX-EM (S2) | 4.42 | 2.73 | −38% | \$0.304 | \$0.151 | −50% |
| | APEX-EM (A5) | 4.40 | 2.50 | −43% | \$0.304 | \$0.137 | −55% |

Table 14: Cost efficiency analysis. **Iterations**: average self-correction iterations per task. **Cost per Correct Task**: estimated LLM API cost amortized over successful tasks only, computed as (avg_iters × cost_per_call) / success_rate. Pricing: Sonnet 4.5 input \$3/MTok, output \$15/MTok; avg context ∼4K input + 1.5K output tokens per iteration. E1 = first epoch (no memory); E10 = tenth epoch (full memory). Cost reduction combines both iteration reduction and success rate improvement.

| ID | Retrieval | Iterate | Judge | Feedback | Judge Model |
|---|---|---|---|---|---|
| A0 | — | — | — | — | — |
| A1 | Semantic | — | — | Binary | Sonnet |
| A2 | Semantic | — | ✓ | Rich | Sonnet |
| A3 | Semantic | ✓ | ✓ | Binary | Sonnet |
| R1 | Semantic | ✓ | ✓ | Rich | Sonnet |
| S1 | Structural | ✓ | ✓ | Rich | Sonnet |
| S2 | Semantic + Structural | ✓ | ✓ | Rich | Sonnet |
| A5 | Semantic + Structural | ✓ | ✓ | Rich | Opus |

Table 15: Ablation configurations. **Retrieval**: which retrieval channels are active (Semantic = embedding similarity; Structural = signature matching via entity-centric index; Semantic + Structural = both). **Iterate**: iterative refinement on failed tasks. **Judge**: quality judge on ingested plans. **Feedback**: what the agent sees from retrieved plans (Binary = pass/fail only; Rich = quality scores + rationale). **Judge Model**: LLM used for quality assessment (Sonnet = Claude Sonnet 4.5; Opus = Claude Opus 4.5). All configs with retrieval also include planning and ingest phases.