# CAUSAL EXPERIENCE INSTANTIATION: GROUNDING PROCEDURAL MEMORY WITH TEMPORAL GRAPHS FOR TOOL-USE AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Tool-use agents need to reuse past procedures without treating task-specific values from those executions as facts about a new task. Retrieving a successful trajectory can import stale identifiers, while abstracting it into a workflow leaves open when and where its arguments should be grounded. We introduce *Causal Experience Instantiation*, which couples a Static Procedural Bank of abstract workflow graphs with an online Dynamic Fact Ledger. The bank retains action and argument dependencies without episode-specific values. As the agent acts, the ledger adds typed facts from the current task to a temporal graph, recording when each value became available and which observation supplied it. At each decision, a Presence Head estimates whether evidence for a required argument is available; a Source Ranking Head selects among compatible observed sources when it is. Otherwise, the readout leaves the argument unresolved rather than supplying an unsupported value. On APPWORLD, the method completes 82/168 tasks versus ReMe's 73/168, while reducing the invalid-call rate from 1.10% to 0.21%. Relative to ExpeL, gains also appear on ALFWORLD, a stratified SCIENCEWORLD panel, and all five evaluated AppWorld language-model backbones. Component ablations show lower AppWorld success when either memory path or the evidence-checking readout is removed.

## 1 INTRODUCTION

Large language models (LLMs) increasingly operate as autonomous interactive agents capable of orchestrating multi-step workflows across complex software environments (OpenAI, 2023; Yao et al., 2023b; Schick et al., 2023; Zhou et al., 2024). To solve realistic long-horizon tasks, an agent cannot rely solely on static pre-trained knowledge; it must accumulate and transfer procedural experience across episodes (Wang et al., 2023; Shinn et al., 2023; Park et al., 2023). When faced with a complex objective—such as synchronizing calendar events and issuing payments—an effective agent should retrieve demonstrated workflows and adapt them to the current environment.

Cross-episode learning can take several forms: EXPEL retrieves successful demonstrations (Zhao et al., 2024), AWM induces reusable workflows (Wang et al., 2024), and other systems maintain longer-lived memories (Packer et al., 2023; Majumder et al., 2023). These sources give LLMs useful guidance, but a procedure or value learned from a past episode may not match the live state of a closed-loop environment such as APPWORLD (Trivedi et al., 2024). We study two resulting failure modes:

Past traces are laden with episode-contingent literals (foreign keys, phone numbers, session tokens). When an agent observes these concrete values in retrieved demonstrations, it may copy a stale literal (e.g., an obsolete `recipient_id="1042"`) into a live API call despite conflicting current evidence. Lacking variable abstraction, flat retrieval conflates reusable workflow logic with incidental demonstration constants.

Demonstrations show completed paths in which prerequisite arguments have already been found. They may suggest where to search, but do not explicitly tell the agent *whether* the needed evidence is present in its current execution prefix. The agent may therefore make a state-changing call before checking a prerequisite.



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.

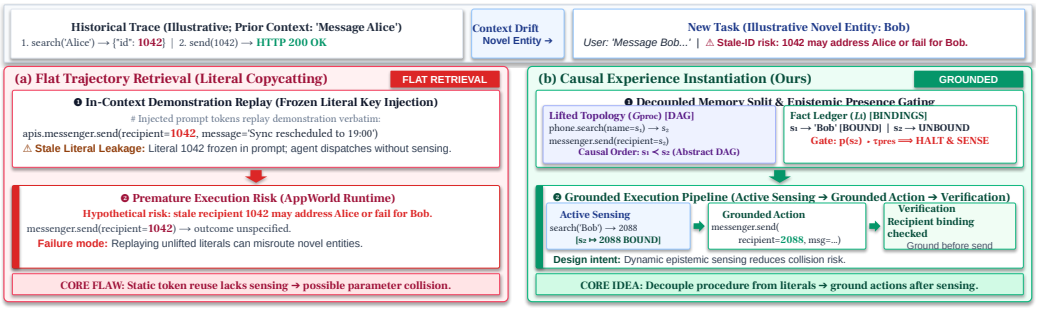



Figure 1: Illustrative stale-ID failure mode and grounded alternative. A prior Alice trace contains an episode-specific ID; copying it into a Bob task may misaddress the action or fail. Our method retains the abstract search-before-send dependency, checks current evidence, and binds an observed ID before dispatch. The names and IDs are schematic rather than a recorded rollout; success is not guaranteed by the diagram.

We propose **Causal Experience Instantiation** to address this mismatch (Figure 1). It keeps two kinds of information separate: a **Static Procedural Bank** turns past trajectories into abstract Temporal Experience Graphs (TEGs) with typed slots, while a **Dynamic Fact Ledger** records values and provenance from the current task. At each decision, a **Presence Head** checks whether a slot has current-task evidence and can issue an `UNRESOLVED` directive if not. A **Source Ranking Head** then chooses among observed candidates when evidence is available. The LLM decides whether and how to act on the directive. On AppWorld (Trivedi et al., 2024) (168 tasks across 56 base task groups), the full method solves 82 tasks (48.81%), compared with 73 (43.45%) for ReMe, the strongest evaluated external baseline, 70 (41.67%) for ExpeL, and 68 (40.48%) for a matched static–dynamic control without the final readout. Its invalid-call rate is 0.21% versus ReMe's 1.10% and ExpeL's 0.72%. On ALFWorld, it solves 78 of 134 tasks versus 70 for AdaMEM and 66 for ExpeL; on a stratified 90-episode ScienceWorld panel, its mean score is 42.1 versus ExpeL's 34.7. Across five tested language models, AppWorld improvements over ExpeL range from +4.76 to +7.14 percentage points.

In summary, this work makes four principal contributions:

- **Failure Mode Analysis**: We characterize stale-value copying and execution without current evidence; on AppWorld, confirmed stale-value copies fall from 14 with ExpeL to 2 with our method.
- **Causal Experience Instantiation**: We separate episodic memory into an abstract Static Procedural Bank and a Dynamic Fact Ledger.
- **Decoupled Neural Readout**: We design a dual-head module combining epistemic refusal with grounded entity ranking.
- **Empirical Evaluation**: Across AppWorld, ALFWorld, and ScienceWorld, we compare against experiential-memory baselines and component controls; on AppWorld, the full method gains +5.36 pp over the strongest evaluated external baseline, ReMe.

## 2 Related Work

**Tool-Augmented LLM Agents and Benchmarks.** Equipping LLMs with tool execution supports interactive reasoning (Schick et al., 2023; Yao et al., 2023b). Prior work addresses API selection and generation (Patil et al., 2024; Qin et al., 2024; Xu et al., 2023), RESTful composition (Song et al., 2023), and orchestration across specialized models and tools (Shen et al., 2023; Lu et al., 2023). Stateful-agent benchmarks span web navigation (WebArena, Mind2Web, and VisualWebArena) (Zhou et al., 2024; Deng et al., 2023; Koh et al., 2024), conversational interaction ($\tau$-bench) (Yao et al., 2024a), and persistent software states (SWE-bench, InterCode, and OSWorld) (Jimenez et al., 2024; Yang et al., 2023; Xie et al., 2024). AppWorld (Trivedi et al., 2024) is particularly relevant here because its 9 daily apps use relational databases and strict API contracts, requiring exact entity grounding.

**Experiential Learning, Memory, and Reflection.** Agent architectures use episodic reflection (Shinn et al., 2023), executable skill libraries (Wang et al., 2023), and memories that can be paged, recalled, or revised (Packer et al., 2023; Zhong et al., 2024; Park et al., 2023; Majumder et al., 2023; Liu et al., 2023b). Other systems use retrospective policy optimization (Yao et al., 2024b) or text-based knowledge and memory in open-world play (Zhu et al., 2023). For procedural reuse, EXPEL (Zhao et al., 2024) retrieves successful demonstration traces and AWM (Wang et al., 2024) induces offline workflows. These artifacts provide useful procedures, but an argument copied from a past episode still needs support in the current observation prefix; our method checks current-task evidence and ranks its sources. More recent approaches refine the stored experience itself: ACE evolves a procedural playbook (Zhang et al., 2025), while REME distills and revises procedural memories (Cao et al., 2025). EXPGRAPH organizes reusable experiences in a utility-aware graph (Feng et al., 2026), and EMG derives graph-based corrections from failed and successful trajectories (Wang et al., 2026). ADAMEM instead synthesizes step-conditioned strategies from offline trajectory memory (Zhang et al., 2026). Our focus is complementary: whether a reusable procedure can be instantiated with an argument whose source is observable in the current task, at the decision where that argument is needed.

**Planning, Program Synthesis, and Grounding.** Autonomous execution relies on sound action decomposition. Beyond Chain-of-Thought (Wei et al., 2022), tree and graph search methods (Yao et al., 2023a; Besta et al., 2024) enable non-linear reasoning paths, while program-aided frameworks (Gao et al., 2023; Chen et al., 2023) interleave language generation with code execution, and LLM+P (Liu et al., 2023a) leverages PDDL planners. In embodied domains, SayCan (Ichter et al., 2022) combines language-model suggestions with affordance scores, while Inner Monologue (Huang et al., 2022) feeds observations and success signals back into language-based planning. For digital API calls, we instead ask whether each argument can be traced to evidence observed in the current task.

**Procedural Grounding and Dynamic Graphs.** The separation of reusable structure from task-specific observations is related to causal and invariant representation learning (Pearl, 2009; Schölkopf et al., 2021; Peters et al., 2016; Arjovsky et al., 2019); causal NLP provides a broader account of such questions (Feder et al., 2022). In our formulation, the abstract procedural graph is the reusable part of an experience, while task-specific entity references are supplied by live evidence. Dynamic graph methods such as TGN (Rossi et al., 2020), DyRep (Trivedi et al., 2019), and TGAT (Xu et al., 2020) motivate encoding timestamped interactions, though they differ in whether they use explicit node memory. Selective prediction (Geifman & El-Yaniv, 2017) and uncertainty-aware assistance (Ren et al., 2023) motivate allowing the readout to abstain when a slot lacks observed support; partial-observability models provide background for the value of gathering information (Kaelbling et al., 1998).

## 3 PRELIMINARIES AND PROBLEM FORMULATION

Autonomous tool use in stateful environments requires balancing exploration against grounded parameter selection. An ungrounded argument can cause an invalid call or an unintended state change. We formalize this interactive setting as a partially observable Markov decision process (POMDP) and state the prefix-causal grounding criterion used by our method. Table 1 summarizes the primary mathematical notations used throughout the paper.

### 3.1 INTERACTIVE TOOL-USE AS A POMDP

We formulate interactive tool use as a POMDP (Kaelbling et al., 1998) defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \Omega, \rho \rangle$, where $\mathcal{S}$ is the environment state, $\mathcal{A}$ contains tool or environment actions, and $\mathcal{O}$ includes structured and natural-language observations. The one-step reward is $\rho$; $\mathcal{R}(f)$ below denotes the argument roles of action type $f$. At step $t$, the agent emits $a_t = (f_t, \theta_t)$, comprising an action type $f_t \in \mathcal{F}$ and an argument assignment:

$$\theta_t : \mathcal{R}(f_t) \to \mathcal{V}, \tag{1}$$

where $\mathcal{R}(f_t)$ denotes the argument roles of action $f_t$, and $\mathcal{V}$ is the domain of values. The environment transitions according to $\mathcal{T}(s_{t+1} \mid s_t, a_t)$ and emits the next observation $o_{t+1} \in \mathcal{O}$ according to $\Omega(o_{t+1} \mid s_{t+1}, a_t)$, with $o_0$ denoting the initial observation.

Table 1: Notation used in the method.

| Notation | Meaning |
|---|---|
| $\mathcal{H}_t$ | Goal and visible interaction prefix before action $t$ |
| $a_t = (f_t, \theta_t)$ | Action type and its parameter assignment |
| $\mathcal{G}_{\text{static}}$ | Abstract procedural graph retrieved from training experiences |
| $\mathcal{L}_t$ | Task-local ledger of observed values and provenance |
| $\mathcal{G}_t^{\text{dyn}}, \mathbf{h}_t$ | Online temporal graph and its contextual state |
| $\tau_{\text{pres}}$ | Decision threshold for the Presence Head |
| $\Theta_{\text{pres}}, \Theta_{\text{rank}}$ | Parameters of the two neural readout heads |

### 3.2 The Prefix-Causal Grounding Constraint

Task-specific entities (e.g., user IDs, objects, or locations) cannot be soundly copied from a different episode. Let the visible prefix before action $t$ be $\mathcal{H}_t = (g, o_0, a_0, o_1, \ldots, a_{t-1}, o_t)$, including the current goal, initial observation, and all observations available before $a_t$. For a dynamic argument $r \in \mathcal{R}(f_t)$ requiring current-task evidence, a supported binding $v_r$ satisfies the **prefix-causal grounding criterion**:

$$v_r \in \text{Derive}(\text{ExtractFacts}(\mathcal{H}_t)) \,. \tag{2}$$

Here Derive admits values explicitly supplied in the current goal and those computed from current observations; fixed schema constants need no episode-specific evidence. A value copied only from a past task does not satisfy the criterion. This criterion guides the readout, but it does not certify every emitted agent action.

## 4 Causal Experience Instantiation

**Causal Experience Instantiation** separates reusable procedures from values that belong to one task. A *Static Procedural Bank* stores abstract workflows, while an online *Dynamic Fact Ledger* records current-task evidence. At each step, a two-head readout checks whether a required value has been observed and, if so, selects a source for it. The method has four parts: (1) abstracting past procedures into graphs, (2) updating the fact ledger, (3) encoding the evolving task graph, and (4) checking evidence and ranking sources before an action.

### 4.1 Static Procedural Bank: Temporal Experience Graphs

To prevent literal leakage, historical successful trajectories are converted into canonicalized **Temporal Experience Graphs** (TEGs). Formally, a TEG is a directed acyclic graph:

$$\mathcal{G}_{\text{static}} = (\mathcal{V}_{\text{proc}}, \mathcal{E}_{\text{proc}}), \tag{3}$$

where each node $u \in \mathcal{V}_{\text{proc}}$ represents an abstract API invocation $f_u$ parameterized by a set of formal variable slots $\mathcal{S}_u = \{s_1, \ldots, s_m\}$. A directed edge $(u, v) \in \mathcal{E}_{\text{proc}}$ records an observed procedural dependency: $u$ precedes $v$ in a historical trace and an output or side effect recorded after $u$ is used as a prerequisite for $v$. During bank construction, an automatic abstraction pass scans historical traces: episode-specific scalar values (numbers, identifiers, names) are replaced with typed, indexed slot symbols:

$$s = (\text{role}, \text{type}, \text{scope}), \tag{4}$$

while domain constants (e.g., fixed enum values, static flags) are preserved. Thus, $\mathcal{G}_{\text{static}}$ retains the observed topological control-flow and data-flow structure of the workflow while eliminating all episode-contingent literals. These edges are used as procedural dependencies; their presence alone does not establish that $u$ causes the environment effect or that the dependency holds outside the observed traces. In the reported implementation, the bank contains 20 induced rules and retrieval returns the Top-3 most similar abstract experiences. Each benchmark builds its own bank from its training trajectories; the bank is read-only during testing.

### 4.2 Dynamic Fact Ledger: Online Causal State Tracking

During live task execution, the agent receives observations and updates task-local evidence. Before the first action, the goal enters the contextual query; values stated in the goal and facts exposed from

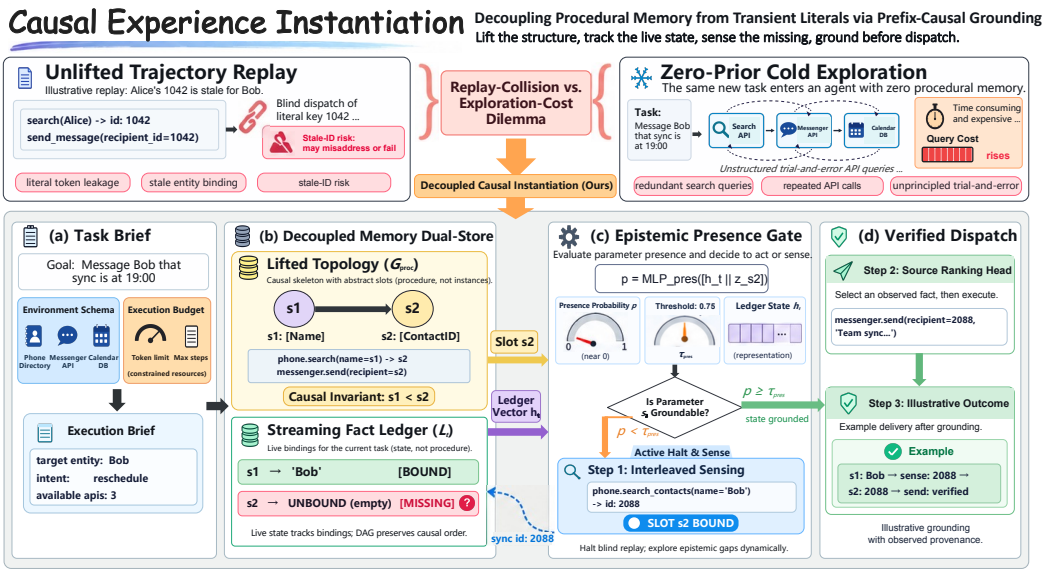



Figure 2: Causal Experience Instantiation architecture, illustrated with a schematic AppWorld task. The static procedural bank supplies abstract dependencies, while a typed, provenance-bearing dynamic ledger represents the current execution prefix. A Presence Head checks whether a required slot has a source (default threshold $\tau_{\text{pres}} = 0.75$); a separate Source Ranking Head chooses among type-compatible observed facts before dispatch. At test time the task-local graph state changes, but the encoder and readout weights remain frozen.

the initial observation $o_0$ by the benchmark adapter can enter the ledger. With no prior executed event, the event-history component is initialized to zero. This defines the initial task-local graph state $\mathcal{G}_0^{\text{dyn}}$ and contextual state $\mathbf{h}_0$; subsequent observations update these states after each action. The **Dynamic Fact Ledger** $\mathcal{L}_t$ incrementally maintains all grounded facts discovered up to step $t$:

$$\mathcal{L}_t = \{\mathbf{f}_k = (\text{val}_k, \text{type}_k, \tau_k, \text{prov}_k)\}_{k=1}^{K_t}, \tag{5}$$

where $\text{val}_k$ is the raw extracted value, $\text{type}_k \in \{\text{ID}, \text{STR}, \text{NUM}, \text{DATE}, \dots\}$ is its inferred semantic type, $\tau_k \leq t$ is the step timestamp at which it was first available, and $\text{prov}_k$ records whether it came from the goal, an action result, or another observation. Unlike unstructured dialogue history buffers, $\mathcal{L}_t$ indexes typed evidence and its provenance. We represent its online state as a temporal typed graph $\mathcal{G}_t^{\text{dyn}}$: nodes are action types, object slots, and evidence entities, while edges encode observed transitions, ownership, and the current-observation relation. On receiving $o_{t+1}$, the new facts and their provenance edges are added to the task-local graph; no edge is interpreted as a causal effect solely because it was added. Model weights remain unchanged.

Benchmark-specific adapters map each environment's action and observation schema into the shared interface: an abstract action node exposes typed argument slots, and observations add typed evidence nodes and provenance-bearing edges. The benchmark-specific training traces determine the induced static bank and learned readout, while the online ledger update operates on this common representation. Here *prefix-causal* means that each decision uses only information available up to that step; it does not identify causal effects of graph edges or actions.

### 4.3 Prefix-Causal Neural Readout Architecture

At decision step $t$, the agent selects an abstract procedure from $\mathcal{G}_{\text{static}}$ to execute next. To bind each formal slot $s \in \mathcal{S}_u$ to a live value, the system invokes our prefix-causal neural readout module. Let $\mathbf{h}_t \in \mathbb{R}^d$ represent the continuous contextual state of the execution prefix, obtained by encoding the task instruction and execution history using a dynamic graph encoder (Rossi et al., 2020; Trivedi et al., 2019; Xu et al., 2020) over $\mathcal{G}_t^{\text{dyn}}$. The frozen semantic encoder produces 384-dimensional text

vectors from the complete visible prefix using 512-token windows with 64-token overlap. A learned projection maps these vectors into the graph/readout space ($d = 128$). The task-local graph state is updated online after each observation, whereas model weights remain frozen during execution. Let $\mathbf{z}_s \in \mathbb{R}^d$ denote the learned semantic representation of slot $s$, computed from its role description and declared type using a dense Transformer text encoder (Vaswani et al., 2017; Devlin et al., 2019; Wang et al., 2020).

**Epistemic Presence Head.** Before attempting to select a value, the model estimates whether the necessary evidence exists in the live environment. The Epistemic Presence Head computes the probability that slot $s$ has a valid source entity in $\mathcal{L}_t$ with a two-layer MLP:

$$p(\text{present} \mid s, \mathbf{h}_t) = \text{MLP}_{\text{pres}}\left([\mathbf{h}_t \,\|\, \mathbf{z}_s]\right), \tag{6}$$

where the MLP outputs a sigmoid probability and $[\cdot \,\|\, \cdot]$ denotes concatenation. Motivated by selective prediction (Geifman & El-Yaniv, 2017), we use a fixed decision threshold $\tau_{\text{pres}} = 0.75$:

$$\text{State}(s) = \begin{cases} \textsc{Present}, & \text{if } p(\text{present} \mid s, \mathbf{h}_t) \geq \tau_{\text{pres}}, \\ \textsc{Unresolved}, & \text{otherwise.} \end{cases} \tag{7}$$

If $\text{State}(s) = \textsc{Unresolved}$, the readout abstains from parameter instantiation and returns an explicit prompt directive indicating that the prerequisite for slot $s$ is unobserved. The LLM agent may then choose an information-gathering action (e.g., a search or lookup), but that choice is outside the neural readout; the directive does not guarantee that the subsequent action will recover the missing evidence.

**Source Ranking Head.** When $\text{State}(s) = \textsc{Present}$, the Source Ranking Head selects a grounded fact from the ledger. We first filter $\mathcal{L}_t$ to the candidate subset $\mathcal{C}_s = \{\mathbf{f}_k \in \mathcal{L}_t \mid \text{TypeMatch}(\text{type}_k, \text{type}_s)\}$. Here TypeMatch is only a type-compatibility filter; it does not assert that a candidate is semantically correct for the slot. If this filtered set is empty, the readout falls back to the same unresolved outcome and emits the information-gathering directive rather than ranking an empty set. For each candidate $\mathbf{f}_k \in \mathcal{C}_s$, we obtain an entity embedding $\mathbf{h}_k \in \mathbb{R}^d$ from its value representation, temporal age $(t - \tau_k)$, and provenance embedding. The ranking score is computed by a two-layer MLP over the slot–candidate features:

$$\ell_k = \text{score}(s, \mathbf{f}_k, \mathbf{h}_t) = \text{MLP}_{\text{rank}}\left([\mathbf{h}_t \,\|\, \mathbf{z}_s \,\|\, \mathbf{h}_k]\right). \tag{8}$$

The implementation ranks candidates by these logits directly: $\mathbf{f}^* = \arg\max_{\mathbf{f}_k \in \mathcal{C}_s} \ell_k$. Its grounded scalar val$^*$ is then bound into slot $s$. No inference-time softmax temperature is needed for this selection.

### 4.4 Training Objective and Integration

The neural readout parameters $\Theta = \{\Theta_{\text{pres}}, \Theta_{\text{rank}}, \Theta_{\text{enc}}\}$ are trained offline on benchmark-specific historical trajectories using a unified multi-task loss. We use AdamW (learning rate $3 \times 10^{-4}$, batch size 32) for at most 15 epochs and select the checkpoint on a development split; test labels are not used for tuning. The 0.75 decision threshold was chosen using held-out training prefixes. For a slot $s$, let $\mathcal{S}_s \subseteq \mathcal{C}_s$ be the set of observed candidates supported by the training trace; more than one candidate can be valid. The Presence Head is trained with a logistic loss on the observed prefix and a paired prefix in which supporting facts are masked. When $\mathcal{S}_s$ is nonempty, the Source Ranking Head uses the set-ranking loss

$$\mathcal{L}_{\text{rank}}(s) = \log \sum_{j \in \mathcal{C}_s} e^{\ell_j} - \log \sum_{j \in \mathcal{S}_s} e^{\ell_j}, \qquad \mathcal{L}_{\text{readout}}(s) = \mathcal{L}_{\text{pres}}(s) + \mathbb{I}[\mathcal{S}_s \neq \varnothing]\, \mathcal{L}_{\text{rank}}(s), \tag{9}$$

where the two terms are summed without a separate tuned loss-weight coefficient. This objective rewards any supported source while keeping the presence decision separate from source identity. The base LLM weights remain frozen. During inference, only the task-local ledger and graph state change; the static bank, encoder, and readout weights are not updated. The readout sends either an observed-source binding or an `UNRESOLVED` directive to the LLM, which retains control of the next action.

# 5 EXPERIMENTS AND RESULTS

We ask four questions: **RQ1**, does the method improve benchmark performance on AppWorld, ALFWorld, and ScienceWorld? **RQ2**, does it reduce invalid actions, old-value copies, and changes made without evidence? **RQ3**, what do the static and dynamic paths and the two readout heads contribute? **RQ4**, are the observed AppWorld gains positive across five model backbones and the three environments for the primary backbone?

## 5.1 EXPERIMENTAL SETUP

We use the complete `test_normal` split of APPWORLD (168 tasks), the 134-task `valid_unseen` split of ALFWORLD, and a stratified 90-episode SCIENCEWORLD panel (30 task classes with three variants each). Success@1 is the fraction of tasks whose final programmatic state satisfies the benchmark oracle. For ScienceWorld, where partial progress is informative, we report the mean score on a 0–100 scale and additionally show the number of fully solved episodes. Invalid-call or invalid-action rate is the number of failed or malformed tool decisions divided by all decisions.

The controlled AppWorld comparison has eight arms: a base agent without memory; AWM, an abstract-workflow-memory baseline; ExpeL, a retrieved trajectory experience baseline; ExpeL combined with a same-scale temporal graph (same-scale TGN); a dynamic memory-only variant; a static-experience-only variant; static plus dynamic memory without the final readout; and the full method. We also evaluate ACE, ReMe, and ExpGraph on AppWorld; ExpGraph, AdaMEM, and EMG on ALFWorld. EMG was not evaluated on ScienceWorld (Appendix B.4). The benchmark-specific protocol, retrieval counts, and implementation configuration are in the appendix. Further closed-loop ablations and readout diagnostics are reported in Appendix G (A2), error categories in Appendix F (A3), cost and latency in Appendix E (A4), and the two-attempt protocol in Appendix G (A7).

## 5.2 RQ1: MAIN BENCHMARK RESULTS

Table 2 compares the eight original AppWorld arms with three recent external baselines. The full method solves 82/168 tasks (48.81%), versus 73/168 (43.45%) for ReMe, the strongest evaluated external comparator, and 70/168 (41.67%) for ExpeL: gains of 5.36 and 7.14 percentage points. Its invalid-call rate is 0.21%, versus 1.10% for ReMe and 0.72% for ExpeL; mean input tokens are 940.0k, 976.1k, and 996.8k per task, respectively. The matched static-plus-dynamic control without final readout solves 68/168.

Table 2: AppWorld `test_normal` (168 tasks): original arms and recent external baselines.

| Method | Success@1 (count) | Success@1 rate (%) | Invalid calls | Invalid rate (%) | Input tokens/task |
|---|---|---|---|---|---|
| Base agent | 48/168 | 28.57 | 19/3467 | 0.55 | 715.9k |
| AWM (Wang et al., 2024) | 47/168 | 27.98 | 18/3401 | 0.53 | 708.0k |
| ExpeL (Zhao et al., 2024) | 70/168 | 41.67 | 26/3598 | 0.72 | 996.8k |
| ACE (offline) (Zhang et al., 2025) | 71/168 | 42.26 | 34/3782 | 0.90 | 893.4k |
| ReMe (fixed) (Cao et al., 2025) | 73/168 | 43.45 | 42/3833 | 1.10 | 976.1k |
| ExpGraph (Feng et al., 2026) | 67/168 | 39.88 | 56/3629 | 1.54 | 1,024.7k |
| ExpeL + same-scale TGN | 58/168 | 34.52 | 12/3665 | 0.33 | 1,016.0k |
| Dynamic memory only | 52/168 | 30.95 | 14/3504 | 0.40 | 730.0k |
| Static experience only | 63/168 | 37.50 | 12/3716 | 0.32 | 850.0k |
| Static + dynamic, no final readout | 68/168 | 40.48 | 11/3800 | 0.29 | 925.0k |
| **Full method** | **82/168** | **48.81** | **8/3890** | **0.21** | **940.0k** |

Table 3 summarizes the cross-benchmark comparisons; internal ablations appear in Appendix G. On ALFWorld, the full method reaches 78/134 (58.21%) versus 70/134 (52.24%) for AdaMEM, the strongest evaluated external comparator, and 66/134 (49.25%) for ExpeL. On ScienceWorld, its mean score is 42.1 versus ExpeL's 34.7, with 25 versus 17 fully solved episodes; EMG remains unevaluated there.

Table 3: Cross-benchmark results for selected external baselines. AppWorld and ALFWorld report Success@1; ScienceWorld reports mean score on a 0–100 scale (fully solved episodes in parentheses). –: not evaluated.

| Method | AppWorld Success@1 | ALFWorld Success@1 | ScienceWorld mean (full) |
|---|---|---|---|
| Base agent | 48/168 (28.57%) | 54/134 (40.30%) | 28.4 (11/90) |
| AWM | 47/168 (27.98%) | 56/134 (41.79%) | 29.5 (12/90) |
| ExpeL | 70/168 (41.67%) | 66/134 (49.25%) | 34.7 (17/90) |
| ExpGraph (Feng et al., 2026) | 67/168 (39.88%) | 61/134 (45.52%) | – |
| AdaMEM (Zhang et al., 2026) | – | 70/134 (52.24%) | – |
| EMG (Wang et al., 2026) | – | 68/134 (50.75%) | – |
| **Full method** | **82/168 (48.81%)** | **78/134 (58.21%)** | **42.1 (25/90)** |

### 5.3 RQ2: Copycat Suppression and Evidence Use

Table 4 separates invalid calls from targeted failure modes. ExpeL makes 14 confirmed copies of historical values and 18 state-changing calls before supporting evidence is observed; the matched internal control makes 6 and 10, and the full method 2 and 4. A recovered task is a successful single-attempt rollout after an earlier tool-use error, not a second attempt or a count of repaired calls. Error categories can overlap for one action and are not components that sum to the invalid-call total. These are event counts; only invalid calls use the decision denominator. The full method makes 8 invalid calls and recovers on five tasks, consistent with fewer stale or unsupported actions alongside higher completion.

Table 4: Mechanism-level action counts on AppWorld. The matched internal control is static plus dynamic memory without the final readout.

| Method | Decisions | Invalid calls | Old-value copies | State changes without evidence | Recovered tasks |
|---|---|---|---|---|---|
| ExpeL | 3598 | 26 | 14 | 18 | 4 |
| Matched internal control | 3800 | 11 | 6 | 10 | 4 |
| **Full method** | **3890** | **8** | **2** | **4** | **5** |

### 5.4 RQ3: Component Attribution

Table 5 reports the controlled AppWorld ablations: 48/168 successes without memory, 63/168 with static experience alone, 52/168 with dynamic memory alone, 68/168 with both paths but no final readout, and 82/168 for the full method. Removing Presence or Source Ranking lowers success to 74/168 or 76/168 and raises invalid-call rate from 0.21% to 0.35% or 0.30%, respectively. Both memory paths contribute, and both readout heads help in the complete system.

Table 5: AppWorld ablations for memory composition and final readout.

| Configuration | Success@1 | Full − configuration (pp) | Invalid rate (%) |
|---|---|---|---|
| Static and dynamic off | 48/168 (28.57%) | +20.24 | 0.55 |
| Static experience only | 63/168 (37.50%) | +11.31 | 0.32 |
| Dynamic memory only | 52/168 (30.95%) | +17.86 | 0.40 |
| Static + dynamic, no final readout | 68/168 (40.48%) | +8.33 | 0.29 |
| Full w/o Presence | 74/168 (44.05%) | +4.76 | 0.35 |
| Full w/o Source Ranking | 76/168 (45.24%) | +3.57 | 0.30 |
| **Full method** | **82/168 (48.81%)** | – | **0.21** |

### 5.5 RQ4: Cross-Model and Cross-Benchmark Results

Table 6 evaluates five AppWorld backbones. Relative to ExpeL, the full method gains 7.14 points on gpt-5.6-luna and DeepSeek-V4-Pro-0813, 4.76 on gpt-5.6-sol, and 5.36 on DeepSeek-V4-flash and gemini-3.7-flash. The direction persists, but the magnitude varies.

Table 6: Cross-model AppWorld results (168 tasks).

| Backbone | ExpeL | Full | Full − ExpeL |
|---|---|---|---|
| gpt-5.6-luna | 70/168 (41.67%) | **82/168 (48.81%)** | +7.14 pp |
| gpt-5.6-sol | 78/168 (46.43%) | **86/168 (51.19%)** | +4.76 pp |
| DeepSeek-V4-Pro-0813 | 64/168 (38.10%) | **76/168 (45.24%)** | +7.14 pp |
| DeepSeek-V4-flash | 60/168 (35.71%) | **69/168 (41.07%)** | +5.36 pp |
| gemini-3.7-flash | 62/168 (36.90%) | **71/168 (42.26%)** | +5.36 pp |

For the Luna backbone, the gain over ExpeL is +7.14 points on AppWorld, +8.96 on ALFWorld, and +7.4 mean-score points on ScienceWorld. Table 4 also records fewer invalid calls, stale-value copies, and unsupported changes.

# 6 Discussion, Limitations, and Conclusion

## 6.1 Epistemic Actions and Active Exploration

Retrieving a useful procedure does not establish that its arguments are observable in the current task. Agents can gather information before acting under partial observability (Kaelbling et al., 1998; Ren et al., 2023); our readout separates seeking evidence from ranking observed sources. On AppWorld, confirmed old-value copies fall from 14 with ExpeL to 2 and unsupported state changes from 18 to 4. Removing Presence or Source Ranking lowers completion from 82/168 to 74/168 or 76/168. These results support explicit evidence checks, but the remaining errors rule out a guarantee of correct grounding.

## 6.2 Error Analysis and Limitations

The gains have three boundaries:

1. **Residual Grounding Errors**: The full method still makes 8 invalid calls across 3,890 decisions, including 2 stale-value copies and 4 unsupported changes. Ambiguous observations, misleading procedures, and unproductive `UNRESOLVED` searches remain.
2. **Compute and Latency**: On AppWorld, mean runtime rises from 125 to 140 seconds per task relative to ExpeL, although input tokens fall from 996.8k to 940.0k and mean runtime per solved task from 300 to 287 seconds. The graph encoder and readout require benchmark-specific training.
3. **Scope of Generalization**: ScienceWorld uses a stratified 90-episode panel, and the graph encoder is retrained per environment. The point estimates improve across the evaluated environments and language models; they do not establish zero-shot transfer of one trained graph or universal robustness.

## 6.3 Conclusion

We introduced **Causal Experience Instantiation** to combine abstract procedural memory with online evidence about the current task, using a dual-head readout that separates missing evidence from selection among observed sources. On AppWorld, the full system solves 82/168 tasks versus ReMe's 73/168 and ExpeL's 70/168, with an invalid-call rate of $0.21\%$ versus $1.10\%$ and $0.72\%$, respectively. Results on ALFWorld, a stratified ScienceWorld panel, and five language models indicate that this static–dynamic combination is useful across the evaluated settings, subject to the training and evaluation boundaries above.

## AI use statement

Generative AI tools assisted with the implementation of experimental code, discussion of experimental design and preliminary results, and language editing of this manuscript. The authors manually reviewed the experimental procedures and results and take responsibility for the final code, analyses, and claims.

## REFERENCES


Martín Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

Zouying Cao, Jiaji Deng, Li Yu, Weikang Zhou, Zhaoyang Liu, Bolin Ding, and Hai Zhao. Remember me, refine me: A dynamic procedural memory framework for experience-driven agent evolution. *arXiv preprint arXiv:2512.10696*, 2025.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*, 2023.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

Amir Feder, Katherine A. Keith, Emaad Manzoor, Reid Pryzant, Dhanya Sridhar, Zach Wood-Doughty, Jacob Eisenstein, Justin Grimmer, Roi Reichart, Margaret E. Roberts, Brandon M. Stewart, Victor Veitch, and Diyi Yang. Causal inference in natural language processing: Estimation, prediction, interpretation and beyond. *Transactions of the Association for Computational Linguistics (TACL)*, 10:1138–1158, 2022.

Tao Feng, Chongrui Ye, Tianyang Luo, Jingjun Xu, Xueqiang Xu, Haozhen Zhang, Zhigang Hua, Yan Xie, Shuang Yang, Ge Liu, and Jiaxuan You. ExpGraph: Model-agnostic experience learning with graph-structured memory for LLM agents. *arXiv preprint arXiv:2605.30712*, 2026.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, 2023.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning (CoRL)*, 2022.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning (CoRL)*, 2022.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024.

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023a.

Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang. Think-in-memory: Recalling and post-thinking enable LLMs with long-term memory. *arXiv preprint arXiv:2311.08719*, 2023b.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Bodhisattwa Prasad Majumder, Bhavana Dalvi Mishra, Peter A. Jansen, Oyvind Tafjord, Niket Tandon, Li Zhang, Chris Callison-Burch, and Peter Clark. CLIN: A continually learning language agent for rapid task adaptation and generalization. *arXiv preprint arXiv:2310.10134*, 2023.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Charles Packer, Vivian Fang, Shishir G Patil, Kevin Lin, Sarah Wooders, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560v1*, 2023.

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

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Judea Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2 edition, 2009.

Jonas Peters, Peter Bühlmann, and Nicolai Meinshausen. Causal inference by using invariant prediction: identification and confidence intervals. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(5):947–1012, 2016.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Allen Z. Ren, Anushri Dixit, Alexandra Bodrova, Sumeet Singh, Stephen Tu, Noah Brown, Peng Xu, Leila Takayama, Fei Xia, Jake Varley, Zhenjia Xu, Dorsa Sadigh, Andy Zeng, and Anirudha Majumdar. Robots that ask for help: Uncertainty alignment for large language model planners. In *Conference on Robot Learning (CoRL)*, 2023.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael M. Bronstein. Temporal graph networks for deep learning on dynamic graphs. *CoRR, abs/2006.10637*, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021.

Sentence-Transformers. all-MiniLM-L6-v2 model card. Hugging Face model card, n.d. URL https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

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

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. RestGPT: Connecting large language models with real-world RESTful APIs. *arXiv preprint arXiv:2306.06624*, 2023.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

Rakshit S. Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. DyRep: Learning representations over dynamic graphs. In *International Conference on Learning Representations (ICLR)*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

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.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Wenjun Wang, Yuchen Fang, Fengrui Liu, Zibo Liang, and Kai Zheng. Experience memory graph: One-shot error correction for agents. *arXiv preprint arXiv:2607.13884*, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

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

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Da Xu, Chuanwei Ruan, Evren Körpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. In *International Conference on Learning Representations (ICLR)*, 2020.

Qiantong Xu, Fenglu Hong, Bo Li, Changran Hu, Zhengyu Chen, and Jian Zhang. On the tool manipulation capability of open-source large language models. *CoRR, abs/2305.16504*, 2023.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024a.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh R. N., Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. In *International Conference on Learning Representations (ICLR)*, 2024b.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. *arXiv preprint arXiv:2510.04618*, 2025.

Yunxiang Zhang, Yiheng Li, Ali Payani, and Lu Wang. AdaMEM: Test-time adaptive memory for language agents. *arXiv preprint arXiv:2606.05684*, 2026.

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 (AAAI)*, 2024.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2024.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024.

Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the Minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. *CoRR, abs/2305.17144*, 2023.

# A Algorithmic Specification of Causal Experience Instantiation

Algorithm 1 summarizes online execution: retrieving abstract procedures, updating task-local evidence, checking the availability of slot evidence, and ranking current candidates. The action representation and observation parser are instantiated separately for each benchmark.

**Algorithm 1** Causal Experience Instantiation Online Execution Loop

**Require:** Task instruction $g$; Static Procedural Bank $\mathcal{B}$; Interactive environment $\mathcal{E}$; LLM executor $\pi_\theta$; TGN model $\mathcal{M}_{\text{TGN}}$; Readout heads $\Phi_{\text{pres}}, \Phi_{\text{rank}}$; Presence threshold $\tau_{\text{pres}}$; Maximum episode horizon $T$.

**Ensure:** Final execution status (Success / Failure) and trajectory $\mathcal{T}$.

Initialize episodic Dynamic Fact Ledger $\mathcal{L}_0 \leftarrow \emptyset$;
Initialize temporal event graph $\mathcal{G}_0 \leftarrow (\mathcal{V}_0, \mathcal{E}_0)$ with zero prior-event memory;
Retrieve the Top-3 abstract procedures $\mathcal{S} \leftarrow \text{RetrieveTopK}(\mathcal{B}, g, 3)$;
Let $\mathcal{T} \leftarrow \langle\rangle$, step counter $t \leftarrow 0$;
Receive initial environment state observation $o_0 \leftarrow \mathcal{E}.\text{reset}()$;
$\mathcal{L}_0 \leftarrow \text{ExtractFacts}(g, o_0, \tau = 0)$, retaining goal/observation provenance;
Add these initial facts to $\mathcal{G}_0$ and form $\mathbf{h}_0$ from $g$, $\mathcal{L}_0$, and zero prior-event memory;
**while** $t < T$ **and not** $\mathcal{E}$.is_terminated() **do**
  Select the next subgoal $\hat{v}_t$ from the retrieved procedures and its slots $S_t = \{s_1, \ldots, s_m\}$;
  $\mathcal{B}_{\text{bound}} \leftarrow \emptyset$; {Dictionary of grounded slot bindings}
  need_sensing $\leftarrow$ **false**;
  **for all** parameter slots $s_i \in S_t$ **do**
    Compute epistemic presence probability:
      $p_{\text{pres}}(s_i) \leftarrow \Phi_{\text{pres}}(s_i, \mathbf{h}_t, \mathcal{L}_t)$;
    Filter current-task candidates $\mathcal{C}_i \leftarrow \{\mathbf{f} \in \mathcal{L}_t \mid \text{TypeMatch}(\text{type}(\mathbf{f}), \text{type}(s_i))\}$;
    **if** $p_{\text{pres}}(s_i) < \tau_{\text{pres}}$ **or** $\mathcal{C}_i = \emptyset$ **then**
      Mark slot state as `UNRESOLVED`;
      need_sensing $\leftarrow$ **true**;
      Set the prompt directive $d_t \leftarrow \texttt{UNRESOLVED}(s_i, \hat{v}_t)$;
      **break**;
    **else**
      Score candidates using Source Ranking Head:
        $\mathbf{f}^* \leftarrow \arg\max_{\mathbf{f} \in \mathcal{C}_i} \Phi_{\text{rank}}(s_i, \mathbf{f}, \mathbf{h}_t)$;
      $\mathcal{B}_{\text{bound}}[s_i] \leftarrow \text{value}(\mathbf{f}^*)$;
    **end if**
  **end for**
  **if** need_sensing **then**
    Assemble prompt $\mathcal{P}_t \leftarrow \text{BuildPrompt}(g, \mathcal{T}, \mathcal{S}, \text{directive} = d_t)$;
  **else**
    Assemble prompt $\mathcal{P}_t \leftarrow \text{BuildPrompt}(g, \mathcal{T}, \mathcal{S}, \text{bindings} = \mathcal{B}_{\text{bound}})$;
  **end if**
  Generate benchmark-specific action $a_t \sim \pi_\theta(\cdot \mid \mathcal{P}_t)$;
  Execute action against live environment: $o_{t+1} \leftarrow \mathcal{E}.\text{step}(a_t)$;
  Append step to trajectory: $\mathcal{T} \leftarrow \mathcal{T} \circ \langle(a_t, o_{t+1})\rangle$;
  Extract newly revealed entity facts: $\Delta\mathcal{L}_{t+1} \leftarrow \text{ExtractFacts}(o_{t+1}, \tau = t+1, \text{source} = a_t)$;
  Update ledger: $\mathcal{L}_{t+1} \leftarrow \mathcal{L}_t \cup \Delta\mathcal{L}_{t+1}$;
  Update typed action, entity, provenance, and temporal edges in $\mathcal{G}_{t+1}$ from $(a_t, o_{t+1})$;
  Update TGN node and episodic memory representations: $\mathbf{h}_{t+1} \leftarrow \mathcal{M}_{\text{TGN}}(\mathcal{G}_{t+1}, \mathcal{L}_{t+1})$;
  $t \leftarrow t + 1$;
**end while**
**return** $\mathcal{E}$.get_status(), $\mathcal{T}$

**Operational scope.** The algorithm constrains its *suggested bindings*: candidates come from the current-task ledger, and an empty or low-confidence candidate set yields an `UNRESOLVED` directive. It does not guarantee that the LLM obeys the directive or that every observed value is the correct one. The measured residual invalid calls in Section 5 reflect these limits. Two design properties matter:

1. **Evidence availability**: Missing evidence is represented explicitly instead of filled with a value from an old demonstration. The directive requests further observation, which the agent may or may not perform successfully.
2. **Candidate provenance**: Each ranked candidate records when and where it was observed in the current task. This reduces direct cross-episode literal copying without implying that every candidate is semantically correct.

# B External Baseline Adaptation and Reproducibility Specifications

This section describes the execution settings and interface mappings used to run experiential-learning baselines in AppWorld. We retain their retrieval or workflow-guidance mechanisms while formatting actions for the interactive Python interface. Table 7 lists the evaluation panels and benchmark-level training sources. Retrieval counts depend on the method: our static bank and ExpeL retrieve three items, and the AWM configuration selects up to three workflows.

Table 7: A1: Evaluation panels and benchmark training sources for the main single-attempt comparison with gpt-5.6-luna (medium-fast inference). Retrieval counts are method-specific.

| Benchmark | Panel | Episodes | Horizon | Benchmark training source |
|---|---|---|---|---|
| AppWorld | `test_normal` | 168 | 40 | 90 train trajectories |
| ALFWorld | `valid_unseen` | 134 | 50 | Train expert-plan replay |
| ScienceWorld | 30 classes $\times$ 3 variants | 90 | 100 | Train-variation gold-path replay |

## B.1 AppWorld Interactive Execution Model

AppWorld (Trivedi et al., 2024) exposes an interactive Python interface. The agent writes code blocks using the `apis` client, executes them against the live state, and inspects returned values and console output. We render each method's retrieved knowledge for that coding interface.

## B.2 ExpeL Interface Mapping and Execution Details

ExpeL (Zhao et al., 2024) formalizes experiential learning through two core components: cross-task insight extraction and in-context trajectory demonstration retrieval.

- **Insight Extraction**: As in Zhao et al. (2024), we compare successful and failed training trajectories for the same task where both are available; we also extract general rules from batches of successful training trajectories. The resulting natural-language insights form a shared rule list, not an embedding-indexed set.
- **Trajectory Demonstration Retrieval**: Given a test task goal, we compute dense semantic embeddings of the task instruction using `all-MiniLM-L6-v2` (Sentence-Transformers, n.d.) and retrieve the Top-3 most similar successful trajectories from the training bank.
- **Prompt Formatting**: The retrieved trajectories are rendered as multi-turn interactive Python sessions. Each historical step displays the input prompt, generated Python code, and corresponding returned output. These demonstrations are injected into the agent's system prompt prior to task execution.
- **Interface Mapping**: We retain ExpeL's cross-task insight extraction and dense trajectory retrieval, and render its demonstrations as executable Python sessions for AppWorld.

## B.3 AWM (Agent Workflow Memory) Interface Mapping

AWM (Wang et al., 2024) uses an LLM to induce reusable workflow schemas from offline execution examples.

- **Workflow Schema Induction**: The LLM abstracts common multi-step routines from batches of training examples. For the AppWorld interface, each routine is represented with canonical API-route signatures rather than task-specific argument values.
- **Guidance Retrieval and Prompt Injection**: Our interface selects up to three workflows by overlap with the current goal terms and, after actions have been observed, their API routes. Their guidance is rendered in the agent prompt. The original offline AWM presents its induced workflows without this AppWorld-specific selection step.

### B.4 Recent External Baselines and Comparison Scope

Tables 2 and 3 report the completed same-panel comparisons with offline ACE, fixed-memory ReMe, ExpGraph, AdaMEM, and EMG where applicable. Their original mechanisms are summarized below; the table entries are results on our evaluation panels, not numbers transferred from the original papers.

For ACE, ReMe, ExpGraph, AdaMEM, and EMG, we follow the configurations recommended in the corresponding papers for the evaluated settings. Benchmark-specific adaptation is limited to mapping their actions and observations to the target interface; methods compared within a panel use the same tasks, language-model backbone, and task horizon in Table 7.

**ACE (offline).** Agentic Context Engineering (ACE) (Zhang et al., 2025) maintains an evolving playbook of reusable strategies. Generation proposes new entries, reflection examines execution feedback, and curation incrementally revises the playbook without rewriting away earlier details. We compare its offline setting on AppWorld, with playbook construction preceding evaluation.

**ReMe (fixed memory).** ReMe (Cao et al., 2025) distills procedural memories from successful and failed executions and their contrasts, retrieves them through scenario-aware indexing, and refines the memory pool according to observed utility. Our AppWorld table reports the fixed-memory setting, in which test-task execution does not modify the memory pool.

**ExpGraph.** ExpGraph (Feng et al., 2026) turns previous trajectories into reusable skills and failure lessons stored in an experience graph. It selects guidance using graph diffusion and utility-aware ranking, with a retrieval copilot trained from the effect of retrieved experience on subsequent performance. We evaluate it on AppWorld and ALFWorld.

**AdaMEM.** AdaMEM (Zhang et al., 2026) combines a long-term store of offline trajectories with short-term strategy memory synthesized as a test task unfolds. This adapts the guidance during execution without online updates to the language model's parameters. We evaluate it on ALFWorld.

**EMG.** Experience Memory Graph (EMG) (Wang et al., 2026) matches action-decision graphs from failed attempts and successful expert trajectories to extract reusable workflows and explicit correction paths. At evaluation time it retrieves these patterns to guide a single execution rather than a trial-and-error retry loop. Our ALFWorld result is reported in Table 3; ScienceWorld was not evaluated for EMG, so we do not claim superiority over EMG on that panel.

## C Prompt Templates and Agent Directive Examples

The following listings illustrate the prompt structure and AppWorld Python REPL interface. Literal task details and identifiers in these listings are schematic; they are not verbatim records of a particular evaluation rollout.

Listing 1: Common Base System Prompt for Interactive Tool-Use in AppWorld.

```
You are an expert autonomous assistant operating within the AppWorld execution
    environment.
You interact with simulated smartphone applications by writing executable Python
    code blocks inside ‘‘‘python ... ‘‘‘ fences.
Environment rules:
```

```
1. All API modules are accessed via the `apis` object (e.g., `apis.phone`, `apis.
    venmo`, `apis.calendar`).
2. Every code block is executed against the persistent environment state; variables
    defined in previous turns persist across turns.
3. Inspect observation outputs before making state-changing calls.
4. CRITICAL: Never hallucinate or copy arbitrary account IDs, transaction UUIDs, or
    contact handles. If an entity is not yet revealed in your observations, you
    must query the corresponding search or listing API first.
```

Listing 2 displays how Causal Experience Instantiation injects abstract procedural schemas alongside grounded variable bindings. The abstract schema omits past task-specific literals. Resolved slots show current-task observations; unresolved slots trigger an active-sensing directive (Listing 3).

Listing 2: Injected Context Format for Causal Experience Instantiation.

```
### Procedural Experience Guidance
Abstract Workflow Schema:
  1. apis.calendar.search_events(query=$event_title, time_range=$window)
  2. Extract $event_id and attendee emails from search results.
  3. apis.venmo.search_transactions(query=$attendee_name)
  4. apis.venmo.refund(tx_id=$tx_id, amount=$refund_amount)

Grounded Entity Ledger Bindings:
  - $event_title: "Dinner with Mark" [Observed at step 0 from task goal]
  - $attendee_name: "Mark" [Observed at step 0 from task goal]
  - $tx_id: UNRESOLVED [Epistemic Presence probability p=0.04 < tau_pres]

### Execution Directive: Active Sensing Required
The required parameter `$tx_id` has not yet been discovered in the local environment
    .
Do NOT attempt to execute `apis.venmo.refund()` with an imagined transaction ID.
Execute an exploratory search call first: `apis.venmo.search_transactions(query="
    Mark")`.
```

Listing 3: Epistemic Active Sensing Directive Prompt Template.

```
[EPISTEMIC REFUSAL & SENSING DIRECTIVE]
The readout system detected that parameter slot `{slot_name}` of type `{slot_type}`
    required by upcoming step `{subgoal_name}` is currently UNRESOLVED.
Reason: The current Dynamic Fact Ledger does not provide a supported candidate for
    this prerequisite.
Required Action:
1. Pause execution of the terminal action `{subgoal_name}`.
2. Emit an exploratory observation query using `{sensing_api}` with search key `{
    search_key}`.
3. If the environment returns a matching entity, a later readout can consider it for
     `{slot_name}`.
```

For comparison, Listing 4 and Listing 5 illustrate the corresponding trajectory and workflow-guidance interfaces. A historical ID in a retrieved demonstration is a potential copy source; whether an agent copies it is an empirical question measured by the old-value-copy counts.

Listing 4: ExpeL Demonstration Prompt Injection Format.

```
### Relevant Past Experience Demonstration
The following illustrates a successful trajectory from a similar previous task:
Task: "Send a refund of $45 for the dinner reservation."
Execution Trace:
```python
# Step 1: Look up calendar event
events = apis.calendar.search_events(query="Dinner")
print(events)
# Observation: [{'event_id': 'evt_9912', 'title': 'Dinner', 'attendee': 'Mark'}]
```
```

````
```python
# Step 2: Refund transaction directly
res = apis.venmo.refund(tx_id="tx_84021", amount=45.0, user="Mark")
print(res)
# Observation: {'status': 'success', 'refund_id': 'ref_3310'}
```
Please solve the current task following the structure of the demonstration above.
````

Listing 5: Schematic AWM workflow-memory card for the AppWorld interface.

```
<AWM_WORKFLOW_MEMORY>
Source: offline train-only workflow induction; use as advisory guidance.
WORKFLOW 01 Refund routine support=4 goal_cues=refund,order
Steps: <canonical route A> -> <canonical route B>
</AWM_WORKFLOW_MEMORY>
```

## D Dynamic Fact Ledger Schema and Provenance Tracing

The Dynamic Fact Ledger records typed values and their provenance from the current task. Table 8 gives illustrative AppWorld field categories used to explain this representation; it is not an exhaustive parser schema.

Table 8: Illustrative AppWorld entity and field categories for the Dynamic Fact Ledger.

| Application Domain | Primary Key Type | Tracked Entity Attributes and Value Types |
|---|---|---|
| **Phone & Contacts** | `contact_id` (string) | Display name, primary phone number, email address, relationship tag, home address. |
| **Messenger** | `message_id` (string) | Thread ID, sender handle, recipient list, timestamp, text content, media attachment URI. |
| **Banking / Venmo** | `transaction_id` (string) | Account number, routing number, transaction amount (float), currency, recipient ID, status string. |
| **Shopping / Amazon** | `order_id` / `asin` (string) | Item title, price scalar, inventory quantity, shipping address ID, tracking code, delivery status. |
| **Spotify** | `track_uri` (string) | Song title, artist ID, album name, playlist ID, duration (seconds), playback state. |
| **Calendar** | `event_id` (string) | Event title, start timestamp, end timestamp, recurrence rule, location string, attendee list. |
| **Email / Gmail** | `email_id` (string) | Thread ID, sender address, recipient list, CC list, subject string, body content, attachment list. |
| **Notes & Reminders** | `note_id` (string) | Note title, body text, modification timestamp, reminder alarm datetime, completion status. |
| **System / Settings** | `setting_key` (string) | Device GPS coordinates, battery level, current simulated timestamp, Wi-Fi SSID, volume level. |

### D.1 Raw Observation to Fact Tuple Transformation

To illustrate how a response becomes ledger entries, consider a schematic transaction lookup:

Listing 6: Schematic transaction lookup response; values are illustrative, not a recorded rollout.

```
# Return value from: apis.venmo.search_transactions(query="dinner")
[
  {
    "transaction_id": "tx_99104",
    "amount": 45.00,
    "sender": "usr_mark_02",
    "note": "Dinner contribution",
    "timestamp": "2026-08-03T19:30:00Z",
    "status": "completed"
  }
]
```

Upon receiving this observation at step $\tau = 2$, the Fact Ledger parser processes the raw dictionary into typed fact tuples:

$$\mathbf{f}_1 = \langle \text{val} = \texttt{"tx_99104"}, \text{type} = \texttt{transaction_id}, \tau = 2, \text{src} = \texttt{apis.venmo} \rangle,$$
$$\mathbf{f}_2 = \langle \text{val} = 45.00, \text{type} = \texttt{monetary_amount}, \tau = 2, \text{src} = \texttt{apis.venmo} \rangle,$$
$$\mathbf{f}_3 = \langle \text{val} = \texttt{"usr_mark_02"}, \text{type} = \texttt{user_id}, \tau = 2, \text{src} = \texttt{apis.venmo} \rangle.$$

The entries retain a source and observation step, allowing the readout to distinguish them from literals that occur only in retrieved demonstrations. This provenance does not by itself validate the semantic correctness of a candidate.

# E Hyperparameters, Compute Infrastructure, and Runtime Overhead

## E.1 System Hyperparameters and Training Setup

Table 9 summarizes the architectural dimensions, optimization settings, and inference hyperparameters of Causal Experience Instantiation. The temporal encoder and two readout heads are trained on benchmark-specific training trajectories with the Presence and conditional Source Ranking losses in Section 4. The semantic text encoder remains frozen; at test time, only task-local graph and ledger state changes.

Table 9: Reported system hyperparameters and training configuration.

| **Module / Hyperparameter** | **Specification / Value** |
|---|---|
| Semantic encoder | 384-dimensional frozen text representation; 512-token windows with 64-token overlap |
| Temporal graph | Action-type, object-slot, and evidence nodes; transition, ownership, and observation edges |
| Temporal graph dimensions | Hidden state 128; time representation 64 |
| Static bank | 20 induced rules; Top-3 retrieval; benchmark-specific training bank |
| Epistemic Presence Head | Two-layer MLP with sigmoid output; threshold $\tau_{\text{pres}} = 0.75$ |
| Source Ranking Head | Two-layer MLP over prefix, slot, and current candidate features; raw-logit argmax |
| Readout objective | Paired logistic presence loss plus conditional multi-positive set-ranking loss; equal weighting |
| Optimizer | AdamW; learning rate $3 \times 10^{-4}$; batch size 32 |
| Training and selection | At most 15 epochs; checkpoint selected on development data |
| Inference | Model weights and static bank frozen; task-local ledger and graph state updated online |

## E.2 Runtime Computational Complexity and Latency Profiling

Dynamic graph updates add inference cost during interactive execution. Table 10 reports the aggregate additional graph/readout cost per decision, while Table 15 reports end-to-end runtime and training cost.

Table 10: Wall-clock inference overhead per decision step.

| **Pipeline Component** | **Mean Latency** |
|---|---|
| Dynamic Fact Ledger, TGN update, and dual-head readout (Full) | **0.18 s / decision** |

Benchmark-level runtime and training GPU-hours are reported in Table 15.

# F Sensitivity Analysis and Failure Mode Diagnostics

## F.1 Sensitivity to Epistemic Presence Threshold $\tau_{\text{PRES}}$

The decision threshold $\tau_{\text{pres}}$ governs the agent's willingness to proceed with action instantiation versus triggering active sensing. To examine the robustness of the system to this parameter, we vary $\tau_{\text{pres}} \in [0.60, 0.90]$ while keeping all other parameters constant. Table 11 reports the benchmark success rate, invalid call rate, and active-sensing decision rate across the 168 evaluation tasks. The

Table 11: Sensitivity of task completion and execution fidelity to the presence threshold $\tau_{\text{pres}}$.

| Threshold $\tau_{\text{pres}}$ | Success@1 | Invalid Calls / Decisions | Sensing Decisions (%) |
|---|---|---|---|
| 0.60 | 78/168 | 10/3950 (0.25%) | 8.5% |
| 0.70 | 81/168 | 9/3970 (0.23%) | 13.2% |
| **0.75** | **82/168** | **8/3890 (0.21%)** | **17.6%** |
| 0.80 | 81/168 | 7/3800 (0.18%) | 23.9% |
| 0.90 | 77/168 | 6/3650 (0.16%) | 34.8% |

latter is the fraction of agent decision steps at which the readout issues an `UNRESOLVED` information-gathering directive, rather than the fraction of subsequent actions that happen to be lookup calls.

In this sweep, lower thresholds produce fewer `UNRESOLVED` directives and more invalid calls; higher thresholds show the opposite pattern. The bold row marks the reported default ($\tau_{\text{pres}} = 0.75$); the sweep does not establish that it is optimal beyond this panel.

### F.2 Four-Group Error Taxonomy

Table 12 reports error counts by application group, contrasting ExpeL, the matched internal control, and the full method. These are raw counts; without group-level exposure denominators, they do not compare error risk across groups.

Table 12: Aggregate AppWorld error counts by application group, not individual task traces. Categories may overlap for a single action.

| Application Group | Method | Invalid Calls | Confirmed Old-Value Copies | Missing-Evidence Changes |
|---|---|---|---|---|
| Shopping / Orders | ExpeL | 10 | 6 | 7 |
| Shopping / Orders | Matched internal control | 4 | 2 | 4 |
| Shopping / Orders | Full | 2 | 1 | 2 |
| Calendar / Productivity | ExpeL | 8 | 4 | 5 |
| Calendar / Productivity | Matched internal control | 3 | 2 | 3 |
| Calendar / Productivity | Full | 3 | 0 | 1 |
| Finance | ExpeL | 5 | 3 | 4 |
| Finance | Matched internal control | 3 | 2 | 2 |
| Finance | Full | 2 | 1 | 1 |
| Social / Contacts | ExpeL | 3 | 1 | 2 |
| Social / Contacts | Matched internal control | 1 | 0 | 1 |
| Social / Contacts | Full | 1 | 0 | 0 |
| **Total** | **ExpeL** | **26** | **14** | **18** |
| **Total** | **Matched internal control** | **11** | **6** | **10** |
| **Total** | **Full** | **8** | **2** | **4** |

The breakdown separates shopping/orders, calendar/productivity, finance, and social/contacts. The schematic examples in Appendix C are not recorded task traces.

## G Summary Experiment Tables

Table 13 extends the task-level ablation across three benchmarks. Table 14 reports the offline readout diagnostics separately: ranking or calibration quality is not itself a measure of completed tasks.

Table 13: A2: Closed-loop ablations. ScienceWorld reports mean score; AppWorld and ALFWorld report completed tasks; invalid rate is for AppWorld.

| Configuration | AppWorld | ALFWorld | ScienceWorld | Invalid rate |
|---|---|---|---|---|
| No memory | 48/168 | 54/134 | 28.4 | 0.55% |
| Static only | 63/168 | 64/134 | 33.1 | 0.32% |
| Dynamic only | 52/168 | 59/134 | 30.3 | 0.40% |
| Static + dynamic, no final readout | 68/168 | 69/134 | 36.5 | 0.29% |
| Full w/o Presence | 74/168 | 74/134 | 39.6 | 0.35% |
| Full w/o Source Ranking | 76/168 | 75/134 | 40.2 | 0.30% |
| **Full** | **82/168** | **78/134** | **42.1** | **0.21%** |

Table 14: A2 continued: Offline evidence-availability (AUROC/ECE) and source-selection (Recall@1/MRR) diagnostics on held-out prefixes. The first row is a no-dedicated-readout control, not a trained dual-head module.

| Readout | AUROC ↑ | ECE ↓ | Recall@1 ↑ | MRR ↑ | UNRESOLVED |
|---|---|---|---|---|---|
| No dedicated readout | 0.71 | 0.15 | 0.51 | 0.64 | 0% |
| Presence only | 0.84 | 0.06 | 0.52 | 0.65 | 12% |
| Source Ranking only | 0.72 | 0.14 | 0.73 | 0.82 | 0% |
| **Full dual-head readout** | **0.85** | **0.06** | **0.74** | **0.83** | **11%** |

The no-readout control has no trained heads. Its offline diagnostics use the control's evidence and candidate-ordering signals; its 0% `UNRESOLVED` rate reflects the absence of a dedicated readout directive.

Table 15: A4: Cost and latency. In/Out are mean input/output tokens in thousands per episode; Calls are mean LLM calls per episode; GPU-h is training GPU-hours for each benchmark run.

| Benchmark | Method | In k | Out k | Calls | Graph s/step | s/task | s/solved | GPU-h |
|---|---|---|---|---|---|---|---|---|
| AppWorld | ExpeL | 996.8 | 14.0 | 21.4 | 0.00 | 125 | 300 | 0.0 |
| AppWorld | Full | 940.0 | 15.0 | 23.2 | 0.18 | 140 | 287 | 4.2 |
| ALFWorld | ExpeL | 292.0 | 9.0 | 32.2 | 0.00 | 80 | 162 | 0.0 |
| ALFWorld | Full | 290.0 | 10.0 | 35.1 | 0.18 | 86 | 148 | 6.5 |
| ScienceWorld | ExpeL | 365.0 | 17.0 | 63.9 | 0.00 | 238 | 1,260 | 0.0 |
| ScienceWorld | Full | 370.0 | 18.0 | 66.7 | 0.18 | 260 | 936 | 8.0 |

The A4 cost table reports mean tokens and calls per episode. The graph/readout column is the incremental per-decision cost of the full method, not its entire wall-clock latency. Seconds per solved task divides total runtime over all evaluated episodes by the number solved; ScienceWorld counts a fully solved episode only at score 100.

The two-attempt analysis below uses a separate reset-environment protocol from the single-attempt main tables. The second-attempt column counts all tasks successful on attempt 2, including those that also succeed on attempt 1; Solved@2 counts a task if either attempt succeeds. Mean attempts to first success uses an early-stop calculation on the observed pair.

Table 16: A7: Two-attempt protocol. Second-attempt success includes tasks successful on both attempts; Solved@2 is the union across attempts.

| Method | Success@1 | Success on attempt 2 | Solved@2 | Mean attempts (solved) | Total calls (two rounds) |
|---|---|---|---|---|---|
| ExpeL | 70/168 | 84/168 | 94/168 | 1.26 | 7,300 |
| **Full** | **82/168** | **99/168** | **108/168** | **1.24** | **7,800** |

*Outcome partition across the two attempts*

| Method | Both successful | First only | Second only | Both failed |
|---|---|---|---|---|
| ExpeL | 60 | 10 | 24 | 74 |
| **Full** | **73** | **9** | **26** | **60** |

# H SCHEMATIC QUALITATIVE EXAMPLES

## H.1 SCHEMATIC EXAMPLE 1: RESOLVING COPYCAT COLLISIONS IN FINANCIAL WORKFLOWS

Table 17 is a schematic illustration contrasting flat demonstration copying (EXPEL) with Causal Experience Instantiation; it is not a recorded task trace.

Table 17: Schematic side-by-side example of a financial workflow; identifiers and outcomes are illustrative, not a recorded task trace.

| **Flat Trajectory Retrieval (EXPEL)** | **Causal Experience Instantiation (Ours)** |
|---|---|
| `Step 1: Retrieve demonstration from memory.`<br>Demonstration contains:<br>`venmo.refund(tx_id="tx_84021", user="Mark")` | `Step 1: Retrieve abstract TEG schema.`<br>`calendar.find_events($s1)` $\prec$ `venmo.refund($s2, $s3)` |
| `Step 2: Generate Action (LLM).`<br>LLM copies historical transaction ID from demonstration:<br>`apis.venmo.refund(tx_id="tx_84021")`<br>**Result:** `VenmoAPIError: Transaction tx_84021 not found.` | `Step 2: Dual-Head Readout on $s2 (tx_id).`<br>$p(\text{present} \mid \$s2, \mathbf{h}_1) = 0.04 < \tau_{\text{pres}}$<br>Status: `UNRESOLVED` (Prerequisite unobserved).<br>**Action emitted:** Epistemic active sensing:<br>`apis.venmo.search_transactions( query="dinner")` |
| `Step 3: Recovery attempt.`<br>LLM is confused by 404 error; retries same transaction ID with alternative syntax; exhausts step limit. | `Step 3: Update Dynamic Fact Ledger.`<br>Observation returns:<br>`[{tx_id: "tx_99104", amount: 45.0, user: "Mark"}]`<br>Fact $\mathbf{f}_1$ added to $\mathcal{L}_2$. |
| `Final Status:` **FAILED** (Invalid API calls, zero state changes applied). | `Step 4: Dual-Head Readout on $s2.`<br>$p(\text{present} \mid \$s2, \mathbf{h}_3) = 0.98 \geq \tau_{\text{pres}}$<br>Ranking Head selects $\mathbf{f}_1$ ($p = 0.96$).<br>**Action emitted:** `apis.venmo.refund( tx_id="tx_99104")`<br>**Result:** `Success. Refund executed.`<br>**Final Status:** **SUCCESS** |

## H.2 SCHEMATIC EXAMPLE 2: CROSS-APPLICATION ENTITY DEPENDENCY

The following schematic requires coordinating across three distinct applications. Table 18 demonstrates how Causal Experience Instantiation coordinates multi-hop entity resolution without cross-domain literal confusion; it is not a recorded task trace.

Table 18: Schematic multi-hop cross-application entity binding example; identifiers and outcomes are illustrative.

| Execution Step | Agent Action and Fact Ledger Grounding State |
|---|---|
| **Step 1** (Calendar Query) | **Emitted Action:** `apis.calendar.search_events(query="design review")` **Observation:** `[{event_id: "evt_512", organizer: "sarah@company.com"}]` **Ledger Update:** Fact $\mathbf{f}_1 = \langle$"sarah@company.com", `email_address`, $\tau = 1\rangle$ added to $\mathcal{L}_1$. |
| **Step 2** (Epistemic Readout) | **Target Subgoal:** `apis.phone.send_sms($recipient_phone, $msg)` **Readout on $recipient_phone:** $p_{\text{pres}} = 0.02 < \tau_{\text{pres}} \implies$ `UNRESOLVED`. **Active Sensing Action:** `apis.phone.search_contacts(email="sarah@company.com")` |
| **Step 3** (Contacts Query) | **Observation:** `[{contact_id: "c_881", name: "Sarah Chen", phone: "+1-555-0192"}]` **Ledger Update:** Fact $\mathbf{f}_2 = \langle$"+1-555-0192", `phone_number`, $\tau = 3\rangle$ added to $\mathcal{L}_3$. |
| **Step 4** (Final Dispatch) | **Target Subgoal:** `apis.phone.send_sms($recipient_phone, $msg)` **Readout on $recipient_phone:** $p_{\text{pres}} = 0.99 \geq \tau_{\text{pres}}$, Ranking Head binds $\mathbf{f}_2$. **Emitted Action:** `apis.phone.send_sms(phone="+1-555-0192", message="Confirmed for Friday.")` **Result:** `SMS sent successfully.` **Final Status: SUCCESS.** |

### H.3 Interpreting the Examples Against Aggregate Results

The two tables above are schematic, not measured per-task trajectories. The aggregate error counts are in Table 12, and task-level ablations are in Table 5. The abstract bank excludes historical identifiers, while the ledger tracks current candidates with source and timestamp; the readout can defer a missing binding or rank type-compatible sources. This design is consistent with fewer confirmed old-value copies (14 for ExpeL, 6 for the matched control, and 2 for the full method), but these counts do not explain every invalid call or prove that a selected entity satisfies the user's goal. The full method still makes eight invalid calls and takes longer per task than ExpeL (Table 15).