MEMGYM: a Long-Horizon Memory Environment for LLM Agents

Wujiang Xu[1*] **Yu Wang**[2] **Kai Mei**[1] **Kaiqu Liang**[3] **Zhenting Wang**[1] **Mingyu Jin**[1]
Han Zhang[1] **Shi-Xiong Zhang**[2] **Wenyue Hua**[4] **Sambit Sahu**[2] **Dimitris N. Metaxas**[1]

[1]Rutgers University [2]Capital One [3]Princeton University [4]Microsoft Research

Abstract

Memory is a central capability for LLM agents operating across long-horizon tasks. Existing memory benchmarks predominantly evaluate retention of personalized information in multi-turn chat scenarios, overlooking the dynamic memory formation that occurs during extended agent execution. Consequently, the memory systems they produce transfer poorly to realistic agentic environments such as coding and web navigation. We present MemGym, a benchmark for agentic memory that unifies existing agent gyms and in-house memory-grounded pipelines behind one memory–reasoning interface. MemGym spans five evaluation tracks grouped into four agentic regimes: tool-use dialogue (τ^2-bench), multi-turn deep-research search (MEMGYM-DR), coding (SWE-Gym and MEMGYM-CODEQA), and computer use (WebArena-Infinity). MemGym reports memory-isolated scores that decouple memory performance from reasoning, retrieval, and tool-use ability, so memory strategies can be ranked without those confounders. Our synthetic pipelines for MEMGYM-CODEQA and MEMGYM-DR are length-controllable, ablation-verified at every stage, and tightly aligned with downstream scenarios. To make evaluation on coding environments academically tractable, we train MemRM, a lightweight reward model (Qwen3-1.7B fine-tuned with QLoRA) that scores compression quality as a fast scalar read in place of full Docker rollouts.

Project Page Code Dataset

1 Introduction

LLM agents operating over long horizons must continuously decide what to preserve, summarize, or evict as observations, tool outputs, and intermediate conclusions accumulate. We refer to this process as *memory formation during agent execution*, distinguishing it from the static recall tested by long-context benchmarks [4, 17]. This capability arises across diverse realistic settings: coding agents revisit earlier debugging evidence across repository-scale tasks [20, 32], retrieval agents preserve bridge facts across search turns [42, 16, 52], and dialogue agents retain user constraints and tool state through extended interactions [55, 5].

Existing memory benchmarks [29, 48, 18, 1] predominantly evaluate retention of personalized information in multi-turn chat, revealing little about memory behavior within agents that interleave perception, reasoning, and tool use. Three obstacles compound this gap. **(i) Entangled metrics:** Agent gyms that involve long-horizon execution (SWE-Gym, τ^2-bench, WebArena) report only end-task success, conflating memory failures with reasoning, retrieval, and tool-use errors. **(ii) Illusory memory pressure:** Settings that appear memory-intensive often admit strong performance without explicit memory management, as facts remain re-derivable from repositories or recoverable from

*Corresponding author: wujiang.xu@rutgers.edu

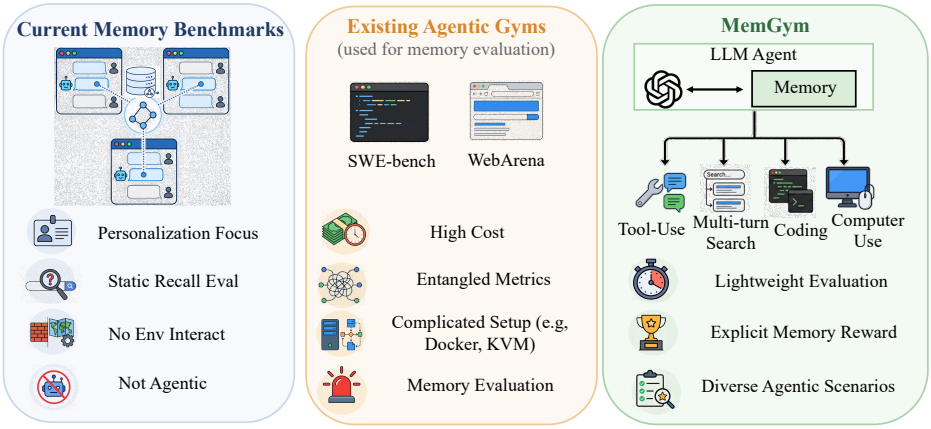


Figure 1: MEMGYM unifies five evaluation tracks across four agentic regimes (tool-use dialogue, multi-turn search, coding, computer use) behind a shared interface that separates memory from reasoning and supports memory-isolated scoring with explicit memory rewards.

pretraining. **(iii) Evaluation cost:** A single SWE-Gym rollout requires Docker infrastructure and tens of execution steps, placing systematic memory design iteration beyond most academic budgets.

We present MEMGYM, a benchmark, training-data pipeline, and lightweight evaluator that targets all three obstacles together (Figure 1). MEMGYM unifies five evaluation tracks behind a shared interface that explicitly separates a memory module from the reasoning model: three wrappers around existing benchmarks (τ^2-bench, SWE-Gym, WebArena-Infinity) plus two memory-grounded tracks we constructed in-house (MEMGYM-DR deep research, MEMGYM-CODEQA from SWE-smith). Every compression event is therefore observable, comparable, and recordable across scenarios. On top of this interface, MEMGYM reports *memory-isolated* scores that disentangle memory performance from the underlying task, so that memory strategies can be ranked without being confounded by reasoning, retrieval, or tool-use ability.

Two complementary components close the loop from measurement to training. First, controllable synthetic pipelines for MEMGYM-CODEQA and MEMGYM-DR produce instances of tunable length and use verifier ablations to test intended memory against no-memory, distractor-only, and leakage-prone conditions; they are tightly aligned with the downstream coding and search scenarios they target, rather than acting as one-shot labor exercises. Second, MEMRM, a lightweight reward model trained on collected trajectories (Qwen3-1.7B fine-tuned with QLoRA), scores compression quality as a fast scalar read in place of full Docker rollouts, which makes coding-environment evaluation academically tractable. The same paired trajectories are released as a labeled corpus for downstream training research. MEMGYM therefore makes three contributions:

- **Five tracks behind one memory interface, scored memory-isolated.** MEMGYM unifies τ^2-bench, SWE-Gym, WebArena-Infinity, and the in-house MEMGYM-DR and MEMGYM-CODEQA pipelines under a shared memory contract, and reports paired baseline-vs-memory deltas under a fixed reasoner so the score reads as a memory effect rather than a confound of reasoning, retrieval, or tool use (§3.2, §3.3).
- **Controllable, ablation-verified synthetic pipelines.** MEMGYM-CODEQA and MEMGYM-DR generate length-tunable instances at scale, and verify via per-stage ablations that the intended memory channel (not parametric leakage or distractor recall) is the one being tested (§3.4).
- **MEMRM: a scalar gate that replaces a Docker rollout.** A 1.7B-parameter QLoRA reward model trained on compression-event outcomes reaches AUROC 0.985 on the SWE-Gym IID split, swapping a per-event rollout for a sub-second classifier call and supplying graded rewards for downstream post-training (§3.3).

Together, these components turn the long-horizon evaluation loop from a one-way measurement into a closed feedback loop: the same trajectories that surface where current memory systems break also become the supervision signal for fixing them. The rest of the paper develops the framework (§3), benchmark construction (§3.4), experiments (§4), and future directions and limitations (Appendix J).

Table 1: Comparison of memory and long-horizon agent benchmarks. *Min. Cost* is the cheapest path to score one memory configuration: Low, Medium, High.

Benchmark	Agentic Scenarios	# Scn	Inter-active	Memory-Isolated	Min. Cost	Train. Data	Length
Dialogue-Centric Memory Benchmarks							
LoCoMo [29]	Long-term dialogue	1	✗	✗	Low	✗	9K
LongMemEval [48]	Long-term dialogue	1	✗	✗	Medium	✗	115K
MemoryAgentBench [18]	Multi-turn dialogue	1	✗	✗	Medium	✗	100K–300K
MemoryBench [1]	Continual dialogue	1	✗	✗	Medium	✗	30K–380K
Long-Horizon Agent Benchmarks							
SWE-Gym [32]	Repository coding	1	✓	✗	High	✓	Task-dep.
τ^2-bench [5]	Tool-agent-user dialogue	1	✓	✗	Medium	✗	Task-dep.
WebArena-Infinity [59]	Web computer use	1	✓	✗	Medium	✗	Configurable
Agent-Centric Memory Benchmarks							
AMA-Bench [57]	Agentic apps (post-hoc QA)	1	✗	✗	Medium	✗	57K
AMemGym [6]	Personalized conversation	1	✓	✗	Medium	✗	Configurable
MEMGYM (Ours)	Coding, web, tool-dialogue, deep-research search, coding QA	**5**	✓	✓	**Low**	✓	**Configurable**

2 Related Work

Agentic Memory Systems. Early memory-augmented LLM systems (MemoryBank [58], MemGPT [31], and ReadAgent [24]) added explicit memory components but evaluated on long-term dialogue or document understanding rather than memory formation during environment interaction. A-Mem [50] introduces agentic note evolution and is evaluated on LoCoMo [29], a long-term conversational-memory benchmark that mainly tests needle-in-a-haystack recall over personas and temporal event graphs rather than memory formed while debugging code, using tools, or navigating websites. LongMemEval [48], MemoryAgentBench [18], and MemoryBench [1] extend this line with scalable histories, incremental multi-turn ingestion, and continual-learning feedback, but the memory target remains a transcript or feedback stream rather than a live trajectory under tool-use pressure. Most recently, AMA-Bench [57] and AMemGym [6] move closer to our setting (the former evaluates memory over agentic trajectories via post-hoc QA, the latter provides on-policy conversation with structured latent-state evolution), but neither offers a unified memory interface with memory-isolated rewards across coding, search, tool dialogue, and web control.

Long-Horizon Agent Benchmarks. Long-horizon agent benchmarks evaluate whether agents can complete extended tasks in executable environments. SWE-bench [20] tests real GitHub issue resolution, and SWE-Gym [32] adds executable training tasks with unit tests and released trajectories; full evaluation is expensive and sparse, so end-task resolve rate alone is impractical for systematic memory iteration. τ-bench and τ^2-bench [55, 5] evaluate tool-agent-user workflows but report end-task success without isolating whether failures came from memory, policy, or tool use. WebArena [60] provides functional web environments, and WebArena-Infinity [59] scales this by automatically generating self-contained applications with verifiable tasks; OSWorld and OSGym [49, 36] extend the same line to desktop and operating-system tasks. Across these benchmarks, memory is load-bearing but not separately measured. MEMGYM wraps such environments with an explicit memory boundary, records compression events, and reports memory-isolated scores so memory systems can be compared independently of the underlying agent's reasoning, retrieval, and tool-use ability.

3 MEMGYM: A Memory-Centric Evaluation and Training Framework

3.1 Overview

MEMGYM evaluates agentic memory across five environments unified by a shared memory module that wraps the prompt sent to the policy LLM: τ^2-bench dialogue [5], SWE-Gym coding [20, 32], WebArena-Infinity computer use [60, 59], MEMGYM-DR deep research, and MEMGYM-CODEQA. The first three are wrappers around existing benchmarks; the latter two are environments for which we additionally constructed memory-grounded instances in-house, growing or extracting the facts the agent must retain rather than relying on benchmarks where memory state is incidental. All five plug

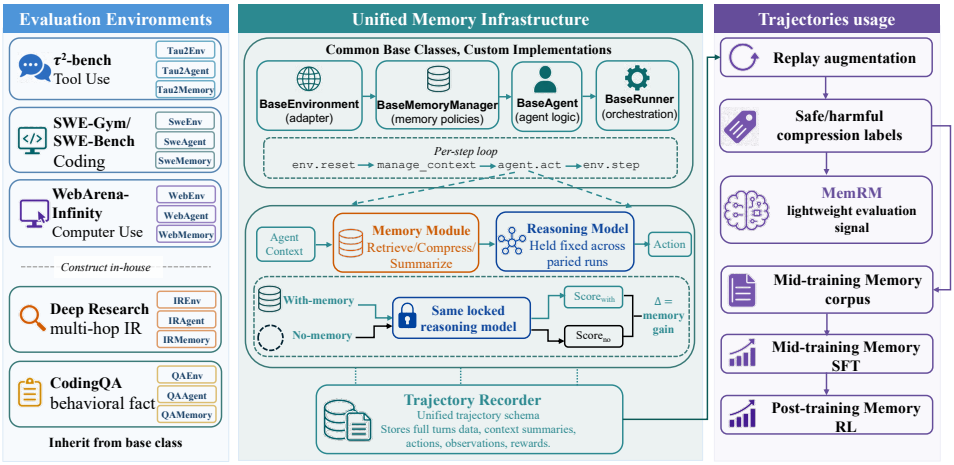


Figure 2: MEMGYM architecture. Five environments share a memory module that wraps the prompt to the policy LLM, so the same strategy runs on any environment unchanged. Trajectories feed a replay-augmentation pipeline producing SAFE/HARMFUL labels for MEMRM; MEMGYM-DR and MEMGYM-CODEQA come from in-house pipelines (Section 3.4).

into the same per-step contract described in Section 3.2; the two construction pipelines are themselves contributions and are detailed in Section 3.4. Every with-memory vs. no-memory comparison in this paper holds the reasoning model fixed across both sides of the paired run, so the score delta (the *memory gain*) isolates memory rather than confounding it with model choice. Trajectories collected through the unified wrapper feed a replay-augmentation pipeline that produces SAFE/HARMFUL compression labels, which train MEMRM, a lightweight classifier (Qwen3-1.7B fine-tuned with QLoRA) that predicts $\Pr[\text{behavior unchanged} \mid \text{compress}]$ in sub-second time, replacing per-episode Docker evaluation as the inner-loop signal for strategy iteration and serving as the reward signal for the post-training experiments in Section 4. Figure 2 shows how the pieces fit together.

3.2 Unified Memory Infrastructure

Seven memory families plus a no-memory *None* control are evaluated across 17 (track, strategy) cells over the five environments (the cell list is enumerated in Section C.2); all share one per-step contract, and the engineering depth (container backends, perturbation taxonomy, and harness patches) is documented in Appendix G. All five environments plug into a common contract (`BaseMemoryEnvironment`, `BaseAgent`, `BaseMemoryManager`, `BaseRunner`) and a single per-step cycle: `env.reset()` → `memory_manager.manage_context` → `agent.act` → `env.step`. The memory manager wraps the prompt to the policy LLM and returns a `FilteredContext` plus a per-event `condensation_event` (summary, forgotten-message indices, compression metadata); together with the per-step trajectory record (Appendix F), this reconstructs the training signal of any episode without re-running it. Per-environment wrappers capture each environment's dominant evaluation concern: τ^2-bench wraps both the agent and the user simulator with independent memory managers, enabling the three-condition ablation that isolates whose compression matters; adding a new environment or strategy is a one-file change via `register_env('name', cls)` or `register_memory_model('name', cls)`.

Memory operations expose a single `manage_context` contract, motivated by the OpenHands `CondenserPipeline` [45], and compose via a `PipelineMemory` wrapper that accumulates per-stage statistics for downstream analysis. A `repair_tool_call_pairs` primitive enforces the invariant that no compression can produce an API-invalid message sequence: if a summarizer drops the assistant turn that issued a tool call, the orphaned tool result is removed (and vice versa), since silently invalid sequences are silently rejected by Bedrock's Converse API and surface as zero-reward task failures rather than diagnosable errors. MEMGYM evaluates four memory-operation families as primary baselines (passthrough, LLM summarizing, structured per-environment summary, retrieval-style); the per-environment cross-product, one additional ported operation (observation masking), and the `set_episode_state` hook are documented in Appendix E.

Replay-and-fork harness. A multi-runtime `ContainerBackend` covers Docker, Singularity (HPC), `BubblewrapBackend` (rootless namespaced sandbox over a Docker-exported rootfs, for clusters where neither Docker nor Singularity is available), `SwerexDockerBackend` (which copies recorded tool-result messages verbatim across the fork point because Bedrock's Converse API rejects orphaned `tool_use` IDs), and a local `tempfile` backend; a `WebArenaServerPool` mirrors this on the WebArena side with port-range allocation, idle-pool reuse, and per-process log capture for parallel task execution. A *replay-and-fork* protocol re-uses recorded tool actions to reconstruct repository state at the compaction step (`compute_auto_fork_step` returns the earliest threshold-crossing step), and `ObservationReplayRunner`, which re-queries the policy LLM only on compaction-triggering steps, records a $10\times$ `policy_call_savings` that is what makes a 170-task hard-tier sweep tractable. The same recorded trajectories drive a counterfactual-replay augmentation pipeline realized as two processes from one rationale: cheap text-only replays from baseline trajectories for breadth, plus Docker-snapshotted replays from memory trajectories that yield ground-truth labels in milliseconds via `docker commit` rather than full re-execution. Together these expose the model at mid-training to what each memory operation causes *in this scene*, so it learns per-step information importance rather than memorizing one canonical trajectory. Trajectory schemas, the harness fixes, and per-source label provenance are detailed in Appendix F, Appendix G, and Appendix H; the score difference between paired baseline-vs-memory runs (always with the same reasoning model on both sides) is what we call *memory gain* throughout the paper. Memory still alters the action distribution downstream of the wrapper, so memory gain attributes effects to the memory module under a fixed reasoner; it is not a clean ablation of memory as an independent capability.

3.3 MEMRM as a Lightweight Evaluation Signal

Measured on local trajectory data under Claude Sonnet 4.5, a single passthrough-memory episode costs \$2.10 on SWE-Gym [32] (median 55 turns and 700K cumulative input tokens), \$1.50 on WebArena-Infinity [60] hard tier, \$0.40 on τ^2-bench [5], and \$0.15 on MEMGYM-DR at list pricing of \$3/MTok input plus \$15/MTok output. A 100-episode sweep across the four interactive environments therefore costs \$420 and a 5-strategy $\times$ 3-seed sweep scales to \$6,300, making per-episode rollouts infeasible as the inner-loop signal for strategy iteration, ablation sweeps, or rollout filtering during training. We want a sub-second scalar predictor: given a (context-before, compressed-context, candidate-action) triple, estimate $\Pr[\text{behavior unchanged} \mid \text{compress}]$, usable as both a lightweight evaluation gate and the downstream reward signal in Section 4. It is a reward model in the RLHF sense [30], not a latent-dynamics world model in the Ha–Schmidhuber sense [14].

We obtain (context, action, label) training triples by *replaying recorded trajectories at compaction events*, then aggregate SAFE/HARMFUL labels from three complementary sources: (i) episode-level task resolution on the parent trajectory, (ii) counterfactual replay where a perturbation of the message window causes an action divergence on re-querying the policy, and (iii) LLM-as-judge over the forgotten content. The pipeline runs two replay processes from one rationale: a cheap text-only process replays baseline trajectories to harvest counterfactual (context, action) pairs at scale, and a Docker-snapshotted process replays memory trajectories and verifies each perturbation's predicted action against a snapshot taken at the compaction event, producing ground-truth labels in milliseconds via `docker commit` rather than full re-execution. More details are in Appendix H.

Training recipe. The augmented corpus contains 18.6K (context, action, label) triples at a median context length of 22K tokens; the train/eval split is grouped by repository (not by instance) to eliminate same-repo leakage. Each row carries the full agent view through the candidate compression and a binary completion ("Y" for SAFE, "N" for HARMFUL). MEMRM is initialized from Qwen3-1.7B-Base and fine-tuned with QLoRA (NF4 4-bit, rank 16, α=32, targets $\{q, k, v, o\}$) using TRL's `SFTTrainer` with `completion_only_loss=True`, so the loss fires only on the single label token. Class imbalance is handled by class-balanced cross-entropy with weights from the sklearn *balanced* formula, capped at $w_{\max} = 3.0$:

$$\mathcal{L}_{\text{MEMRM}}(\theta) \;=\; -\,\mathbb{E}_{(c,y)\sim\mathcal{D}_{\text{aug}}}\big[\, w_y \,\log p_\theta\big(y \mid c\big)\,\big], \qquad y \in \{\text{SAFE}, \text{HARMFUL}\}, \tag{1}$$

where c is the (context-before, compressed-context, candidate-action) triple linearized into the Qwen3 chat template and w_y is the (capped) class-balanced weight. At inference time, we sweep a decision threshold $t^\star$ on held-out trajectories under the constraint $F_1^{\text{HARMFUL}} \geq 0.90$ and $\text{prec}^{\text{SAFE}} \geq 0.80$, and classify a candidate compression as SAFE iff $\Pr[\text{SAFE} \mid c] > t^\star$. MEMRM's primary use in this paper is as a lightweight evaluation gate (turning a 10-minute coding rollout into a sub-second scalar

read) and as the post-training reward signal whose results we report in Section 4. Augmented-corpus statistics, the perturbation taxonomy, the env-feedback Docker snapshot protocol, the threshold sweep, and held-out calibration are in Appendix H.

3.4 Constructed Pipelines for Memory-Grounded Evaluation

Template and sources. Two of the five evaluation environments (MEMGYM-CODEQA and MEMGYM-DR) additionally require constructing memory-grounded instances rather than wrapping an existing benchmark. Both pipelines follow the same template: ingest a source, extract a taxonomy of memory-only versus discoverable facts, inject shortcut-blocking distractors, scale to a target context length, and certify with a multi-criterion verifier. MEMGYM-CODEQA ingests SWE-smith [51] bug-and-patch instances; MEMGYM-DR ingests results from academic search backends (arXiv, Semantic Scholar [22], OpenAlex [35], and Wikipedia).

Fact extraction. What constitutes a "memory-only fact" differs by domain. MEMGYM-CODEQA runs a three-pass extraction over each SWE-smith bug (gold-patch-visible seed, patch-hidden extraction, then discoverability re-examination with the full repository) and tags each fact as `discoverable` or `memory-only`; an instance is kept only if it exposes at least two critical `memory-only` facts (the threshold below which the question reduces to "search the repo"). MEMGYM-DR instead grows an ordered bridge-fact chain through iterative search, where each fact carries a `retention_span` (the number of hops between when it is first discoverable and when it must be applied), so a 4-hop instance contains bridge facts requiring spans of 3, 2, and 1, plus a terminal fact. The retention-span structure is the technical fingerprint that distinguishes multi-hop retention from single-pass retrieval.

Distractors and length. Both pipelines inject shortcut-blocking distractors, but the shortcut taxonomy differs by domain. MEMGYM-CODEQA blocks four shortcuts (topical inference, conventions-based inference, surface-pattern matching, and confidence-via-repetition) using cross-instance bug reports, same-repo docstrings, adversarial near-misses, and same-function contradictions; difficulty is then a composable post-hoc dial (prompt fuzzing, distractor scaling, indirection, fact fragmentation), so length and noise are decoupled from the underlying instance. MEMGYM-DR injects a four-tier distractor hierarchy (natural / near-miss / adversarial-contradiction / bulk filler) and scales each instance to a target budget (10K–1M tokens). MEMGYM-DR additionally requires a load-bearing *fictionalization* stage: an LLM extracts entities (methods, models, organizations, people, numbers, years), generates fictional substitutes, and a deterministic regex applies the substitution registry uniformly across questions, answers, facts, documents, and distractors. Without fictionalization, frontier models score 0.70–0.85 on MEMGYM-DR by answering from pretraining alone; with it, no-memory drops to near-zero and the memory gap rises to 0.85–0.95 (Section C.6).

Verifier and corpus. The verifier in MEMGYM-CODEQA runs three independent checks per QA pair (solvability, distractor-confusion, and question-leakage), each catching a distinct shortcut; the three checks are jointly necessary because a single-check verifier had a 62% false-positive rate. The verifier in MEMGYM-DR runs a memory-ablation curve plus an adversarial-hack check, requiring `score_all_memory` $\geq$ `score_long_context` so that curated multi-hop notes must beat the long-context dump. The current corpora are 670 verified MEMGYM-CODEQA instances (2,131 deduplicated QA pairs from a 1,000-instance candidate pool) and 1,194 verified MEMGYM-DR instances (161 3-hop, 916 4-hop, 117 5/6-hop). The shared template (hidden gold artifact, memory-only/discoverable taxonomy, composable hardening, and multi-criterion verification) generalizes to any domain with a hidden ground-truth artifact and external documentation (medical guideline adherence, legal precedent application, scientific reproduction). Per-filter thresholds, the four-source MEMGYM-CODEQA distractor taxonomy, the four-tier MEMGYM-DR distractor hierarchy, the verifier pass criteria, and the SWE-Gym container backends are in Appendix D and Appendix D.

4 Experiments: When Memory Does and Does Not Matter

The experiments map to three claims. (i) Memory's payoff is regime-dependent across three wrapped gyms: roughly neutral on coding, where progress lives in the file system, and clearly positive on dialogue and web (Section 4.2). (ii) Under controlled pressure on two unrelated synthetic axes, the same strategy ranking reproduces and A-Mem leads at the maximum-pressure point on both (Section 4.3). (iii) MEMRM ranks held-out compression events near-perfectly with calibrated probabilities and remains deployable on a characterized OOD subset (Section 4.4).

Table 2: Baseline and +memory resolve / success rates on three wrapped gyms (harness-verified). Δ in pp; *Compr.* is the wrapper compression ratio averaged across compaction-triggering episodes.

Gym	Model	Memory	n	Baseline	+Memory	Δ	Compr.
SWE-Gym	Sonnet 4.5	Summary	1041	42.8	42.8	0.0	1.47×
SWE-Gym	Haiku 4.5	Summary	1003	44.0	43.0	−1.0	1.32×
SWE-Gym	GPT-OSS-120B	Summary	1003	22.3	19.1	−3.2	1.45×
τ^2-bench	Haiku 4.5	Summary	288	50.0	**58.7**	+8.7	2.29×
τ^2-bench	Haiku 4.5	Structured	288	57.6	60.1	+2.5	1.86×
WebArena-Infinity	Haiku 4.5	Structured	140	34.3	**38.6**	+4.3	1.37×
WebArena-Infinity	Haiku 4.5	Summary	140	34.3	35.0	+0.7	1.45×

Per-track compression-ratio estimators, corpus composition, and pairing protocols are in Section C.4.

4.1 Experimental Setup

Reasoners and gyms. Wrapped-gym evaluations use Sonnet 4.5 on SWE-Gym (paired baseline-vs-memory via fork-batch replay over 1,003–1,041 SWE-bench-style instances per reasoner), and Haiku 4.5 on τ^2-bench (the 288-task base split across mock, telecom, airline, and retail) and on WebArena-Infinity (a 140-task hard slice across gmail, paypal, and gitlab; Playwright Chromium with text accessibility-tree observations and `max_steps`=50). On SWE-Gym we additionally run Haiku 4.5 and GPT-OSS-120B as cross-reasoner controls. Trajectory compaction is triggered uniformly at 100 messages or 32K context tokens, whichever fires first, and the same trigger is used for every memory strategy in a comparison.

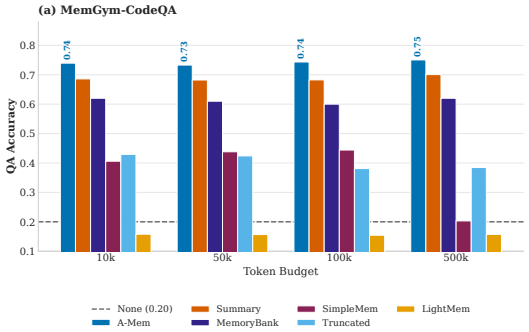
Memory strategies and synthetic benchmarks. We compare seven families: rolling Summary [44], A-Mem [50] (note-evolution), MemoryBank [58], LightMem [11], SimpleMem [27], Naive RAG [25] / BM25 retrieval (MEMGYM-DR only), and a no-memory *None* control. MEMGYM-CODEQA stresses token budgets at {10k, 50k, 100k, 500k} on the 4,289-instance verified set built from SWE-smith repositories; MEMGYM-DR stresses retrieval depth across 3-, 4-, and 5/6-hop questions on a fictionalized 100K-token deep-research pipeline. Both benchmarks use Sonnet 4.5 as the reasoner.

MEMRM training. MEMRM is Qwen3-1.7B-Base fine-tuned with QLoRA (NF4 4-bit, rank 16, α=32, target $\{q, k, v, o\}$ projections) on 18,642 SWE-Gym compression-event labels at 32K context, 600 steps on 8×A100-40GB (∼3 wall-clock hours). It is trained only on SWE-Gym compression events; OOD probes on memory-strategy and scenario axes (Table 3) are evaluation-only. Per-track configuration details (Playwright settings, τ^2-bench base-split caveat, fictionalization mechanics, and the full MEMRM recipe) are in Section C.1 and Appendix H.

4.2 Memory Across Three Wrapped Gyms

Across the three wrapped gyms, memory is roughly information-neutral on coding (where progress lives in the file system and the reasoner can re-read what was summarized away) and clearly beneficial on dialogue and web, where state hidden in past turns is expensive to re-derive (Table 2).

On SWE-Gym, the resolve-rate change tracks reasoner strength rather than the memory mechanism: Sonnet 4.5 absorbs the lossy summary at zero cost (Δ=0), Haiku 4.5 takes a small drop, and GPT-OSS-120B drops the most, reflecting weaker comprehension under compressed context. We do not see baseline-unsolvable instances rescued by adding memory: the coding tasks are difficult enough that the binding constraint is the reasoner, not the working-memory mechanism. What memory does buy on SWE-Gym is context compression (per-episode ratios of 1.32–1.47× across the three reasoners), which keeps long trajectories within the policy's effective context without changing the resolve rate. On τ^2-bench and WebArena-Infinity the picture inverts. Both involve state the reasoner cannot reliably hold across a long trajectory: multi-turn dialogue accumulates user constraints, open tool-call threads, and prior commitments hidden in earlier turns, while batch web operations require knowing which items have already been handled, information not in the rendered DOM. Once trajectories grow, memory summarizes the parts the reasoner can no longer keep in working context, and Haiku 4.5 answers correctly more often (+8.7 with Summary on τ^2, +4.3 with Structured on WebArena). The effect size therefore tracks the cost of re-deriving discarded state: low for code, high for dialogue and web. The synthetic benchmarks below isolate that pressure on a single axis.



(a) MEMGYM-CODEQA accuracy.

Strategy	3-hop	4-hop	5/6-hop
A-Mem [50]	0.709	0.540	**0.518**
BM25 [38]	**0.808**	**0.555**	0.425
Naive RAG [25]	0.753	0.537	0.442
MemoryBank [58]	0.699	0.537	0.482
SimpleMem [27]	0.614	0.467	0.415
LightMem [11]	0.610	0.467	0.400
None	0.330	0.290	0.009

(b) MEMGYM-DR judge score.

Figure 3: Memory strategies on the two synthetic-memory benchmarks. **(a)** MEMGYM-CODEQA: QA accuracy across token budgets (10k–500k); A-Mem bar values labelled and the no-memory *None* baseline shown as a dashed reference. **(b)** MEMGYM-DR: judge scores at 3-, 4-, and 5/6-hop, with column-best in bold. A-Mem leads at the maximum-pressure point on both benchmarks (500k tokens on (a), 5/6-hop on (b)); the strongest non-A-Mem baselines are rolling Summary [44] on MEMGYM-CODEQA and Naive RAG [25] / BM25 [38] on MEMGYM-DR.

4.3 Synthetic Memory Benchmarks: Head Ordering Under Pressure

Wrapped gyms tell us *when* memory pays off; the synthetic benchmarks ask which mechanism wins when the pressure is isolated to a single axis. MEMGYM-CODEQA stresses the token budget from 10k to 500k on code; MEMGYM-DR stresses retrieval depth from 3 to 5/6 hops on scientific text (Figure 3). A-Mem is the best-performing strategy at the maximum-pressure point on both benchmarks, reaching 0.75 on MEMGYM-CODEQA at the 500k-token budget (+0.55 vs. the no-memory baseline) and 0.518 on MEMGYM-DR at 5/6-hop (+0.509 vs. baseline). The strongest non-A-Mem baselines are domain-dependent (rolling Summary on the coding QA axis and Naive RAG on retrieval), both of which A-Mem beats by a comfortable margin under maximum pressure.

Without memory, both benchmarks collapse under pressure: the no-memory baseline reaches only 0.20 on MEMGYM-CODEQA and 0.009 on MEMGYM-DR's 5/6-hop slice (essentially random). Most flat baselines also lose ground as pressure rises; BM25 drops from 0.808 at 3-hop to 0.425 at 5/6-hop, confirming that the difficulty here is driven by the memory dimension rather than by task length per se. The payoff of memory therefore grows with pressure, and the gap is most visible on retrieval depth: A-Mem's lead over Naive RAG widens with hop count because note-evolution links bridge passages whose isolated query-relevance is low, which is the failure mode of a flat retriever at 5/6-hop. On the coding axis the binding constraint is the strategy rather than the budget: every memory-equipped strategy is roughly flat across the four budgets (10k–500k), and LightMem's aggressive eviction policy actually underperforms the no-memory baseline on MEMGYM-CODEQA, suggesting that policy quality matters more than window size for this working set. The same ordering reproduces on two unrelated domains with different bottlenecks; this consistency is the contribution rather than any single number. We make no domain-general claim on $n{=}2$ benchmarks, but the agreement makes A-Mem's lead unlikely to be a code-specific or scientific-text-specific artefact. Fictionalization mechanics and the no-fictionalization pilot are in Section C.6.

4.4 MEMRM: A Learned Memory Critic

MEMRM replaces a Docker rollout (minutes per event) with a sub-second classifier call that decides whether a candidate compression is SAFE to keep or HARMFUL to revert. We report three standard metrics: AUROC (rank quality of the gate's SAFE-vs-HARMFUL score, where 1.0 is perfect and 0.5 is chance), Expected Calibration Error (ECE; the gap between predicted and empirical SAFE rate over equal-mass probability bins), and Coverage (the fraction of out-of-distribution events that pass the per-axis selection rule). Table 3 reports gate quality on a held-out SWE-Gym split.

On the SWE-Gym held-out IID split, MEMRM reaches AUROC 0.985 with near-zero ECE: the gate ranks compression events near-perfectly and its predicted probabilities are calibrated. Figure 4 traces the training-time dynamics: cross-entropy descends steadily over 600 steps and IID AUROC plateaus from step 200 onward, so the headline metric is not the product of a lucky final checkpoint. We

Table 3: MEMRM gate quality on the SWE-Gym IID split and on two out-of-distribution axes (Qwen3-1.7B QLoRA). OOD rows report AUROC on the covered subset selected by a pre-declared per-axis selection rule; the rule, aggregate-OOD numbers, bootstrap protocol, and per-event detail metrics are in Appendix H and Appendix H.

Split	n	**AUROC** [95% CI]	Coverage	ECE
SWE-Gym IID	3,007	**0.985**	–	0.009
Strategy-OOD (sliding-window, masking, structured)	166	**0.714** [0.54, 0.87]	26.5%	0.850
Scenario-OOD (WebArena V2)	426	**0.748** [0.65, 0.86]	20.4%	0.237

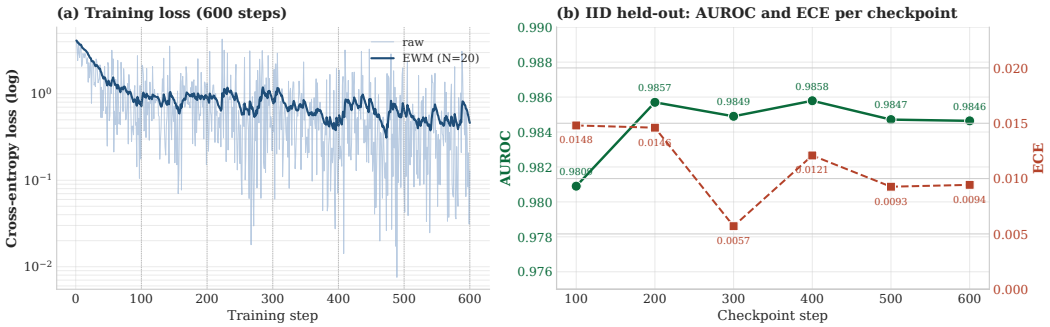


Figure 4: MEMRM training dynamics on SWE-Gym compression events (Qwen3-1.7B-Base + QLoRA, 32K context, 600 steps, 8×A100-40GB).

release this held-out split as part of the MEMGYM artifacts so other groups can evaluate their own gates against the same examples. MEMRM also shows partial generalization beyond the training distribution along two axes. Within the coding domain, the gate transfers to memory strategies that did not appear in training (sliding-window, observation-masking, and structured memories), reaching AUROC ≈ 0.71 at $\sim$27% coverage on the cohorts retained by a pre-declared per-axis selection rule based on selective classification [12]. Across domains, the gate transfers to WebArena V2 browsing trajectories and reaches AUROC $\approx$ 0.75 at $\sim$20% coverage on a class-balanced covered subset. Aggregate AUROC over the full OOD sweeps is near-random and we do not claim deployment outside the covered subset; per-track MEMRM variants are ongoing work, and the selection rule, polarity-flip diagnostics, and full training recipe are in Appendix H.

5 Conclusion

MEMGYM treats agentic memory as a first-class evaluation target rather than a number folded into task accuracy. Five tracks (τ^2-bench, SWE-Gym, WebArena-Infinity, plus the in-house MEMGYM-CODEQA and MEMGYM-DR built from length-controllable, ablation-verified synthetic pipelines) plug into a common per-step contract that wraps the prompt before it reaches the policy LLM, so a memory module can be swapped without touching the reasoner; we score each run by the difference between paired baseline-vs-memory rollouts with the same reasoner on both sides. Alongside the wrappers, MEMRM (Qwen3-1.7B fine-tuned with QLoRA) trades a multi-minute Docker rollout for a sub-second compression-quality call, which is what makes the full coding-environment evaluation academically tractable. The picture our experiments paint is regime-dependent. On the wrapped gyms, memory's payoff tracks the cost of re-deriving discarded state: roughly neutral on coding (where progress lives in the file system) and clearly positive on dialogue and web, with mid- to high-single-digit-percentage success-rate gains. On the synthetic axes, the same strategy ranking reproduces on two unrelated domains at the largest pressure point on each, with A-Mem the most robust under maximum pressure where flat retrievers fail. MEMRM achieves near-perfect IID ranking with calibrated probabilities on the SWE-Gym held-out split and remains deployable on a characterized covered subset of out-of-distribution traffic (Appendix H). We release the five-track wrappers, the MEMGYM-CODEQA and MEMGYM-DR synthetic pipelines, the MEMRM weights, and the labeled paired-trajectory corpus as artifacts; downstream training recipes that consume these trajectories are future promising research directions.

References

[1] Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. Memorybench: A benchmark for memory and continual learning in llm systems. *arXiv preprint arXiv:2510.17281*, 2025.

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

[3] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137, 2024.

[4] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

[5] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

[6] Jiayang Cheng, Dongyu Ru, Lin Qiu, Yiyang Li, Xuezhi Cao, Yangqiu Song, and Xunliang Cai. Amemgym: Interactive memory benchmarking for assistants in long-horizon conversations. *arXiv preprint arXiv:2603.01966*, 2026.

[7] Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3829–3846, 2023.

[8] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[9] Payel Das, Subhajit Chaudhury, Elliot Nelson, Igor Melnyk, Sarath Swaminathan, Sihui Dai, Aurelie Lozano, Georgios Kollias, Vijil Chenthamarakshan, Soham Dan, et al. Larimar: Large language models with episodic memory control. In *International Conference on Machine Learning (ICML)*, 2024.

[10] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[11] Jizhan Fang, Xinle Deng, Haoming Xu, Ziyan Jiang, Yuqi Tang, Ziwen Xu, Shumin Deng, Yunzhi Yao, Mengru Wang, Shuofei Qiao, et al. Lightmem: Lightweight and efficient memory-augmented generation. *arXiv preprint arXiv:2510.18866*, 2025.

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

[13] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[14] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2(3):440, 2018.

[15] Zexue He, Yu Wang, Churan Zhi, et al. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

[16] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, 2020.

[17] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[18] Yuanzhe Hu, Yu Wang, and Julian McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. *arXiv preprint arXiv:2507.05257*, 2025.

[19] Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376, 2023.

[20] 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? 2023.

[21] Greg Kamradt. Needle in a haystack – pressure testing LLMs. https://github.com/gkamradt/LLMTest_NeedleInAHaystack, 2023. GitHub repository.

[22] Rodney Kinney, Chloe Anastasiades, Russell Authur, Iz Beltagy, Jonathan Bragg, Alexandra Buraczynski, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Arman Cohan, et al. The Semantic Scholar open data platform. *arXiv preprint arXiv:2301.10140*, 2023.

[23] Dong-Ho Lee et al. Realtalk: A 21-day real-world dataset for long-term conversation. *arXiv preprint arXiv:2502.13270*, 2025.

[24] Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[26] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

[27] Jiaqi Liu, Yaofeng Su, Peng Xia, Siwei Han, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Simplemem: Efficient lifelong memory for llm agents. *arXiv preprint arXiv:2601.02553*, 2026.

[28] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[29] Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of llm agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13851–13870, 2024.

[30] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[31] Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. Memgpt: towards llms as operating systems. 2023.

[32] Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. Training software engineering agents and verifiers with SWE-Gym. *arXiv preprint arXiv:2412.21139*, 2024.

[33] Richard Yuanzhe Pang, Alicia Parrish, Nitish Joshi, Nikita Nangia, Jason Phang, Angelica Chen, Vishakh Padmakumar, Johnny Ma, Jana Thompson, He He, et al. Quality: Question answering with long input texts, yes! In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5336–5358, 2022.

[34] Joon Sung Park, Joseph C. O'Brien, Carrie J. 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.

[35] Jason Priem, Heather Piwowar, and Richard Orr. OpenAlex: A fully-open index of scholarly works, authors, venues, institutions, and concepts. *arXiv preprint arXiv:2205.01833*, 2022.

[36] Zengyi Qin, Jinyuan Chen, Yunze Man, Shengcao Cao, Ziqi Pang, Zhuoyuan Wang, Xin Sun, Gen Lin, Han Fang, Ling Zhu, et al. Osgym: Super-scalable distributed data engine for generalizable computer agents. *arXiv preprint arXiv:2511.11672*, 2025.

[37] Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[38] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*, volume 4. Now Publishers Inc, 2009.

[39] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *International Conference on Learning Representations (ICLR)*, 2024.

[40] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

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

[42] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[43] 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.

[44] Qingyue Wang, Liang Ding, Yanan Cao, Zhiliang Tian, Shi Wang, Dacheng Tao, and Li Guo. Recursively summarizing enables long-term dialogue memory in large language models. *arXiv preprint arXiv:2308.15022*, 2023.

[45] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[46] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[47] Yu Wang, Yifan Gao, Xiusi Chen, Haoming Jiang, Shiyang Li, Jingfeng Yang, Qingyu Yin, Zheng Li, Xian Li, Bing Yin, Jingbo Shang, and Julian McAuley. MemoryLLM: Towards self-updatable large language models. In *International Conference on Machine Learning (ICML)*, 2024.

[48] Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*, 2024.

[49] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

[50] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *arXiv preprint arXiv:2502.12110*, 2025.

[51] John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

[52] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018.

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

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

[55] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

[56] 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*, 2024.

[57] Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. Ama-bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*, 2026.

[58] 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*, volume 38, pages 19724–19731, 2024.

[59] Shuyan Zhou. WebArena-Infinity: Generating browser environments with verifiable tasks at scale. *shuyanzhou.com*, March 2026. URL https://webarena.dev/webarena-infinity/.

[60] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Contents

A Appendix Overview

The appendix is organized around the artifacts referenced in the main text. Appendix E documents the full memory-operation catalog including the operations × environments cross-product (Table 11). Appendix F gives the per-environment trajectory schemas, the multi-runtime container backends, the replay-and-fork protocol, and the observation-replay runner. Appendix G lists the five SWE-Gym evaluation-harness bugs we patched. Appendix D and Appendix D together cover the two construction pipelines: per-filter rationales, three-pass behavioral fact extraction, and four-source distractor taxonomy for MEMGYM-CODEQA; deep-research configuration, four-tier distractor hierarchy, and verifier pass criteria for MEMGYM-DR. Appendix H gives the full MEMRM training recipe (data, LoRA, threshold sweep, held-out metrics). Appendix I lists per-environment evaluation hyperparameters. Appendix B expands the main-text related-work section with detailed coverage of agentic memory systems, context compression, long-horizon agent benchmarks, and reward models. Appendix C expands the main-text experiments with per-track setup, per-app and pilot breakdowns for the wrapped gyms, the MEMGYM-DR fictionalization story plus the full per-strategy MEMGYM-DR matrix, and the MEMRM data-augmentation taxonomy.

B Extended Related Work

B.1 Agentic Memory Systems (Detailed)

The earliest line of memory-augmented LLMs treated memory as a way to compensate for finite context windows in long-form assistant settings. MemoryBank [58] stores and updates user memories for long-term companionship; MemGPT [31] treats the prompt as a virtual memory hierarchy, paging between working and archival context; ReadAgent [24] builds episodic gist memories for long-document reading with optional look-up into the source text. Reflexion [41] introduced verbal self-reflection as an episodic-memory mechanism, in which the agent records natural-language critiques after each trial and conditions on them in subsequent attempts; this preceded the current "memory-augmented agent" line and remains a useful baseline mechanism for agent-side memory.

A broader line of work treats memory primarily as a *prompting pattern* rather than an architectural change. Reflective and self-revision prompting (Reflexion [41], Self-Refine [28], Self-Consistency [46], and Tree of Thoughts [53]) uses prior outputs or critiques as in-context memory to guide later generations. Action-grounded reasoning prompts (ReAct [54] and Toolformer [40]) interleave thought, action, and observation traces that themselves become a working-memory record. Persona- and role-driven memory has been studied through generative-agent simulations that combine episodic logs, reflective summaries, and retrieval-by-importance into a memory stream [34], and through skill-library agents that distill repeated experience into reusable code or natural-language rules [43, 56]. Hierarchical and graph-structured retrieval treats memory as an indexed knowledge structure rather than a flat history: tree-organized recursive abstraction [39], community-summary graphs [10], and neurobiologically inspired hippocampal indexing [13]. Recursive summarization explicitly compresses dialogue history into a maintained summary state across turns [44]. Production-oriented memory stacks [8] and parametric long-term memory architectures [47, 9] push memory either out to a managed service or into the model weights themselves. Retrieval-augmented self-critique extends memory use to the retrieval policy itself [2]. These works span architecture, prompting strategy, and infrastructure; what they share is that the memory mechanism is benchmarked on a single domain (chat, code, web, or open-domain QA) using either end-task success or recall-style probes. MEMGYM is complementary: it fixes the memory interface across five tracks and uses paired memory-isolated scoring so any of the above mechanisms can be plugged in and compared on the same memory event.

A-Mem [50] introduced an explicitly agentic memory mechanism: each new interaction is converted into a structured note, linked to related memories, and allowed to evolve existing memory representations. It evaluates on LoCoMo [29], a very-long-term conversational-memory benchmark built from 300-turn dialogues averaging 9K tokens across up to 35 sessions, with question answering, event summarization, and multimodal dialogue generation tasks. LongMemEval [48] adds six explicit memory competencies (information extraction, multi-session reasoning, knowledge update, temporal reasoning, single-session preference, abstention) over scalable conversational histories. MemoryAgentBench [18] adds incremental multi-turn ingestion to test how an agent updates memory as new information arrives. MemoryBench [1] uses simulated user feedback to drive continual learning

across sessions. REALTALK [23] provides a 21-day real-world conversation dataset for long-term dialogue memory. MemoryArena [15] evaluates agent memory in interdependent multi-session agentic tasks, where later sessions depend on facts established earlier. Most recently, AMA-Bench [57] evaluates memory over real and synthetic agentic trajectories with QA, while AMemGym [6] provides on-policy interactions with simulated users and structured latent-state evolution. Across these benchmarks, the memory target is a transcript, a feedback stream, or a personalized-dialogue state; none provides a unified memory interface with memory-isolated rewards across coding, search, tool dialogue, and web control.

B.2 Context Compression and Long-Context Evaluation

A complementary line of work studies how to compress or evaluate long contexts in isolation, rather than how an agent should manage memory across an interaction. LLMLingua [19] compresses prompts at the token level using a small language model to score perplexity, removing low-information tokens before the prompt is sent to the target model. AutoCompressors [7] learn soft summary vectors end-to-end so a long context can be replaced by a short sequence of compressed embeddings. Both target inference-time efficiency and assume a static prompt, in contrast to MEMRM, which scores compression decisions taken during multi-step interaction.

Long-context evaluation is anchored by Needle-in-a-Haystack [21], RULER [17], LongBench [3], and LongBench v2 [4], which probe what an LLM can recall from a single long input. Multi-hop QA benchmarks (HotpotQA [52], 2WikiMultiHopQA [16], MuSiQue [42]) and long-document QA such as QuALITY [33] test compositional retrieval and reasoning over a fixed corpus. These benchmarks measure what can be *recalled* from a static long context. Agentic memory must additionally decide *what to preserve* across an interaction in which the context is generated turn-by-turn under tool-use pressure; MEMGYM measures the latter and uses memory-isolated paired rollouts to keep the comparison clean.

B.3 Long-Horizon Agentic Benchmarks (Detailed)

SWE-bench [20] evaluates agents on real GitHub issue resolution, with the original test split spanning thousands of bug reports across mature Python repositories. SWE-Gym [32] adds 2,438 Python training tasks with executable runtimes, unit tests, and released trajectories. We use SWE-Gym heavily because coding trajectories expose natural memory events: repeated evidence, stale observations, failed hypotheses, and patch-specific facts that survive multiple debugging steps. The cost is substantial: full evaluations require Dockerized repositories, test execution, and many interaction steps, so end-task resolve rate alone is too expensive and too sparse for systematic memory iteration. The OpenHands framework [45] provides the strategy-library lineage on which our memory-operation library is built.

τ-bench and τ^2-bench [55, 5] evaluate tool-agent-user workflows in domains such as retail, airline, and telecom, but report end-task success without isolating whether failures came from memory, policy, or tool use. WebArena [60] provides fully functional web environments across e-commerce, forums, collaborative software development, and content management; WebArena-Infinity [59] scales this idea by automatically generating self-contained web applications with verifiable tasks. These web benchmarks are realistic but require browser/server infrastructure and multi-step exploration, and their metrics conflate memory with navigation and planning. OSWorld [49] broadens computer-use evaluation to desktop and operating-system tasks under VM-style infrastructure with GUI grounding; OSGym [36] extends this to a gym-style training surface. Across these benchmarks, memory is load-bearing but not separately measured. MEMGYM wraps such environments with an explicit memory boundary, records compression events, and reports memory-isolated scores, enabling memory systems to be compared independently of the underlying agent's reasoning, retrieval, and tool-use ability.

B.4 Reward Models and Learned Evaluators

Reward models trained from human or automated feedback have become a central tool for post-training LLMs [30], and process reward models [26] extend this to step-level supervision over reasoning traces. MEMRM is such a reward model specialized to memory: it supplies a step-level scalar that generalizes binary task success into a signal usable both for inference-time gating (a sub-second classifier replacing per-episode Docker rollouts) and for RL post-training as a dense

reward channel. MEMRM is deliberately not a world model in the Ha–Schmidhuber sense [14], which predicts environment dynamics in a latent space; it predicts only whether a candidate compression would change behavior given the recorded action.

C Extended Experimental Details

This section expands Section 4.1 (per-track configuration), Section 4.2 (per-app and pilot rows), Section 4.3 (fictionalization mechanics for MEMGYM-DR), and Section 4.4 (MEMRM data-augmentation taxonomy).

C.1 Per-Track Experimental Setup

Coding track. We evaluate on two SWE-Gym splits across 11 Python repositories: a 500-instance diverse subset spanning all repositories and a 554-instance long-trajectory subset drawn from issues whose baseline trajectory exceeds the compaction trigger. Most baseline numbers in Table 2 are reported on the combined ∼1,000-instance set; the per-strategy hyperparameter sweeps in Table 5 use the smaller n=20 `moto` subset for cost reasons. The agent scaffold is mini-swe-agent v2.2.4 with a 250-step limit. Models: Claude Sonnet 4.5, Claude Haiku 4.5, and GPT-OSS-120B via Bedrock. For LLM-based memory strategies, the summarization model is Claude Haiku 4.5, reducing summarization cost by roughly $4\times$ compared to using the main reasoning model. Evaluation uses the official swebench harness, identical to the public leaderboard.

MEMGYM-DR track. We report the 100K-token deep-research pipeline with Claude Haiku 4.5 as the worker and Claude Sonnet 4.5 as the verifier and judge. Historical no-fictionalization runs diagnose parametric leakage; the Apr. 12 paper run characterizes the fictionalized pipeline at scale.

Dialogue track. We evaluate on all 288 τ^2-bench tasks across the mock, telecom, airline, and retail domains using Haiku 4.5 on Bedrock. Results are paired baseline-vs-memory runs in `mode=both`; the current headline numbers are on the base split, with official test-split reruns pending server-side support for `-task_split_name`.

Web GUI track. We evaluate WebArena-Infinity hard tasks with Haiku 4.5, Playwright-controlled Chromium, text accessibility-tree observations, and a `max_steps`=50 cap. The current web results use two protocols: smart replay on 170 hard tasks across five apps, and prefix injection on 140 hard tasks across three apps. Smart replay compares memory strategies after replaying recorded baseline actions; prefix injection fixes the early trajectory prefix and measures whether memory improves the live tail.

Compute. MEMRM training (Qwen3-1.7B QLoRA, full recipe in Appendix H) runs on $8\times$A100-40GB for ∼3 hours (≈ 2h45m training + 22 min tokenization). Wrapped-gym evaluation is API-bound: per-task wall-clock is dominated by tool execution and reasoner latency rather than memory-manager compute. Preliminary and failed experiments not reported in the paper consumed roughly $2\times$ the reported training compute.

C.2 Track-Strategy Evaluation Grid

Table 4 enumerates every (track, strategy) cell evaluated in the paper, with the summarizer and reasoner model used in each. Cells marked ✓ were verified against the recorded run configuration; the only contaminated cell is MEMGYM-DR × Structured (gpt-4o-mini summarizer used in error before Bedrock Haiku 4.5 became the documented default), reported in Table 8 for completeness and excluded from the main-text figure.

The grid contains 17 verified cells and 1 contaminated cell. Summarizer model is Bedrock Haiku 4.5 throughout (single exception above); reasoner is Sonnet 4.5 on SWE-Gym and MEMGYM-CODEQA, Haiku 4.5 on τ^2-bench and WebArena-Infinity, and Haiku 4.5 (worker) + Sonnet 4.5 (verifier) on MEMGYM-DR. Per-track hyperparameters are listed in Appendix I.

C.3 Asset Licenses

Third-party assets are used under their respective open licenses. SWE-Gym [32], SWE-bench [20], τ^2-bench [5], SWE-smith [51], and the OpenHands condenser interface [45] are released under the

Table 4: Audit surface: (track, strategy) cells evaluated, with summarizer and reasoner. ✓ = configuration verified; × = contaminated (one cell). *None* is the no-memory control. Cells are dashes where the strategy is not applicable to the track (e.g., BM25/Naive RAG only apply to MEMGYM-DR).

Strategy	SWE-Gym	τ^2-bench	WebArena-Inf.	MEMGYM-CODEQA	MEMGYM-DR
None (control)	✓	✓	✓	✓	✓
Summary	✓	✓	–	✓	–
Structured	✓	✓	✓	–	×
A-Mem	–	–	–	✓	✓
MemoryBank	–	–	–	✓	✓
LightMem	–	–	–	✓	✓
SimpleMem	–	–	–	✓	✓
BM25 / Naive RAG	–	–	–	–	✓
Truncated	–	–	–	✓	–

MIT license. WebArena [60], the Qwen3-1.7B base model used for MEMRM, and GPT-OSS-120B (used as one of the trajectory-collection reasoners) are released under Apache-2.0. Closed-source reasoners (Claude Sonnet 4.5 and Claude Haiku 4.5) are accessed via the AWS Bedrock API under Anthropic's commercial terms of use. The MemGym wrappers, the paired-trajectory corpus, and the synthetic MEMGYM-CODEQA / MEMGYM-DR instances are released under MIT; MEMRM weights inherit the Apache-2.0 license of the Qwen3-1.7B base.

C.4 Wrapped-Gym Compression-Ratio Estimators and Pairing Protocol

Corpus composition and pairing. For SWE-Gym we pool the released diverse subset and the long-trajectory subset (1,054 instances across 11 Python repositories). Instances whose baseline trajectory never reaches the compaction trigger (≤ 100 messages) produce identical baseline and +memory rollouts and are therefore excluded from the paired comparison; the +Memory column in Table 2 reuses the baseline n for these instances. The two τ^2-bench baseline rows (50.0 vs. 57.6 in Table 2) are paired against independently sampled rollouts, so the within-row Δ is the paired statistic. WebArena-Infinity uses a 50% prefix-injection ablation in which a randomly selected prefix of the trajectory is replayed verbatim into the agent before measurement begins.

Per-track compression-ratio formulas. Let episode e have K_e compaction events and $r_{e,k} = T^{\text{pre}}_{e,k}/T^{\text{post}}_{e,k}$ denote the per-event input-token ratio (tokens before compaction divided by tokens after). Write $\mathcal{E}^+ = \{e : K_e \geq 1\}$ for the set of compaction-triggering episodes and $T^{\max}_e$, T^{end}_e for the peak and final token counts within episode e. The three reporters differ in which terms the wrapper persists:

$$\overline{r}_{\text{SWE}} = \frac{1}{|\mathcal{E}^+|} \sum_{e \in \mathcal{E}^+} \frac{1}{K_e} \sum_{k=1}^{K_e} r_{e,k},$$

$$\overline{r}_{\tau^2} = \frac{1}{|\mathcal{E}^+|} \sum_{e \in \mathcal{E}^+} r_{e,K_e},$$

$$\overline{r}_{\text{WA}} = \frac{1}{|\mathcal{E}^{++}|} \sum_{e \in \mathcal{E}^{++}} \frac{T^{\max}_e}{T^{\text{end}}_e}, \qquad \mathcal{E}^{++} = \left\{e \in \mathcal{E}^+ : T^{\max}_e / T^{\text{end}}_e \geq 1.05\right\}.$$

SWE-Gym averages every per-event ratio (wrapper field `memory_stats.avg_compression_ratio`); τ^2-bench reports only the last compaction event per episode (`wrapper_last_compression_ratio`); WebArena derives the ratio from peak and final token counts at episode end and filters episodes whose agent terminated at peak (uninformative ratio of 1.0). All three estimators are conditional on $K_e \geq 1$ so episodes that never triggered compaction do not pull the mean toward 1.0.

C.5 Wrapped Gyms: Per-App and Pilot Breakdowns

From pairs to full coverage. The fork-batch protocol only forks instances whose baseline trajectory crosses the compaction-trigger threshold (more than 100 messages). Below that threshold no com-

Table 5: Extended memory configuration sweep on SWE-Gym (Sonnet 4.5, `moto` subset, n=20, baseline $12/20 = 60\%$, Haiku 4.5 summarizer). *Resolved*: harness verdict on the final patch; *LimExc*: trajectories hitting the 250-step ceiling. The two columns are independent and a trajectory may count under both. Run IDs map to `experiments.md`.

Run	Strategy	Params	Resolved	LimExc	Compress	Δ vs base
6a	none (baseline)	—	12 (60%)	0	1.00×	0 pp
6c	structured_summary	ms=100, r=0.75, kf=3	12 (60%)	5	1.50×	0 pp
6d	structured_summary	ms=200, r=0.75, kf=1	12 (60%)	0	1.00×	0 pp
6e	structured_summary	ms=100, r=0.5, kf=1	11 (55%)	14	2.28×	−5 pp
6b	observation_masking	w=100	10 (50%)	7	1.16×	−10 pp
7a	llm_summarizing	ms=100, r=0.75, kf=3	12 (60%)	0	1.35×	0 pp
7b	observation_masking (sel.)	w=100, kf=3	12 (60%)	7	1.08×	0 pp
7c	sliding_window	w=75, kf=3	12 (60%)	9	1.23×	0 pp
7e	pipeline (mask→struct)	w=100, ms=100, r=0.75, kf=3	12 (60%)	4	1.42×	0 pp

paction event fires and the +memory rollout is bit-identical to the baseline, so only a fraction of the full ∼1,000-instance evaluation set produces a non-trivial paired comparison: $678/1041$ (65.1%) for Sonnet 4.5, $653/1003$ (65.1%) for Haiku 4.5, and $556/1003$ (55.4%) for GPT-OSS-120B. The lower coverage on GPT-OSS reflects its higher rate of early termination at the 250-step ceiling rather than a memory effect. Reporting the +Memory rate against the full evaluation set requires extrapolating the paired delta back over the entire scale baseline:

$$\begin{aligned}\text{full } +\text{memory resolved } &= \text{paired memory resolved} \\ &\quad + \big(\text{scale baseline resolved} - \text{paired baseline resolved}\big),\end{aligned}$$

which is equivalent to the weighted mean $\Delta_{\text{full}} = \Delta_{\text{paired}} \times (n_{\text{paired}}/n_{\text{scale}})$. Plugging in the values above yields $446/1041$, $431/1003$, and $192/1003$ as the full-1k +memory resolution counts for Sonnet 4.5, Haiku 4.5, and GPT-OSS-120B respectively. All conditional Δ statistics (R→R, R→U, U→R, U→U transition counts; per-split breakdowns) are reported on the paired subset only.

Hyperparameter sweep. Table 5 reports a Sonnet 4.5 sweep over five strategy families (free-text summarization, selective observation masking, sliding-window, structured summarization at three trigger densities, and a chained masking-then-structured pipeline) against a fixed n=20 `moto` subset whose baseline resolves 12 of 20 instances (60%). The summarizer model is held at Haiku 4.5 throughout. None of the strategies improves resolve rate at the pilot size, but they differ substantially in compression ratio achieved and in how often they trip the limits-exceeded ceiling, providing a controlled view of the cost side of memory.

τ^2-bench gains are domain-conditional. The aggregate +8.7pp Summary gain on τ^2-bench masks substantial domain heterogeneity. Table 6 disaggregates the gain across the four base-split domains (mock, telecom, airline, retail; n=288): Summary memory contributes its largest paired delta on telecom (+17.5pp on n=114) and a near-zero delta on retail, while structured memory wins on airline (+14.0pp on n=50). The per-domain Δ in each row uses that strategy's own independently sampled baseline rollouts (consistent with the cross-row caveat noted under Table 2). Compression columns report the per-episode last-event ratio averaged over compaction-triggering episodes within each domain; the Total cells reproduce the run-level aggregator from Table 2 and may differ slightly from a domain-weighted mean since the runtime accumulates the global last ratio at run scope rather than averaging per-domain task means.

The mock domain is an LLM-simulated user (rather than the canonical scripted user) and has historically been the highest-variance slice of τ^2; readers comparing to scripted-domain numbers should weight it accordingly. The dominant Summary gain (+17.5pp) sits on telecom, where dialogues are long enough to compact 18 times per episode (median) but the per-episode last-ratio (1.38×) is among the least aggressive, consistent with kept-floor logic limiting how much can be pruned. Airline shows the largest Structured gain (+14.0pp) at a moderate compression of 1.76×. Retail compresses most aggressively (2.05–2.46×) but yields the smallest paired Δ, illustrating that compression aggressiveness and task gain are not co-aligned.

WebArena-Infinity: gains on stateful apps. Table 7 disaggregates the WebArena rows of Table 2 across two evaluation protocols: a 170-task smart-replay sweep across five apps (gmail, superhuman, linear, paypal, gitlab) and a 140-task prefix-injection ablation that controls for the effect of replaying

Table 6: τ^2-bench per-domain results with Haiku 4.5 (base-split, n=288 over mock/telecom/airline/retail). Baseline columns pair with the corresponding strategy row's independently sampled baseline rollouts (notes of Table 2); Δ is the within-row paired statistic. Last-ratio columns are the episode-mean of `wrapper_last_compression_ratio` over compaction-triggering episodes; mock has zero compaction events under either strategy at this kl horizon (dialogues are short). Total ratios are reproduced from Table 2's run-level aggregator. Numbers from `probe_tau2_per_domain.py`.

		Summary (paired)		Structured (paired)		Compr. (last-event)	
Domain	n	+Summ	Δ	+Struct	Δ	Summ	Struct
mock	10	50.0	0.0	50.0	0.0	—	—
telecom	114	57.0	+17.5	50.0	+1.8	1.38×	1.24×
airline	50	54.0	+6.0	70.0	+14.0	2.45×	1.76×
retail	114	63.2	+1.8	66.7	−1.8	2.46×	2.05×
Total	**288**	**58.7**	**+8.7**	**60.1**	**+2.5**	**2.29×**	**1.86×**

Table 7: WebArena-Infinity per-app results with Haiku 4.5 and text accessibility-tree observations. Smart replay uses 170 tasks across gmail, superhuman, linear, paypal, and gitlab. Prefix injection uses 140 tasks across gmail, paypal, and gitlab.

Smart replay: baseline and memory on matched hard tasks

App	Tasks	none replay	summ. ms=10	summ. ms=15	struct. ms=10
gmail	20	4/20 (20%)	6/20 (30%)	7/20 (35%)	**9/20 (45%)**
superhuman	15	5/15 (33%)	**7/15 (47%)**	5/15 (33%)	6/15 (40%)
linear	15	2/15 (13%)	**6/15 (40%)**	3/15 (20%)	5/15 (33%)
paypal	60	33/60 (55%)	33/60 (55%)	33/60 (55%)	**35/60 (58%)**
gitlab	60	4/60 (7%)	5/60 (8%)	5/60 (8%)	**6/60 (10%)**
Total	**170**	**48/170 (28.2%)**	**57/170 (33.5%)**	**53/170 (31.2%)**	**61/170 (35.9%)**

Prefix control: live baseline, replay-only control, and memory after matched prefix

App	Tasks	live baseline	none + prefix	summ. + prefix	struct. + prefix
gmail	20	6/20 (30%)	9/20 (45%)	8/20 (40%)	**10/20 (50%)**
gitlab	60	4/60 (6.7%)	7/60 (11.7%)	8/60 (13.3%)	**10/60 (16.7%)**
paypal	60	33/60 (55%)	32/60 (53.3%)	33/60 (55%)	**34/60 (56.7%)**
Total	**140**	**43/140 (30.7%)**	**48/140 (34.3%)**	**49/140 (35.0%)**	**54/140 (38.6%)**

the trajectory prefix itself. The two protocols agree on which apps benefit from memory and which are flat, isolating the memory effect from the replay effect.

In smart replay, structured memory improves aggregate success by 7.6 points over the replayed no-memory control (28.2% to 35.9%). The largest app-level gain is gmail, where structured memory rises from 20% to 45%, because multi-email batch operations require remembering which items have already been handled. Linear also benefits strongly, rising from 13% to 40% with summarizing memory. Paypal and gitlab are mostly flat: paypal tasks are short and rarely trigger useful condensation, while gitlab is near the capability floor for Haiku 4.5. The prefix-injection ablation separates two effects: replaying the early trajectory itself improves the live baseline from 30.7% to 34.3%, and structured memory still adds another 4.3 points on top of the matched prefix. The final held-out app split and multimodal screenshot ablation remain pending.

Preliminary online MEMRM acceleration. On the SWE-Gym online A/B, the median effective compression ratio when the MEMRM gate accepts a candidate compression is $3.46\times$. Aggregate wall-clock speedup against the full Docker rollout requires a complete benchmark sweep and is being measured; it is not reported in the main text.

Table 8: Memory strategy comparison on the latest MEMGYM-DR verified corpus. Scores are continuous judge-scores in $[0, 1]$; intervals are 95% CIs computed as $1.96 \cdot \text{SE}$. The structured row is contaminated; see paragraph above.

Strategy	3-hop (n=161)	4-hop (n=916)	5/6-hop (n=117)
Passthrough (no memory)	0.330 ± 0.049	0.290 ± 0.014	0.009 ± 0.011
BM25 over notes [38]	$\mathbf{0.808 \pm 0.037}$	$\mathbf{0.555 \pm 0.017}$	0.425 ± 0.048
Naive RAG [25]	0.753 ± 0.039	0.537 ± 0.016	0.442 ± 0.045
Structured summary	0.684 ± 0.045	0.340 ± 0.017	0.257 ± 0.035
A-Mem [50]	0.709 ± 0.043	0.540 ± 0.016	$\mathbf{0.518 \pm 0.037}$
LightMem [11]	0.610 ± 0.049	0.467 ± 0.017	0.400 ± 0.043
MemoryBank [58]	0.699 ± 0.043	0.537 ± 0.016	0.482 ± 0.041

C.6 MEMGYM-DR Fictionalization and Strategy Detail

Fictionalization. Early no-fictionalization runs failed for the wrong reason: the model answered from parametric knowledge. In the Apr. 5 100K-token pilot, no-memory scores remained at 0.70–0.85 even though the task was meant to require retention across turns. Fictionalization is therefore the load-bearing intervention for the retrieval track: it replaces real entities and numbers as retrieved documents enter the pipeline, then sanitizes the constructed instance before verification so the agent cannot recover the answer from pretraining alone. After fictionalization was combined with verifier and retry fixes, the Apr. 12 paper run reached a mean no-memory score of 0.113, a mean all-memory score of 0.808, and a mean gap of 0.694; verified-instance yield is being re-audited.

Per-strategy MEMGYM-DR scores. Table 8 reproduces the full strategy-by-hop matrix that backs Figure 3(b). The *structured* row was contaminated by an upstream summarizer-model misconfiguration (gpt-4o-mini was used in error before Bedrock Haiku 4.5 became the documented default) and is reported here for completeness only; it must not be cited as a strategy result. Every other (track, strategy) cell in Section C.2 was verified against its recorded summarizer-model and reasoner-model configuration; this is the only cell affected.

C.7 MEMRM Data Augmentation Taxonomy

Each MEMRM training example is constructed from a recorded SWE-Gym compression event and a counterfactual perturbation of that event. The unperturbed event provides the SAFE label whenever continuing from the compressed view does not alter downstream task behavior; the perturbed counterpart is labeled HARMFUL. Table 9 lists the six perturbation families, grouped into three categories: indiscriminate degradation of the summary surface (`aggressive_0.5`, `random_drop_0.2`), targeted removal or substitution of task-relevant content (`attr_delete_paths`, `summary_redaction`, `summary_noise`), and history truncation that bypasses the summary entirely (`truncate_last_10`). Together they cover the dominant failure modes a deployed condenser would induce: information loss, plausible-but-incorrect content, and loss of recent context.

Table 9: MEMRM data-augmentation taxonomy. Each operation produces a HARMFUL counterpart from the same SAFE source compression event.

Operation	**Construction**
`aggressive_0.5`	Replace 50% of summary tokens with noise or blanks, modeling an aggressive but content-blind compression.
`random_drop_0.2`	Randomly drop 20% of summary sentences, modeling lossy summarizers that omit content uniformly at random.
`attr_delete_paths`	Remove file-path attributes from structured summary fields, modeling schema-aware summarizers that under-record locator information.
`summary_redaction`	Redact key entities and values referenced later in the trajectory, modeling over-compression that loses task-critical specifics.
`summary_noise`	Inject plausible-but-incorrect facts (wrong file names, wrong test outcomes), modeling hallucinated summarizer output.
`truncate_last_10`	Drop the last 10 messages from the agent view, modeling a sliding-window policy that discards recent context not yet folded into the summary.

Applied to 846 unique (instance, fork-step) pairs together with multi-compaction events, the six operations yield 18,637 labeled events: 16,357 HARMFUL (87.8%) and 2,280 SAFE (12.2%). The repo-grouped split is 15,630 train / 3,007 eval; Table 3 reports gate quality on the eval split.

D Synthetic Pipeline Details

Shared construction template. Both synthetic tracks follow the same control flow: source adoption, memory-dependence transformation, task hardening, distractor/noise injection, length scaling, leakage removal, and verifier certification. For MEMGYM-DR, the source is a research topic plus retrieved academic documents; for MEMGYM-CODEQA, it is a SWE-smith bug plus hidden patch evidence. The accepted instance stores the source pointer, grounding facts, distractor provenance, verifier scores, token-budget metadata, and the fields needed by the MEMGYM trajectory recorder.

MEMGYM-DR deep-research configuration. The active paper pipeline uses a 100K-token target budget, Haiku 4.5 as the worker model, and Sonnet 4.5 as the verifier. A typical instance uses 4 target hops and 5 search results per hop. The *grow* stage starts from a topic, iteratively searches arXiv (a local SQLite FTS5 index for rate-limit-free volume), Semantic Scholar, OpenAlex, and Wikipedia, and builds a dependent bridge-fact chain; *craft* expands the chain into a multi-turn research session with natural, near-miss, adversarial, and bulk distractors; *scale* fills the requested budget with hybrid filler ($\sim 70\%$ real search results, $\sim 30\%$ LLM-generated academic prose); *fictionalize* applies post-hoc entity substitution after the *scale* stage (an LLM extracts entities and a deterministic regex applies the substitution registry across questions, answers, facts, documents, and distractors), so that filler text is fictionalized as well; and *verify* runs memory ablation, long-context, and adversarial-hack checks. The legacy filtered-multihop-QA pipeline (v2) was deprecated after plateauing on its eviction-gap target and is not used in any reported result.

MEMGYM-DR four-tier distractor hierarchy. The *craft* stage injects four distractor types, each blocking a different shortcut: (1) *natural distractors* drawn from the same searches that produced the gold facts (block topic-recognition); (2) *near-miss rewrites* of gold facts with one entity perturbed (block surface-pattern matching); (3) *adversarial contradictions* about the correct entities (block majority voting across documents); and (4) *bulk academic-style filler* (load the context window without adding signal, exposing the agent's compression policy under volume).

MEMGYM-DR verifier pass criteria. A verified instance must satisfy all of: (a) `score_all_memory` ≥ 0.85 (the agent answers correctly when given the curated bridge facts as notes), (b) `score_long_context` ≤ 0.90 (the answer is *not* findable by dumping all documents into a single context window), (c) `score_no_memory` ≤ 0.5 (the answer is *not* findable from any single document in isolation), and (d) `score_all_memory` $\geq$ `score_long_context` (curated multi-hop notes must beat the long-context dump). Conditions (b)–(d) jointly operationalize the "not RAG" claim of Section 3.4: an instance survives only if memory genuinely matters above and beyond what dropping all documents into context would achieve.

MEMGYM-CODEQA configuration. MEMGYM-CODEQA starts from SWE-smith instances and filters for nontrivial patches (at least 5 changed lines, 2 hunks, 2 failing tests, and a non-short problem statement). The prescreen rejects bugs that are solvable from the repository and vague prompt alone. The extraction stage combines static patch analysis, LLM behavioral-fact extraction, and verifier relabeling of whether each fact is discoverable from the post-fix repository. Accepted instances require at least two critical memory-only facts. QA conversion then creates single-fact and multi-fact questions, adds adversarial answer distractors, verifies solvability/distractor/leakage conditions, deduplicates near-duplicate pairs, and evaluates under the *Evicted protocol*: the QA question is asked only after the conversation prefix carrying the gold memory facts has been evicted from context, so the agent must rely on whatever the memory module retained rather than scrolling back to the raw debugging trace.

MEMGYM-CODEQA per-filter rationale. Each raw-data filter encodes a specific assumption about what makes an instance "memory-needing", motivated by what would otherwise contaminate the eval. *Patch size* 5–500 *changed lines*: smaller patches are typically single-symbol fixes recoverable from the test name; larger patches are multi-feature refactors that confound memory effects with planning effects. *At least one structural diff hunk and one* `FAIL_TO_PASS` *test*: ensures the bug has a localized behavioral signature, so a behavioral fact about *this bug* is well-defined. *Problem statement* ≥ 100 *characters*: rules out one-line bug reports that under-specify the symptom. *LLM pre-screen pass*:

the operational definition of "memory-needing" (Section 3.4); if the bug is solvable from problem statement plus repository alone, the instance is dropped because no memory record is needed.

MEMGYM-CODEQA fact extraction. The extraction stage runs three passes designed to expose facts that the repository itself does not reveal: (i) with the gold patch visible, the LLM seeds candidate questions about what a developer would need to know (not what code to change); (ii) with the patch hidden, the LLM extracts facts from the problem statement and test names alone, labeling each as `discoverable` (recoverable from the post-fix repository) or `memory-only` (visible only from the debugging record); (iii) the discoverability labels are re-examined with the full repository context. The instance-retention criterion (at least two critical `memory-only` facts) is the threshold below which the question reduces to "search the repo".

MEMGYM-CODEQA distractor taxonomy. Distractors are drawn from four sources, each targeting a different shortcut: *cross-instance bug reports* block topical inference; *same-repo docstrings* block conventions-based inference; *adversarial near-misses* block surface-pattern matching; and *same-function contradictions* block confidence-via-repetition. Memory files interleave these distractors with critical facts and are written as natural documents (incident reports, debug notes, code review comments), mimicking the documentation a developer would actually have rather than a clean Q/A index. Difficulty is then a composable post-hoc dial (prompt fuzzing, distractor scaling, indirection, fact fragmentation) with preset combinations mapping to easy/medium/hard, so length and noise are decoupled from the underlying instance.

MEMGYM-CODEQA verification breakdown. Table 10 reports the per-check breakdown of the three-check verifier on the 1,000-instance candidate pool referenced in Section 3.4. The table makes the failure-mode distribution explicit: the largest single failure category is distractor-confusion (a same-conversation-shape session of an unrelated repo lets the agent answer), which is exactly the failure the three-check design was added to catch.

Table 10: MEMGYM-CODEQA three-check verification on the 1,000-instance candidate pool (7,690 raw QA pairs). Checks: solvability (A), distractor-confusion (B), question-leakage (C); an instance is kept if any pair passes. Final yield: 670 instances, 2,131 QA pairs after dedup.

QA-level result	Count (%)
Valid (passes all three checks)	2,972 (38.6%)
Broken (fails solvability)	1,325 (17.2%)
Confusable (fails distractor check)	1,642 (21.4%)
Leaky (fails leakage check)	1,103 (14.3%)
Confusable and leaky	648 (8.4%)
Instance-level: kept	673 / 1,000 (67.3%)
After dedup: final instances	670
After dedup: final QA pairs	2,131

E Memory Strategy Implementation Details

LLM summarizing. The LLM summarizing strategy is closely aligned with the OpenHands condenser implementation. Key hyperparameters: `max_size` (default 100 messages, triggers compaction when exceeded), `keep_first` (default 1, number of initial messages pinned), and `ratio` (default 0.75, fraction of non-pinned messages to compress). Compaction works by splitting the view into a pinned head, a compressible prefix, and a recent tail. The prefix is sent to a summarization LLM with instructions to produce a concise summary of actions, findings, and decisions. The resulting summary replaces the prefix, and a `CondensationRecord` is stored to enable persistent rebuilding of the condensed view.

Structured summary. Uses function calling with 17 explicit fields: current objective, files inspected, files modified, test results, errors encountered, hypotheses, confirmed findings, rejected approaches, open questions, dependencies, environment state, commands run, code changes, remaining steps, confidence level, blockers, and key decisions. The LLM fills these fields via a structured output call, and the result is formatted into a machine-readable state document that replaces the compressed prefix.

Observation masking. Replaces the content of old tool-call responses with `<MASKED: observation too old>`. The attention window (default 100 messages) determines how many recent observations remain visible. This is a zero-LLM-cost strategy that reduces token count without rewriting content.

IR memory baselines. The MEMGYM-DR track includes six memory managers. *Passthrough* exposes no retained notes beyond the current prompt and acts as the no-memory control. *BM25* retrieves notes by sparse lexical matching [38]. *Naive RAG* stores observations as independent notes and retrieves nearest notes without graph updates [25]. *Structured summary* condenses the session into a fixed research-state schema. *A-Mem* stores Zettelkasten-style notes with LLM-generated metadata, links, and memory evolution [50]. *LightMem* uses a staged memory system inspired by sensory, short-term, and long-term memory [11]; in our MEMGYM-DR adapter, its topic-aware consolidation is used as a retrieval context for the final answer. The upstream `METADATA_GENERATE_PROMPT` ships a LOCOMO chat-only Personal-Information-Extractor template that primes the metadata stage to look for biographical facts in conversational text. When fed non-conversational MEMGYM-DR content (paragraphs of scientific prose), this template causes the metadata stage to emit empty or hallucinated entries, which propagate to the vector index and degrade retrieval. We replace it at call time with a domain-neutral storage prompt modeled on A-Mem's `METADATA_PROMPT`, threaded through the per-call `METADATA_GENERATE_PROMPT` keyword of `LightMemory.add_memory` (no upstream fork). The output schema $\{$`source_id`, `fact`$\}$ is preserved unchanged so the Qdrant indexer remains byte-compatible. We deliberately do *not* thread the per-task `task_prompt` into the storage stage; storage stays task-agnostic (mirroring A-Mem's discipline), and the task description is consumed only at retrieval time for query rewrite. This change preserves apples-to-apples cross-method comparison at the storage layer.

Operations $\times$ environments cross-product. Table 11 lists the operation families evaluated as primary baselines and which environment each is exercised in. All operations implement the same `manage_context` contract and compose via `PipelineMemory`; per-environment strategies select which to instantiate, with environment-specific field schemas. Three additional ported operations (observation masking, sliding-window with pinned anchors, adaptive token budget) are implemented but not used as primary baselines in this paper.

Table 11: Memory operations evaluated as primary baselines in MEMGYM. ✓/✗ indicate whether the operation is exercised in the released environment configuration.

Operation	Dialogue	Search	Coding	Web
Passthrough (no filtering)	✓	✓	✓	✓
LLM summarizing (rolling)	✓	✓	✓	✓
Structured summary (per-environment schema)	✓ (8-field)	✗	✓ (17-field)	✓ (14-field)
Retrieval-style memory[‡]	✗	✓	✗	✗

[‡]BM25 [38], RAG [25], A-Mem [50], LightMem [11].

F Gym Wrappers and Trajectory Schema

Wrapper placement. All environments implement memory by wrapping the prompt sent to the policy LLM. The underlying gym maintains its native state and full interaction history, while the memory manager receives the accumulated message history before each model call and returns the filtered context actually shown to the reasoning model. This choice keeps the gym implementation unchanged, makes memory strategies portable across environments, and lets the recorder log both the raw history statistics and the filtered view.

Per-step trajectory record. Each step is serialized as a `Step` record in `adapter/trajectory_recorder.py`, carrying the step index and timestamp, the raw `observation`, the agent's `reasoning_action` and `memory_action`, the environment-returned `reward`, `terminated`, and `truncated` flags, and the filtered `context` actually shown to the reasoning model. The accompanying `FilteredContext.metadata` dict carries the post-filter `tokens` count, the pre-filter `original_tokens` count, the `compression_ratio`, the `was_compacted` flag, and the `strategy` identifier. The episode header stores the task identifier and run-level configuration. For LLM-based compression, the record additionally stores the generated summary or structured

state and the indices of the messages covered by the compaction. These fields are sufficient to reconstruct the context seen by the reasoning model, identify rising-edge compression events, and build supervised examples for MEMRM or policy mid-training.

Replay-and-fork protocol. For Docker-backed coding tasks, MEMGYM additionally stores the executable action trace and fork metadata. A fork-batch run replays the baseline tool actions until the first step at which the memory manager would trigger compaction under the target strategy. At that step, replay mode stops: the memory-conditioned agent receives the same repository/container state and continues with live model calls, memory summaries, and tool observations. If a baseline trajectory never exceeds the compaction threshold, it is skipped as structurally ineligible. The paired record stores the fork step, parent instance, baseline outcome, fork outcome, final patch score, and token-efficiency statistics. This protocol is exposed in the released trajectories so users can separate effects of memory compression from effects of different early search paths.

G SWE-Gym Evaluation Harness Fixes

Scaling SWE-Gym evaluation to all 11 repositories required fixing five bugs in the swebench evaluation harness:

1. **Mypy test syntax:** The pytest `-k` flag cannot parse mypy's `[case]` test markers. We implemented custom log parsing for mypy test output.
2. **Docker pull timeout:** The default 60-second timeout was insufficient for 500MB–2GB images. Increased to 600 seconds.
3. **Image cache thrashing:** The `cache_level="env"` setting deleted pulled images after each instance. Auto-switched to `cache_level="instance"`.
4. **Pandas conda crash:** Bash `set -u` with unset conda variables caused crashes. Wrapped conda activation with `set +u`/`set -u`.
5. **Pydantic log parser:** Pydantic's output format was incompatible with the default log parser. Switched to `parse_log_pytest_v2`.

These fixes added approximately 37 resolved instances for Sonnet 4.5 (129→166) and 15 for GPT-OSS-120B (77→92) on the 500-instance evaluation, underscoring the importance of correct evaluation infrastructure.

H MEMRM Training Details

Task and data. MEMRM is a binary classifier over *compression events*. Each example is a (prompt, label) pair where the prompt serializes the (context-before, compressed-context, candidate-action) triple introduced in Section 3.3 (i.e., the pre-compression context, the proposed compression summary, and the candidate next action sampled under the compressed view), together with a short task descriptor; the label is `Y`/`N` indicating whether the compressed trajectory is SAFE (task still solvable). Labels come from three sources: (i) episode-level resolution on the parent trajectory, (ii) counterfactual replay where an aggressive/lenient/random-drop perturbation causes an action divergence in Docker, and (iii) LLM-as-judge on the forgotten content. The augmented corpus contains 18,637 labeled events (Table 3) drawn from Sonnet 4.5, GPT-OSS-120B, and Haiku 4.5 fork-batch rollouts on the SWE-Gym diverse subset; the repo-grouped split yields 15,630 train / 3,007 eval examples.

Model and optimization.

- Base: Qwen3-1.7B-Base [37] with QLoRA adapters (NF4 4-bit, r$=16$, $\alpha=32$, dropout 0) on `q/k/v/o_proj`; max sequence length 32,768, gradient checkpointing, flash attention; per-checkpoint adapter size $\sim$25.7 MB.
- Loss: class-weighted cross-entropy ($w_{\text{SAFE}}\approx3.0$, $w_{\text{HARMFUL}}\approx0.57$, cap 3.0) on the single-token label.
- Schedule: 600 steps, lr 5×10^{-5}, batch 1 per GPU $\times8$ GPUs, gradient accumulation 4 (effective batch 32), bf16, cosine warmup 0.05, completion-only masking via TRL 0.16.

- Decision threshold: t^*=0.88, selected on held-out validation under the constraints HARMFUL-F1 ≥ 0.90 and SAFE-precision ≥ 0.80.

Held-out metrics. AUROC 0.9847, accuracy 0.972 at t^*, SAFE-F1 0.861 at t^*, HARMFUL-F1 0.987, ECE 0.0093, HARMFUL false-alarm rate 0.003. Compared to a vanilla-CE baseline (A0), C1 improves ECE by 3× and halves the HARMFUL false-alarm rate.

Online A/B smoke test. Effective-context compression was 3.46× at the median (4,139 tokens memory-on vs. 14,322 tokens memory-off). A known OOD concern: on Qwen3.6-Thinking trajectories, the gate saturates at reject-rate 1.0; broadening the training distribution to additional reasoning-trace styles is part of ongoing work.

OOD scope. MEMRM is trained on SWE-Gym compression events; out-of-distribution (OOD) behavior is not implied by the IID metrics of Table 3. We characterize OOD scope along two axes: a *strategy-OOD* axis (memory mechanisms unseen in training: sliding-window, observation-masking, structured with varied trigger budgets, and pipeline-mask; n=166) and a *scenario-OOD* axis (WebArena V2 browsing trajectories, an agent domain disjoint from the SWE-bench training corpus; n=426). Aggregate AUROC on the full sweeps is near-random on both axes (≈0.43 on each), reflecting heterogeneous per-cohort behavior: MEMRM is calibrated on a subset of cohorts and uncalibrated or polarity-inverted on others. Rather than averaging this heterogeneity into a single misleading number, we adopt *selective classification* [12]: a pre-declared rule decides which inputs fall within MEMRM's deployable scope, and we report performance on the covered subset alongside the coverage rate.

Selection rule. We retain a cohort c if its per-cohort $\mathrm{AUROC}(c)>0.5$ (the random-baseline floor) and its sample size $n(c) \geq N_{\text{axis}}$, with N_{strategy}=20 and N_{scenario}=30. The asymmetric thresholds follow the cohort-size distributions (a discrete cliff between n=23 and n=43 on strategy-OOD, with no cohorts in between); both are pre-declared, not tuned. Tightening to a uniform $N\geq30$ collapses strategy-OOD coverage to 0% but leaves scenario-OOD unchanged, and $N\geq40$ matches $N\geq30$ on both axes; thus N_{strategy}=20 is an exploratory threshold that surfaces the structured-memory cohorts where ranking transfers, while the scenario-OOD covered subset is robust to the threshold choice in $\{30, 40\}$.

Within-scope reading. The scenario-OOD covered subset (n=87, two `linear` WebArena cohorts) is class-balanced, so all metrics agree: AUROC 0.748, AUPRC 0.785, HARMFUL-F1 0.747 at t^*, ECE 0.237, a deployable operating point on a WebArena cohort, not just a deployable rank-order. The strategy-OOD covered subset (n=44, two structured-memory cohorts) is class-imbalanced, so we restrict the strong claim to AUROC 0.714 [0.544, 0.872]: MEMRM ranks pairs above chance on these cohorts, while threshold-dependent metrics on 2 HARMFUL pairs are uninformative.

Out-of-scope reading. On most strategy-OOD cohorts MEMRM is uncalibrated (near-random ranking); on one WebArena cohort (`linear_summ_ms15`, n=30) it is *systematically inverted* (per-cohort AUROC 0.27 → 0.73 under a polarity-flip rule), a recoverable failure mode there but not elsewhere. We do not claim deployment outside the covered subset; per-track MEMRM variants are part of ongoing work. We also exclude τ^2-bench from this analysis: its runtime persists neither `step.context` nor the post-compaction `summary` text for compaction-positive steps, so the (history, task, compressed memory) input triplet that anchors Table 3 cannot be reconstructed from on-disk artefacts; a rigorous-input τ^2 OOD evaluation is deferred to future work.

I Per-Track Hyperparameters

- **SWE-Gym coding.** mini-swe-agent v2.2.4, 250-step limit, Sonnet 4.5 / GPT-OSS-120B reasoners, Haiku 4.5 summarizer. `max_size`= 100, `condensation_ratio`= 0.75, `keep_first`= 3.
- **MEMGYM-DR retrieval.** 100K-token budget, Haiku 4.5 worker, Sonnet 4.5 verifier; four-tier distractor hierarchy (natural, enhanced, adversarial, bulk); fictionalization Phase 1 LLM entity pass followed by Phase 2 regex substitution.
- **τ^2-bench dialogue.** Haiku 4.5 on Bedrock, 288-task base split, paired mode=both, summarizing configuration `ms=10, kl=2`, structured configuration `ms=10, kl=4` (where `ms` is `max_size`, the message-count compaction trigger, and `kl` is `keep_last`, the number of recent messages preserved verbatim); official test-split rerun pending.

- **WebArena-Infinity.** Playwright-driven Chromium, text-default accessibility-tree observations, Haiku 4.5 policy, `max_steps`$= 50$. Smart replay uses 170 hard tasks across gmail, superhuman, linear, paypal, and gitlab with `ms=10/15`; prefix injection uses 140 hard tasks across gmail, paypal, and gitlab with `prefix_fraction`$= 0.5$.

J Discussion, Limitations, and Future Work

J.1 Discussion

MEMGYM explores a new, practical direction for memory evaluation (memory-isolated paired scoring across coding, retrieval, dialogue, and web tracks, with a learned gate (MEMRM) replacing per-event Docker rollouts) and provides a unified playground for studying agentic memory as a foundational capability rather than as a benchmark-specific add-on.

J.2 Broader Impacts

MEMGYM is a benchmark for LLM-agent memory rather than a deployed system. Positive impact: MEMRM-gated scoring cuts per-event memory evaluation from minutes of Docker rollout to a sub-second scalar, lowering the compute floor for academic memory research; the released paired-trajectory corpus and pipelines further reduce duplicated infrastructure work. Negative impacts are inherited from the underlying agent benchmarks rather than introduced by memory itself: web-automation benchmarks could be repurposed for spam or scraping, and coding agents could be repurposed to generate exploits. We mitigate by releasing on top of public benchmarks under their original terms, by limiting MEMRM to a 1.7B reward classifier with no generative misuse surface beyond its Qwen3 base, and by documenting fictionalization and verifier safeguards that make the synthetic tracks reproducible without exposing real personal data.

J.3 Limitations

The main limitation of this work is data scale: per-track pilots and the MEMRM training set are sized to support the headline comparisons reported in §4 but are not yet large enough to characterize every cell of the strategy $\times$ reasoner $\times$ track grid at full statistical power. Scaling the trajectory corpus and broadening per-track coverage are ongoing.

J.4 Future Work

The natural next step is post-training: turning MEMGYM's paired trajectories and MEMRM reward into supervision for *learned* memory policies. Concretely, an agent can be supervised-fine-tuned on the SAFE compressions in the corpus and then improved with RL post-training using MEMRM as the critic, so the policy learns when to keep, summarize, or evict context conditioned on task type and memory state. Instantiating this SFT+RL recipe across the five tracks and reporting the resulting downstream gains is the main planned extension of this work.