AGENTCL: TOWARD RIGOROUS EVALUATION OF CONTINUAL LEARNING IN LANGUAGE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Language agents spend substantial inference time solving individual tasks, yet the experience acquired in one episode is often underutilized in future episodes. Continual learning expects an agent to accumulate experience across a stream of tasks, improve over time, and avoid interference from irrelevant experiences. Unfortunately, existing benchmarks struggle to evaluate this problem rigorously. Most efforts focus on retrieval and reasoning over long-context conversations or documents, while recent lifelong-adaptation benchmarks often stream existing datasets whose cross-task dependencies are unknown, making it difficult to understand what an agent learns and reuses over time. This paper presents an evaluation framework AGENTCL for continual learning in agents, centered on distinguishable task streams and metrics for CL properties. AGENTCL distinguishes three cross-task relations by the availability of reusable knowledge. In dependent streams, later tasks can reuse knowledge from earlier ones. In conventional streams, whether such reuse exists is unknown. Besides, independent tasks are held out with little transferable knowledge. We use the benchmark to evaluate non-parametric memory designs for continual learning. To diagnose how memory design choices affect continual learning, we develop MEMPROBE, a probing method that stores interactions, insights, and skills, while filtering unreliable experiences during consolidation. Empirical analysis across coding, deep research, and language understanding/reasoning tasks shows that conventional streams offer limited ability to distinguish memory designs, whereas dependent streams more clearly distinguish their plasticity. Meanwhile, conventional streams and independent tasks often yield limited gains and can expose memory-induced degradation. These results highlight the need for stronger memory designs that balance plasticity and stable reuse.

1 INTRODUCTION

Language agents (Su et al., 2024) powered by large language models (LLMs) spend substantial inference time interacting with complex environments, yet the experience acquired in each episode is often underutilized before the next one begins. The fact that intensive inference does not automatically compound into long-term competence motivates the *continual learning (CL) problem for language agents*: given a stream of tasks, can an agent accumulate reusable experience, successfully transfer it to future environments, and remain robust as its memory scales?

However, evaluating this capability challenges the conventional evaluation paradigms of machine learning. Traditionally, LLMs are trained and evaluated in static batches under the i.i.d. assumption. In contrast, agents often operate in a sequential setting with limited task experience, where cross-task relations are essential to measure continual learning. Also, a delicate balance between *plasticity* (adapting to new tasks) and *stability* (retaining past knowledge) is required during the learning process (Wang et al., 2024a). Given these unique characteristics, we argue that a rigorous benchmark for agentic CL research must satisfy two core requirements: 1) **Realistic task relations:** continual learning typically unfolds around the work process within a domain, so a benchmark should simulate how knowledge is reused in a task stream, rather than chaining arbitrary and isolated tasks into a sequence. 2) **Metrics for CL properties:** the evaluation protocol must decouple and quantify the properties of plasticity, stability, and generalization within the learning process beyond simply reporting average accuracy.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

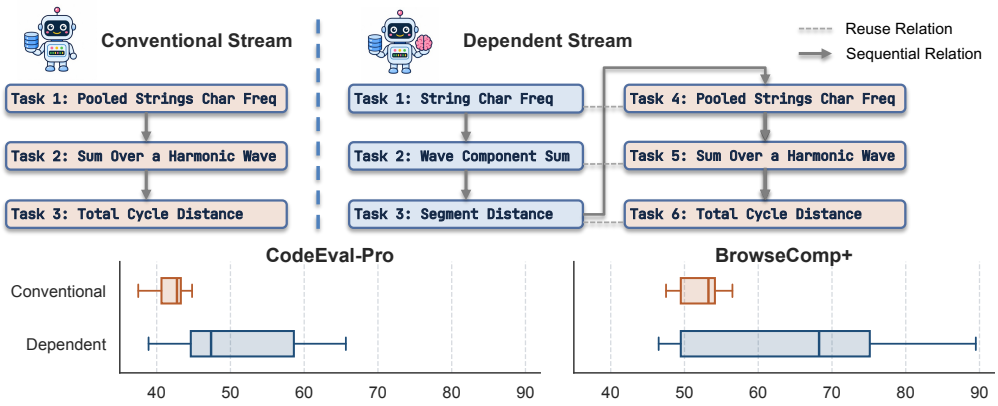

Figure 1: AGENTCL with task streams. *Reuse relation* denotes a task pair in which the latter one can reuse the solution to the former one. *Sequential relation* denotes the order of task execution. **Top**: conventional streams order tasks from the same domain with unknown reuse relations, while dependent streams include reusable subtasks before related complex tasks. **Bottom**: boxplots of average accuracy across evaluated methods on CodeEval-Pro and BrowseComp+, showing that dependent streams yield larger method separation than conventional streams for memory reuse.

While more benchmarks focus on long-term agentic interactions, they fundamentally struggle to satisfy these requirements simultaneously. First, long-term memory benchmarks (Maharana et al., 2024; Wu et al., 2025; Hu et al., 2025) evaluate whether agents can retrieve and reason over extended documents or conversations, yet they rely on static corpora rather than dynamic task adaptation. Second, recent lifelong or streaming benchmarks (Wu et al., 2024; Zheng et al., 2025; Wei et al., 2025b) move closer to sequential adaptation, but they typically stream (i.i.d.) tasks from existing datasets, leaving cross-task relations unknown. It remains ambiguous whether an agent genuinely reused an abstracted experience or merely benefited from incidental task overlap. Thus, they simply lack the scenarios to evaluate agentic experience reuse. Besides, metric design is essential for evaluation, while existing benchmarks often rely on average accuracy, leaving CL properties such as plasticity and stability largely overlooked.

To bridge these fundamental gaps, we introduce AGENTCL, an evaluation framework for CL in language agents with **task streams** and **metrics**. As shown in Figure 1, AGENTCL distinguishes three cross-task relations. *Conventional streams* leave cross-task reuse unknown, as in i.i.d.-built benchmarks. Our proposed *dependent streams* simulate capability-centered progression, where later tasks can reuse knowledge from earlier ones. *Independent tasks* are held out from a different source, with little reusable knowledge. Coupled with a two-pass evaluation protocol, AGENTCL allows us to separate and compute explicit metrics for *Plasticity Gain*, *Stability Gain*, and *Generalization Gain*.

We leverage AGENTCL to conduct a comprehensive evaluation of non-parametric memory designs, a paradigm for inference-time learning without costly training. We further develop MEMPROBE, a non-parametric memory probing method for empirical studies. Our empirical analysis across coding, deep research, and language reasoning tasks reveals a critical benchmarking insight: conventional streams offer surprisingly weak discriminative power, compressing the performance differences among memory designs. Conversely, our dependent streams clearly make experience reuse traceable and show varying plasticity across methods. Moreover, our evaluation uncovers an unresolved stability bottleneck: while existing memory designs achieve notable transfer gains when reusable knowledge is present, they frequently induce memory interference or performance degradation on conventional streams and independent tasks. These findings underscore the urgent need for memory designs that transcend passive retrieval to achieve robust, balanced consolidation.

Our core contributions are threefold: 1) We formalize AGENTCL, the first evaluation framework that explicitly distinguishes cross-task relations alongside metrics for plasticity, stability, and generalization. 2) We construct dependent streams, conventional streams, and independent held-out tasks spanning coding, deep research, and reasoning domains to capture whether reusable knowledge is available. 3) Comprehensive evaluation shows that dependent streams substantially improve the discriminative power of evaluation, while exposing critical stability bottlenecks in existing memory designs.

2 Related Work on Continual Learning in Language Agents

2.1 Benchmarks

The streaming setting has been studied in prior evaluation work. StreamBench (Wu et al., 2024) evaluates whether language agents can improve over an input–feedback sequence, while Memory-Bench (Ai et al., 2025) focuses on whether LLM systems can construct useful memories from user feedback during service time. LifelongAgentBench (Zheng et al., 2025) and SWE-Bench-CL (Joshi et al., 2025) instantiate this idea in interactive agent and software-engineering domains, respectively, by organizing tasks into sequential settings that measure knowledge accumulation, transfer, and forgetting. Evo-Memory (Wei et al., 2025b) extends this direction by benchmarking self-evolving memory across diverse task streams and comparing multiple memory designs. Concurrent with our work, Continual Learning Bench (Asawa et al., 2026) similarly evaluates agents over multi-episode environments with task relevance, such as recurring database schemas, debugging patterns, and forecasting dynamics. While these benchmarks adopt online or streaming evaluation protocols, AGENTCL differs by explicitly contrasting different cross-task relation structures and by assuming no detailed user feedback for CL.

2.2 Methodology

Non-parametric memory records contexts in an intuitive way, but its applications to long-context or streaming tasks have not been rigorously evaluated for CL yet. Systems like Mem0 (Chhikara et al., 2025) and A-MEM (Xu et al., 2025) focus on storing and organizing long-term memories for **adaptive retrieval**. Another line emphasizes **procedural reuse**: Agent Workflow Memory (Wang et al., 2025) induces reusable workflows from prior trajectories, and Dynamic Cheatsheet (Suzgun et al., 2026) maintains concise strategies, code snippets, and abstractions that can be reused at test time. ExpRAG and Evo-Memory's ReMem (Wei et al., 2025b) accumulate task experiences online and retrieve them in later tasks under **streaming** evaluation. Our evaluation targets not only how to store and retrieve experience, but also studies the learning dynamics from streams when experience reuse is available and when it is not.

Beyond non-parametric memory, recent work studies CL via parameter or policy updates in agentic settings. Agent-Dice (Wu et al., 2026) addresses stability–plasticity by disentangling shared and conflicting knowledge during parameter fusion for GUI and tool-use agents. CGL (Yao et al., 2026) explores continual GUI learning with supervised and reinforcement fine-tuning to balance new-domain adaptation and capability retention. ACuRL (Xue et al., 2026) autonomously explores environments, synthesizes capability-matched tasks, and updates policies across environments. These training-based methods internalize reusable experience into model parameters, but often require reward models, rollouts, and repeated optimization. ***We instead focus on non-parametric memory as a lightweight inference-time adaptation. Evaluating training-based approaches with* AGENTCL *is a direction for future work.***

3 AgentCL: Evaluation Framework for Continual Learning

3.1 Problem Formulation

An agent observes a sequence of tasks $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_T\}$ and solves them sequentially through interaction with the environment. In contrast to maintaining long contexts for each task, the agent maintains a persistent memory $\{M_t\}_{t=0}^{T}$ shared across the task stream $\mathcal{T}$. For each task τ_t, the agent may first retrieve relevant information $c_t = \mathcal{R}(M_{t-1}, \tau_t)$ from the current memory, and then interact with the environment conditioned on c_t. After completing the task, it updates the memory via $M_t = \mathcal{U}(M_{t-1}, \xi_t)$, where ξ_t denotes the trajectory collected during τ_t. Given the sparsity of feedback in real-world online environments, we adopt a strict setting where the evaluated agent operates without any ground-truth guidance during memory construction, leaving the LLM to synthesize memories entirely on its own. In contrast to a memoryless setting, the order of tasks matters because the agent's accessible experience evolves over time. Thus, we will proceed to discuss two core issues in benchmarking: the design of task streams and metrics.

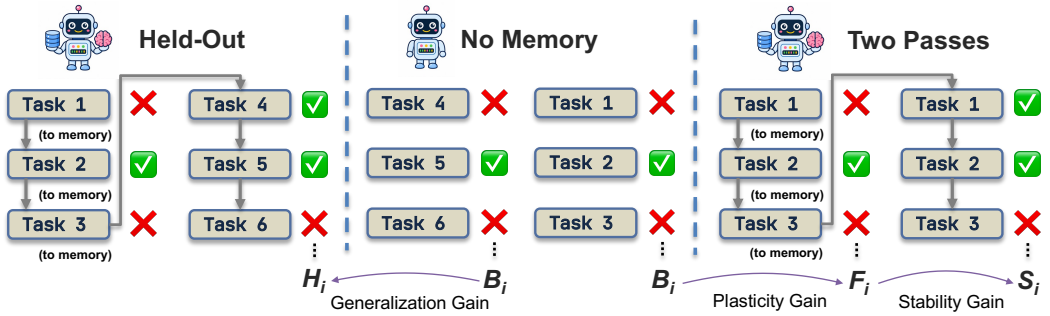

Figure 2: Evaluation protocol and metrics of AGENTCL. **Middle**: memoryless baseline. **Right**: two passes are performed on the same stream. The first pass allows both reading from and writing to memory, whereas the second pass (or independent evaluation) is read-only. **Left**: independent tasks are not used to build memory. For task i, B_i, F_i, S_i, and H_i denote the performance of the memoryless method, the first-pass performance, the second-pass performance, and the performance on the independent task, respectively. Plasticity Gain measures whether memory built from earlier tasks helps on later tasks. Stability Gain measures whether experience gained from solving a task becomes persistently reusable. Generalization Gain measures transfer to independent tasks after memory construction.

3.2 TASK RELATIONS AND STREAMS

Task relations and ordering determine what can be learned and evaluated. Conventional task streams fail to capture a key property of real-world work: noise coexists with reusable experience. We therefore distinguish three cross-task relations by the availability of reusable knowledge. The first two are instantiated as task streams, on which memory is both built and evaluated. The third consists of held-out tasks evaluated with the memory frozen after either stream. 1) **Conventional:** tasks come from the same environment or domain, but whether reusable knowledge exists between them is unknown. This is the typical state of benchmarks built under an i.i.d. setting, which does not reflect continual learning around a capability within a domain. 2) **Dependent:** the stream simulates such capability-centered progression, where later tasks can reuse knowledge from earlier ones. These dependencies are hidden from the agent, which is never told which earlier task is relevant. Unrelated task pairs act as distractors for each other. 3) **Independent:** evaluated tasks are drawn from a different source than the memory-building stream, so that little transferable knowledge is available by design.

3.3 METRICS

We first report basic metrics, such as accuracy, success rate, or progress rate, depending on the criterion of each environment. Then, we focus on CL properties: *plasticity*, *stability*, and *generalization* (Wang et al., 2024a), as shown in Figure 2. These properties from classical and parametric CL manifest differently in agentic settings. Traditionally, stability is often framed as avoiding catastrophic forgetting after new tasks are absorbed into model weights. In contrast, we do not force an agent to learn through training, but mainly focus on non-parametric memory as the approach. In this case, past experiences can be explicitly preserved as trajectories, summaries, skills, or other records. The central stability challenge is therefore not whether past episodes are recorded, but whether they can still be retrieved from memory and utilized correctly after many later updates.

This distinction from parametric CL motivates our two-pass evaluation protocol. The agent processes a task stream sequentially. The **first pass** permits both memory retrieval and updating, as described in §3.1. The **second pass** is read-only with frozen memory. Let B_i denote the performance on task i under a memoryless setting. Let F_i denote the performance on task i during the first pass, after the agent has processed tasks 1 to $i-1$. Let S_i denote the performance on the same task i during the second pass with frozen memory. We define

$$PG_i = F_i - B_i, \tag{1}$$

$$SG_i = S_i - F_i. \tag{2}$$

Table 1: The number of tasks in our constructed streams. * denotes our synthesized data.

Datasets	Dependent Stream	Conventional Stream
CodeEval-Pro (BigCodeBench-Lite-Pro) (Yu et al., 2025)	48 subtasks → 48 complex tasks	48 complex tasks
BrowseComp+ (Chen et al., 2025)	308 subtasks* → 100 complex tasks	100 complex tasks
AgentBoard BabyAI (Ma et al., 2024)	28 subtasks → 28 complex tasks	40 tasks
AgentBoard ScienceWorld (Ma et al., 2024)	–	90 tasks
MMLU-Pro (Wang et al., 2024b)	–	300 tasks

Plasticity Gain (**PG**) measures whether experience accumulated from preceding tasks helps the agent solve the current task. It captures the ability of a memory system to turn earlier episodes into useful context for later tasks.

Stability Gain (**SG**) measures persistent reuse under stream-level memory consolidation. As F_i is observed *before* the experience from task i is written into memory, SG captures the net effect of writing the current task experience into memory, preserving it through subsequent updates, and retrieving it from the final frozen memory state. Thus, SG measures whether stream-level consolidation turns a task experience into stable support that remains accessible after later episodes are added. Low or negative SG reflects a failure of stable reuse, where the relevant experience is either poorly consolidated, difficult to retrieve, or overwhelmed by memory interference.

We further use independent tasks to evaluate generalization. Let H_j denote the performance on an independent task j, evaluated after constructing and freezing memory on a task stream. We define

$$GG_j = H_j - B_j. \tag{3}$$

Generalization Gain (**GG**) measures whether the constructed memory generalizes beyond the observed tasks used to build the memory. Together, AGENTCL combines the designed task streams and metrics, as shown in Figure 2. Both conventional and dependent streams are used for two-pass evaluation. Different passes of a stream use the same tasks and sequence.

4 BENCHMARK CONSTRUCTION

4.1 DATA COLLECTION

Stream Design. Table 1 summarizes the constructed streams. In real work within a domain, solutions to earlier, simpler tasks can often be reused in later, more complex ones. However, such reusable experience is sparse among many unrelated tasks, and the relations between tasks are usually implicit. We construct each dependent stream by placing randomly ordered subtasks before the complex tasks of its conventional counterpart, and keep the subtask–task relations hidden from the agent. As a result, each complex task can reuse the solution to an earlier subtask, yet on CodeEval-Pro and BrowseComp+ only about 1% of the stored episodes are relevant when it arrives, and these episodes lie about half a stream earlier, so the agent must identify reusable experience among many distractors. Since both streams evaluate exactly the same complex tasks, performance differences are attributable to the availability of reusable experience. The construction of specific benchmarks across environments is detailed below.

Coding. We build coding streams from CodeEval-Pro self-invoking problem pairs (Yu et al., 2025), which contain both BigCodeBench-Lite-Pro and HumanEval-Pro. Each pair consists of a base problem (subtask) and a self-invoking problem (complex task) whose solution requires the solution to the base problem. Each task pair, from subtask to complex task, is guaranteed to exhibit a function reuse relation, as shown in Appendix Figure 5. Our filtered dataset includes 96 tasks (48 pairs) from BigCodeBench-Lite-Pro. The conventional stream randomly orders all complex tasks, which are diverse: their solutions may share a few Python packages, but their underlying logic differs without clear reuse relations. We use 120 tasks from HumanEval-Pro as independent tasks in §5.3.

Deep Research. We build deep research streams based on BrowseComp+ (Chen et al., 2025). BrowseComp+ provides an offline environment with a collected corpus from BrowseComp (Wei et al., 2025a). We construct the streams similarly: the conventional stream contains 100 randomly sampled BrowseComp+ tasks, while the dependent stream prepends 308 subtasks. The subtasks

are synthesized from these 100 parent tasks by GPT-5.2 (OpenAI, 2025). The synthesis takes the original QA pair and the supporting documents as input, and yields subtasks and their answers. Each subtask shares a subset of evidence documents with its parent complex task. We set the reuse relations between subtasks and their corresponding complex tasks without hinting at the answers, as illustrated by the examples in Appendix B.1 and details in Appendix D.

Language Understanding & Reasoning. To analyze streams from existing datasets, we select the single-turn MMLU-Pro and the multi-turn AgentBoard BabyAI and ScienceWorld tasks from Evo-Memory (Wei et al., 2025b). For MMLU-Pro (Wang et al., 2024b), its tested knowledge is widely distributed, and we take it as a conventional stream. We selected three subsets (economics, engineering, and philosophy) and sampled 100 tasks from each subset. For AgentBoard ScienceWorld (Ma et al., 2024), we take all the 90 original tasks as a conventional stream, and also evaluate a block stream in Appendix B.8. For AgentBoard BabyAI (Ma et al., 2024), we build two streams, detailed in Appendix B.13. The conventional stream contains 40 tasks from ten diverse easy subtask families, with four tasks per family. The dependent stream contains 56 tasks, where each task pair links a simpler subtask to a related compositional extension.

4.2 Quality Control

Most tasks used in our benchmark are directly adopted and restructured from existing resources. For CodeEval-Pro, we apply simple filtering during restructuring. For synthesized BrowseComp+ subtasks, we apply stricter quality control. Each candidate subtask must be evidence-supported, deterministic, and logically relevant to the parent task, and is further checked by a separate verifier for entity specificity, evidence support, determinism, and answer completeness. We also report diagnostics in Appendix D.2 to confirm that the subtasks do not collapse into parent-task paraphrases: the mean token-set Jaccard similarity between subtasks and their parent is 0.10, no pair exceeds 0.30, only 5.8% of subtasks cover the full parent evidence set, and only 1.6% share both the normalized answer and the evidence set with the parent task. These controls support compositional reuse through a chain of evidence documents rather than direct answer exposure.

5 Experiments

5.1 Setup

We evaluate representative methods from four categories. 1) **Memoryless** agent denoted as ReAct (Yao et al., 2023). 2) **Adaptive memory** methods that flexibly determine storage and retrieval: LangMem (LangChain AI, 2026) and Mem0 (Chhikara et al., 2025). 3) Memory methods with **procedural knowledge**: Agent Workflow Memory (AWM) (Wang et al., 2025) and Dynamic Cheatsheet (DC-RS) (Suzgun et al., 2026). 4) **Self-evolving memory**: ExpRAG, Evo-Memory's ReMem (Wei et al., 2025b), and MemProbe. These categories span the main non-parametric designs that can approach CL in language agents. We provide a detailed discussion of their coverage and differences in Appendix B.2.

Given the tool support, we use Qwen3.5-35B-A3B (Qwen Team, 2026) with the stateful Qwen-Agent code interpreter by default, and gpt-oss-120b (Agarwal et al., 2025) with its natively trained browser tool for deep research. Qwen3-Embedding-8B (Zhang et al., 2025) serves as the embedding model. Experiments with other backbones show similar trends (Appendix B.9). For cases where public code was unavailable or not directly applicable to our tasks, we reimplemented the method and matched the original workflow and prompts as closely as possible. We report averages over three runs for coding tasks. For all memory methods, we uniformly retrieve the top-2 memory chunks to serve as context for the current task, with more discussion in Appendix B.10.

5.2 MemProbe: Probing Memory Design

To diagnose how different components in memory designs affect CL, we introduce a non-parametric memory probing method, MemProbe, which analyzes how agents accumulate and consolidate experience across a task stream while balancing plasticity and stability. It adopts a retrieve–solve–consolidate loop over the task stream. Before solving a task, it retrieves semantically similar prior episodes as the reference context. After solving, it validates the agent responses and consolidates them

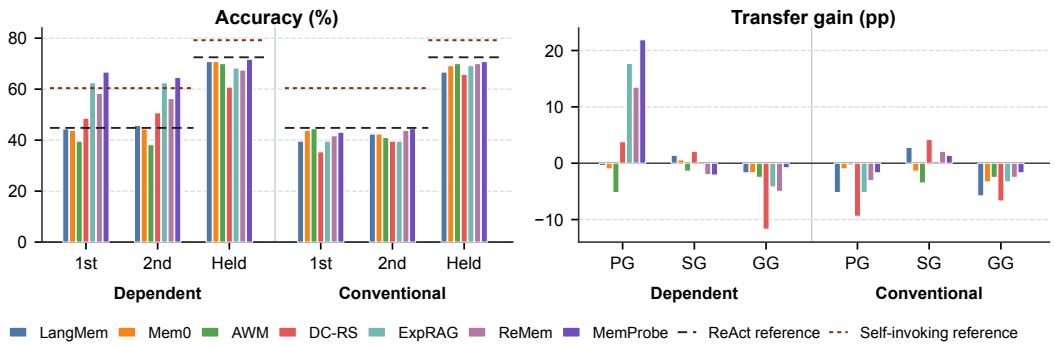

Figure 3: Performance on CodeEval-Pro complex tasks. The BigCodeBench-Lite-Pro subset is used for conventional and dependent task streams. The HumanEval-Pro subset is used as independent tasks after the dependent or conventional stream. ReAct denotes a memoryless reference. For self-invoking, each complex task is provided with its corresponding subtask as a reference context, so no retrieval is required. Metrics are shown in Figure 2. Each value is the average over three runs. *transfer gain* refers collectively to PG, SG, and GG, and *pp* denotes percentage points.

into three complementary views: *interaction* memory for concrete trajectories and final responses, *insight* memory for task patterns and failure modes, and *skill* memory for procedure-level abstractions or short reusable snippets. Invalid outputs are discarded by a syntactic checker, and unreliable interaction histories are filtered by an LLM judge before being written into memory. The prompt formats for constructing and using memory contexts are provided in Appendix B.5. The procedure of MEMPROBE is detailed in Appendix B.6.

5.3 DISCUSSION

Dependent Streams Reveal Specialized Reuse. We show that dependent streams yield substantially clearer separation among memory methods than conventional streams. On CodeEval-Pro in Figure 3 and Table 10, the standard deviation of complex-task accuracy across methods is 9.4 and 8.8 in the two dependent-stream passes, compared with only 3.0 and 1.9 in the conventional stream. The same pattern is even more pronounced on BrowseComp+, where the corresponding dispersions are 14.9 and 16.0 under the dependent stream, but only 2.3 and 5.7 under the conventional stream, as shown in Figure 4 and Table 11. BabyAI is consistent with this trend in Table 3. By contrast, MMLU-Pro, evaluated as a conventional stream in Table 2, shows only modest method separation, with average-accuracy standard deviations of 1.7 and 1.8 across the two passes. These results suggest that designing task relations is essential to distinguish how different CL methods behave when reusable experience is available. This pattern is further supported by pairwise bootstrap, task-distance, and stream-position analyses (Appendices B.7, B.11, and B.12), suggesting that the gains of existing memory designs primarily reflect specialized reuse.

Conventional Streams Are Useful for Stability Tests. The unknown task relation in conventional streams is reflected in the first-pass results: in the conventional CodeEval-Pro stream in Figure 3, PGs are compressed and mostly non-positive across memory methods. MMLU-Pro also shows modest method separation across the two passes in Table 2. Conventional streams provide limited evidence about experience reuse, but they are still informative for stability evaluation: whether accumulated memory remains helpful, or at least harmless, after being updated by a stream with unknown reuse structure. In the conventional CodeEval-Pro stream, Appendix Table 10 shows that SG ranges from -3.5 to $+4.2$, indicating that consolidation helps some methods but hurts others. BrowseComp+ in Appendix Table 11 exposes even larger SG swings in the conventional stream, from -11.0 to $+11.0$. Appendix Table 13 further shows that even grouping ScienceWorld tasks by near-identical goal families produces limited method separation, suggesting that repeated exposure alone is not sufficient to create a diagnostic CL stream.

Independent Tasks Are Necessary for Generalization Testing. Validation on held-out data is a standard practice in machine learning, yet not all the benchmarks for CL include such evaluation. We use 120 HumanEval-Pro tasks as independent tasks after the dependent or conventional stream from BigCodeBench-Lite-Pro. They are still coding tasks but drawn from a different source and are

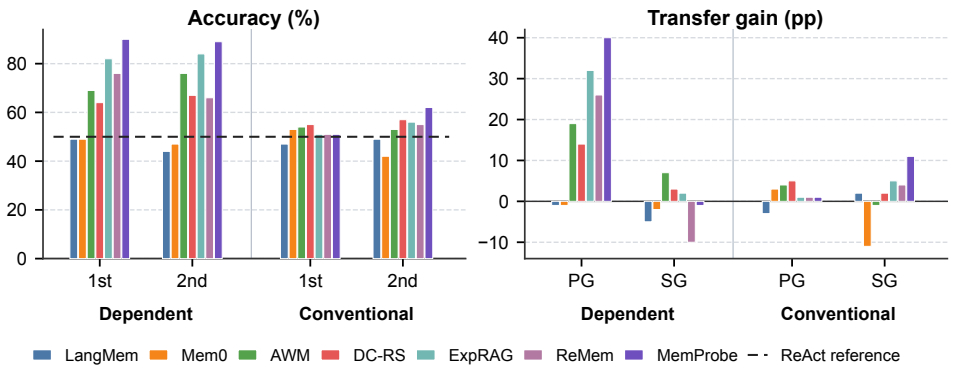

Figure 4: Performance on the selected 100 original BrowseComp+ tasks.

not paired with any previous task. As shown in Appendix Table 10, GGs are limited for all memory methods, and memoryless ReAct remains strongest overall at 72.5. Most memory methods perform slightly below ReAct, suggesting that memory accumulated from weakly related tasks does not automatically transfer and may introduce mild interference. The dependent and conventional streams provide different numbers of learning tasks, but the GGs after these two stream types are similar, indicating that the structural mismatch between learning and testing tasks is the main challenge, rather than the sheer number of preceding tasks. Among memory methods, MEMPROBE is closest to ReAct at 71.7 after the dependent stream and 70.8 after the conventional stream, because it filters out irrelevant memory contexts in most cases. We additionally evaluate generalization on independent AgentBoard BabyAI tasks, with results reported in Appendix Table 19.

Plasticity–Stability Tradeoff. These results expose a plasticity–stability tension in CL: methods that gain substantially when reusable knowledge is present are not necessarily stable across passes or stream types. On CodeEval-Pro (Appendix Table 10), ExpRAG, Evo-Memory's ReMem, and MEMPROBE reach PG of $+17.7$, $+13.5$, and $+21.9$, yet their SG is only $+0.0$, -2.0, and -2.1, and they still trail memoryless ReAct on independent tasks (GG of -4.2, -5.0, and -0.8). DC-RS attains the largest conventional-stream SG ($+4.2$) but suffers the largest GG drop (-11.7), suggesting that memory useful within a stream can hurt under distribution shift. On BrowseComp+ (Appendix Table 11), ReMem gains $+26.0$ dependent-stream PG but loses 10.0 SG, and Mem0 has the worst conventional-stream SG (-11.0) despite a slightly positive PG, whereas MMLU-Pro (Table 2) shows much smaller changes. Thus, CL in language agents is not solved by storing more trajectories or summaries. The central challenge is deciding which past experience to abstract, retrieve, or ignore. This imbalance is further tied to the data distribution: when reusable experience is scarce, an agent may need to actively explore its environment to build a useful memory context (Gao et al., 2026).

Insights on Memory Design. Controlled streams help isolate the effects of memory design choices. Table 4 reports ablations of MEMPROBE on BigCodeBench-Lite-Pro. During the memory construction process, we have the syntactic checker and the LLM judge determine whether the output of an interaction is worth recording (§5.2). Replacing this validation step with the oracle boolean feedback yields similar overall performance, suggesting that the memory construction does not require a perfect judge. Removing interaction, insight, or skill memory weakens performance on dependent streams in most cases. In contrast, these ablations have smaller and less consistent effects on conventional streams. This supports the view that memory representations cannot be fairly diagnosed without task streams that expose opportunities for reuse. The results also suggest that different forms of experience serve complementary roles. Interaction memory can preserve concrete solving episodes, insight memory can distill task-specific lessons and failure modes, and skill memory can capture reusable procedural knowledge.

5.4 CASE STUDY

Table 20 in Appendix B.14 provides two coding cases where retrieved memories help or hurt. In the first case, MEMPROBE retrieves a single-pair version of the same computation and reuses the correct DataFrame construction inside a loop. In the second case, the retrieved task is topically similar but semantically incompatible: it encourages selecting numeric columns, while the target task requires

Table 2: Accuracy (%) on the conventional task stream from MMLU-Pro. *Eco.*: economics. *Eng.*: engineering. *Phil.*: philosophy. Each domain contains 100 samples. Memory is maintained separately by domain. *Std. Dev.* denotes the standard deviation of the average metrics across evaluated methods. The highest value in each column is **bolded**, and the second-highest value is underlined. The same applies to the following tables.

	1st Pass				2nd Pass			
Method	Avg (PG)	Eco.	Eng.	Phil.	Avg (SG)	Eco.	Eng.	Phil.
ReAct	78.7	90.0	66.0	80.0	78.7	90.0	66.0	80.0
ExpRAG	81.3 (+2.6)	**91.0**	73.0	80.0	**83.0** (+1.7)	**91.0**	78.0	80.0
ReMem	79.0 (+0.3)	86.0	74.0	77.0	81.7 (+**2.7**)	87.0	**81.0**	77.0
MEMPROBE	**82.7** (+**4.0**)	**91.0**	**76.0**	**81.0**	**83.0** (+0.3)	**91.0**	76.0	**82.0**
Std. Dev.	1.7	2.1	3.8	1.5	1.8	1.6	5.6	1.8

Table 3: Performance (%) on AgentBoard BabyAI. *SR*: success rate. *PR*: progress rate. The conventional stream in AgentBoard BabyAI exhibits a pattern that tasks appear sequentially by task family, with each family comprising four tasks that share the same goal but feature different environmental settings.

	Dependent Stream				Conventional Stream			
Method	1-SR	1-PR	2-SR	2-PR	1-SR	1-PR	2-SR	2-PR
ReAct (Yao et al., 2023)	50.0	63.6	50.0	63.6	57.5	72.9	57.5	72.9
ExpRAG (Wei et al., 2025b)	37.5	53.1	58.9	73.3	**67.5**	76.7	70.0	78.7
ReMem (Wei et al., 2025b)	51.8	**68.6**	**62.5**	**75.7**	**67.5**	77.9	70.0	79.2
MEMPROBE	**55.4**	65.7	53.6	69.0	**67.5**	**80.0**	**72.5**	**80.4**
Std. Dev.	6.7	5.8	4.8	4.6	4.3	2.6	5.9	2.9

Table 4: Accuracy (%) of MEMPROBE ablations on BigCodeBench-Lite-Pro with two passes. *Oracle feedback* means we replace the syntactic check and the LLM judge during memory consolidation with oracle feedback.

	Dependent		Conventional	
Variant	1st	2nd	1st	2nd
MEMPROBE	**66.7**	**64.6**	43.1	44.5
w/ oracle feedback	64.6	60.4	43.8	43.8
w/o skill	56.3	59.0	43.8	**45.8**
w/o insight	60.4	**64.6**	35.4	41.7
w/o interaction	52.1	50.0	**45.8**	41.7

rejecting any non-numeric column. These cases illustrate that the effect of memory depends both on retrieval and on how the agent uses the retrieved context.

6 CONCLUSION

Existing benchmarks either emphasize long-context reasoning or build task streams without considering cross-task relations, which makes rigorous evaluation for agent CL difficult. We propose an evaluation framework AGENTCL with task streams and metrics, contrasting conventional streams with dependent streams where earlier experience can support later tasks, and testing generalization on independent held-out tasks. Across diverse agentic settings, empirical analysis indicates that: 1) Task-stream design is essential for distinguishing CL capabilities. 2) Balancing plasticity and stability remains a formidable challenge for agent memory, particularly in scenarios where opportunities for cross-task reuse are unclear. These findings suggest that future CL benchmarks should publish task streams more transparently and discuss the relations between tasks, rather than simply streaming existing data and focusing solely on average metrics.

AI USE STATEMENT

In this work, we used generative AI tools to synthesize data: the BrowseComp+ subtasks, generated and verified by GPT-5.2, detailed in Appendix D. AI tools are also used as the evaluated backbone models of the agents and memory methods (§5). Additionally, we used generative AI tools for language polishing and limited coding assistance. We have reviewed all AI-assisted work: synthesized subtasks passed deterministic filtering and a separate verification stage, and AI-assisted code was manually checked and tested by the authors. We did not use AI tools to assist with research ideation or for any other purposes. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work builds an evaluation framework based on publicly released datasets. Each asset and its license is listed in Appendix C. The newly released artifacts, i.e., synthesized BrowseComp+ subtasks together with derived streams, are documented in Appendix D, including the LLM-based generation procedure and the quality control process. Our continual learning method could potentially be exploited to develop harmful agents with progressively stronger capabilities. We believe that mitigating this risk mainly depends on the LLM's ability to reject harmful requests.

REPRODUCIBILITY STATEMENT

The data[1] and code[2] are available for preview. We have taken the following steps to make our results reproducible. Complete reproduction instructions, covering the serving stack, model versions, decoding parameters, and agent interaction budgets, are given in Appendix B.3. The hardware and compute requirements are reported in Appendix B.4. All prompts used for memory construction, memory update, and evaluation are provided verbatim in Appendix B.5. The construction of task streams, including the data sources, filtering criteria, and the two-stage quality-control procedure applied to generated subtasks, is detailed in Appendix B.13 and Appendix D.

REFERENCES

Sandhini Agarwal et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025. doi: 10.48550/arXiv.2508.10925. URL https://arxiv.org/abs/2508.10925.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. MemoryBench: A benchmark for memory and continual learning in LLM systems. *CoRR*, abs/2510.17281, 2025. doi: 10.48550/ARXIV.2510.17281. URL https://doi.org/10.48550/arXiv.2510.17281.

Parth Asawa, Chris Glaze, Gabe Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual Learning Bench. https://continual-learning-bench.com/docs/, 2026. Documentation, accessed 2026-05-04.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, Sahel Sharifymoghaddam, Yanxi Li, Haoran Hong, Xinyu Shi, Xuye Liu, Nandan Thakur, Crystina Zhang, Luyu Gao, Wenhu Chen, and Jimmy Lin. BrowseComp-Plus: A more fair and transparent evaluation benchmark of deep-research agent. *CoRR*, abs/2508.06600, 2025. doi: 10.48550/ARXIV.2508.06600. URL https://doi.org/10.48550/arXiv.2508.06600.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. In Inês Lynce, Nello Murano, Mauro Vallati, Serena Villata, Federico Chesani, Michela Milano, Andrea Omicini, and Mehdi Dastani (eds.), *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025,*

[1] https://dataverse.harvard.edu/previewurl.xhtml?token=46e2cda4-079e-48ee-bc41-7c8a2b722dad

[2] https://anonymous.4open.science/r/Task-Stream-CL-5C9E/

Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025), Frontiers in Artificial Intelligence and Applications, pp. 2993–3000. IOS Press, 2025. doi: 10.3233/FAIA251160. URL https://doi.org/10.3233/FAIA251160.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Qihan Ren, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang, Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Cheng Qian, Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Trans. Mach. Learn. Res.*, 2026, 2026. URL https://openreview.net/forum?id=CTr3bovS5F.

Yuanzhe Hu, Yu Wang, and Julian J. McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. *CoRR*, abs/2507.05257, 2025. doi: 10.48550/ARXIV.2507.05257. URL https://doi.org/10.48550/arXiv.2507.05257.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. SWE-Bench-CL: Continual learning for coding agents. *CoRR*, abs/2507.00014, 2025. doi: 10.48550/ARXIV.2507.00014. URL https://doi.org/10.48550/arXiv.2507.00014.

LangChain AI. LangMem, 2026. URL https://github.com/langchain-ai/langmem. GitHub repository.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn LLM agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/877b40688e330a0e2a3fc24084208dfa-Abstract-Datasets_and_Benchmarks_Track.html.

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 13851–13870. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.747. URL https://doi.org/10.18653/v1/2024.acl-long.747.

OpenAI. Introducing GPT-5.2. https://openai.com/index/introducing-gpt-5-2/, December 2025.

Qwen Team. Qwen3.5-35B-A3B. Hugging Face model card, March 2026. URL https://huggingface.co/Qwen/Qwen3.5-35B-A3B.

Yu Su, Diyi Yang, Shunyu Yao, and Tao Yu. Language agents: Foundations, prospects, and risks. In Junyi Jessy Li and Fei Liu (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - Tutorial Abstracts, Miami, Florida, USA, November 12-16, 2024*, pp. 17–24. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.EMNLP-TUTORIALS.3. URL https://doi.org/10.18653/v1/2024.emnlp-tutorials.3.

Mirac Suzgun, Mert Yüksekgönül, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2026 - Volume 1: Long Papers, Rabat, Morocco, March 24-29, 2026*, pp. 7080–7106. Association for Computational Linguistics, 2026. URL https://aclanthology.org/2026.eacl-long.333/.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(8):5362–5383, 2024a.

doi: 10.1109/TPAMI.2024.3367329. URL https://doi.org/10.1109/TPAMI.2024.3367329.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024b. URL http://papers.nips.cc/paper_files/paper/2024/hash/ad236edc564f3e3156e1b2feafb99a24-Abstract-Datasets_and_Benchmarks_Track.html.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. Browsecomp: A simple yet challenging benchmark for browsing agents. *CoRR*, abs/2504.12516, 2025a. doi: 10.48550/ARXIV.2504.12516. URL https://doi.org/10.48550/arXiv.2504.12516.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-memory: Benchmarking LLM agent test-time learning with self-evolving memory. *CoRR*, abs/2511.20857, 2025b. doi: 10.48550/ARXIV.2511.20857. URL https://doi.org/10.48550/arXiv.2511.20857.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. StreamBench: Towards benchmarking continuous improvement of language agents. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/c189915371c4474fe9789be3728113fc-Abstract-Datasets_and_Benchmarks_Track.html.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=pZiyCaVuti.

Zheng Wu, Xingyu Lou, Xinbei Ma, Yansi Li, Weiwen Liu, Weinan Zhang, Jun Wang, and Zhuosheng Zhang. Agent-Dice: Disentangling knowledge updates via geometric consensus for agent continual learning. *CoRR*, abs/2601.03641, 2026. doi: 10.48550/ARXIV.2601.03641. URL https://doi.org/10.48550/arXiv.2601.03641.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: agentic memory for LLM agents. *CoRR*, abs/2502.12110, 2025. doi: 10.48550/ARXIV.2502.12110. URL https://doi.org/10.48550/arXiv.2502.12110.

Tianci Xue, Zeyi Liao, Tianneng Shi, Zilu Wang, Kai Zhang, Dawn Song, Yu Su, and Huan Sun. Autonomous continual learning of computer-use agents for environment adaptation, 2026. URL https://arxiv.org/abs/2602.10356.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Zhenquan Yao, Zitong Huang, Yihan Zeng, Jianhua Han, Hang Xu, Chun-Mei Feng, Jianwei Ma, and Wangmeng Zuo. CGL: advancing continual GUI learning via reinforcement fine-tuning. *CoRR*, abs/2603.02951, 2026. doi: 10.48550/ARXIV.2603.02951. URL https://doi.org/10.48550/arXiv.2603.02951.

Zhaojian Yu, Yilun Zhao, Arman Cohan, and Xiaoping Zhang. Humaneval pro and MBPP pro: Evaluating large language models on self-invoking code generation task. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, Findings of ACL, pp. 13253–13279. Association for Computational Linguistics, 2025. URL https://aclanthology.org/2025.findings-acl.686/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. doi: 10.48550/arXiv.2506.05176. URL https://arxiv.org/abs/2506.05176.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, Zhong-Zhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating LLM agents as lifelong learners. *CoRR*, abs/2505.11942, 2025. doi: 10.48550/ARXIV.2505.11942. URL https://doi.org/10.48550/arXiv.2505.11942.

Appendix Contents

A Limitations **15**

B Experimental Setting and Details **15**

B.1 Examples of Compositionality 15
B.2 Evaluated Memory Designs 15
B.3 Reproduction Instructions 17
B.4 Compute Resources 18
B.5 Prompts 18
B.6 Details of MEMPROBE 18
B.7 Statistical Significance 21
B.8 Detailed Performance 22
B.9 Performance Across Backbone Models 24
B.10 Sensitivity to Retrieval Budget 24
B.11 Sensitivity to Source–Target Distance 25
B.12 Sensitivity to Stream Position 25
B.13 Details in Benchmark Construction 26
B.14 Case Study 27

C License of Existing Assets **27**

D New Assets **28**

D.1 Synthesis of BrowseComp+ Subtasks 28
D.2 The Statistics of Dependent Streams 29

A LIMITATIONS

This paper focuses primarily on continual learning for language agents through non-parametric memory. This scope is appropriate for isolating how reusable experience can be accumulated and applied across task streams without changing model parameters. However, the current study does not systematically evaluate parametric memory or training-based adaptation methods, which may offer complementary advantages in continual learning. Incorporating such approaches into the evaluation framework would be a valuable direction for future work.

B EXPERIMENTAL SETTING AND DETAILS

B.1 EXAMPLES OF COMPOSITIONALITY

Figure 5 illustrates an example task pair from CodeEval-Pro, where the complex task preserves the core solution logic of the subtask while requiring composition over multiple generated strings. Figure 6 further shows an example from BrowseComp+, where our synthesized subtasks are not simple decompositions that directly reveal the original answer, but instead share intrinsically related solution logic with the original task. Together, these examples demonstrate how our construction exposes reusable problem-solving patterns across related tasks while avoiding direct answer leakage.

B.2 EVALUATED MEMORY DESIGNS

Our evaluation is designed to cover the main non-parametric memory design choices that can support continual learning in language agents. We organize representative methods by what they store, how they update memory, and how the memory is reused during later task solving.

Memoryless reference. We take ReAct (Yao et al., 2023) as a memoryless reference method. The tools available to the agent vary depending on the specific environment. Generally, we employ LLMs equipped with built-in reasoning capabilities, which means these models think before invoking a tool or responding to the user. During each round of interaction with the environment, the preceding reasoning process is preserved. However, this reasoning process cannot be carried over to a subsequent, new task.

Self-invoking reference. We adopt the self-invoking method as a reference for CodeEval-Pro (Yu et al., 2025), as it provides an intuitive solution to self-invoking tasks that we take as a source for our task stream. Note that self-invoking uses inputs distinct from our default experimental setting. It is not treated as a method for direct comparison, but serves as a reference. Each complex task is accompanied by corresponding subtasks as input without any other memory contexts, thereby assessing the LLM's capacity for self-invoking the sub-solution rather than its ability for continual learning. A variation of the self-invoking method is to provide a solution to the subtask alongside its description. Although the input configurations differ, the two self-invoking variants constitute a meaningful comparison because memory methods do not uniformly fall above or below these references.

Adaptive semantic and episodic memory. LangMem (LangChain AI, 2026) and Mem0 (Chhikara et al., 2025) represent adaptive long-term memory systems that decide what information should be extracted, consolidated, and retrieved. This family mainly treats memory as persistent natural-language or structured records that can be searched and injected into future contexts. LangMem provides general primitives for extracting semantic, episodic, and procedural memories from interactions, updating them with an LLM-based memory manager, and retrieving them through an external store. Mem0 similarly builds a scalable memory layer that dynamically extracts salient information from ongoing interactions, compares new candidates with existing memories, and performs memory operations such as adding, updating, deleting, or ignoring entries. These methods cover the design where memory is primarily an adaptive store of facts, events, or prior episodes, and the central question is whether a memory manager can preserve useful information while avoiding redundancy and conflict.

Procedural memory. AWM (Wang et al., 2025) and DC-RS (Suzgun et al., 2026) represent methods that focus on procedural knowledge. AWM induces reusable workflows from successful

SUBTASK

```
import collections
import string
import random

def task_func(length, seed=0):
    """
    Generate a random string of a given length using ASCII letters and calculate
    the frequency of each character.

    Parameters:
        length (int): The length of the random string to be generated.
        seed (int, Optional): The seed to be used for the random number generator.
        Default is 0.

    Returns:
        dict: A dictionary with the frequency of each character in the generated
        string.
    """
```

COMPLEX TASK

```
# You are tasked with generating multiple random strings of varying lengths and
calculating the combined frequency of each character across all generated strings.
The function should accept a list of lengths and a seed for random generation. The
output should be a dictionary with the frequency of each character in the combined
strings.

import collections
import string
import random

def combined_frequency(lengths, seed=0):
```

Figure 5: Example of a CodeEval-Pro task pair. The subtask computes character frequencies for one generated string, while the complex task extends the same logic to aggregate frequencies across multiple random strings of varying lengths.

agent trajectories. A workflow abstracts away instance-specific details and stores a reusable routine that can guide future action generation, making it suitable for long-horizon interactive tasks where similar sub-procedures recur across tasks. DC-RS maintains an evolving cheatsheet of strategies, heuristics, and code snippets. It retrieves similar past input–output pairs and synthesizes them with the current memory to update the cheatsheet before solving the next task. This family covers the design where memory is not merely episodic records, but a compact representation of how to solve recurring classes of problems.

Self-evolving memory. ExpRAG, ReMem (Wei et al., 2025b), and MEMPROBE represent self-evolving memory methods that explicitly operate over a task stream. ExpRAG is a simple experience-retrieval method: it stores previous task experiences, retrieves similar ones for the current task, and uses them as in-context demonstrations. This isolates the effect of experience-level retrieval without adding more complex memory reasoning. ReMem goes further by integrating memory refinement into the agent loop, allowing the agent to reason, act, and refine memory as interacting components rather than treating memory as a passive retrieved context. Our MEMPROBE follows the same broad direction but is designed as a probing method for this benchmark. It separates interaction, insight, and skill memories and applies quality-aware consolidation before writing experiences into memory.

COMPLEX TASK

What was the name of the TV drama, whose title role was played by the actor, who also worked in the debut picture of a DOP? The debut movie of this cinematographer was directed by a man who attended the Wellington College in UK.

Answer: Young Ramsay

SUBTASK 1

According to the Australian Cinematographers Society biography "John Seale AM ACS ASC", what was John Seale's first attempt in the role of DOP (his debut as DOP)?

Answer: Deathcheaters (1976)

SUBTASK 2

In the film "Deathcheaters" (1976), who is credited as the director?

Answer: Brian Trenchard-Smith

SUBTASK 3

In the Wikipedia biography "Brian Trenchard-Smith", what school is listed under his education?

Answer: Wellington College

Figure 6: Example of a BrowseComp+ task and our synthesized subtasks. The subtasks derived from the original task are distinctly different. The answers are provided solely to aid the reader's understanding. During the solving process, the agent has access only to the task statement and the corpus. The synthesis of these subtasks ensures that even if the agent reads all the subtask statements, it cannot directly derive the answer to the original task without research.

Together, these methods cover the design where memory is expected to evolve during deployment and where the key challenge is not only retrieval, but also deciding which experiences should be abstracted, preserved, or ignored over time.

B.3 Reproduction Instructions

We deployed the LLM and embedding model using vLLM v0.16.1.[3] For the Qwen 3.5 model, we used thinking mode and adopted the sampling parameters recommended in the official documentation for precise coding tasks: temperature $= 0.6$, top_p $= 0.95$, top_k $= 20$, min_p $= 0.0$, presence_penalty $= 0.0$, repetition_penalty$= 1.0$.[4] For the OpenAI gpt-oss model, we set reasoning effort to *high* and used default values for the other hyperparameters in the OpenAI client. All deployed LLMs and embedding models are official, non-quantized versions. For hyperparameters, we set the maximum number of tool calls for BrowseComp+ to 100, and the maximum number of action steps for AgentBoard environments to 50. For memory methods, we used a shared top-k retrieval setting with $k = 2$.

[3] https://github.com/vllm-project/vllm

[4] https://huggingface.co/Qwen/Qwen3.5-35B-A3B

Table 5: Average execution statistics on complex tasks under conventional and dependent streams. *Search calls* refer to the number of times the document corpus is retrieved, rather than the number of memory retrievals. Search calls are included in tool calls.

Benchmark	Stream	Time (s)	Tool Calls	Search Calls	LLM Calls
BigCodeBench-Lite-Pro	Conventional	26.2	1.9	–	2.9
	Dependent	24.4	1.7	–	2.7
BrowseComp+	Conventional	109.7	49.9	31.4	52.8
	Dependent	46.2	30.7	16.2	32.8

Table 6: Token usage statistics on the BigCodeBench-Lite-Pro and BrowseComp+ dependent streams (96 tasks and 408 tasks, respectively; subtasks included). Both datasets are evaluated over two passes. All values are reported in millions of tokens. The tokenizers are derived from the corresponding LLMs.

Method	BigCodeBench-Lite-Pro			BrowseComp+		
	Total	Input	Output	Total	Input	Output
LangMem	1.8	1.4	0.4	542.5	539.1	3.4
Mem0	2.5	2.1	0.4	543.6	540.2	3.4
AWM	3.9	3.4	0.5	375.9	373.3	2.7
DC-RS	6.2	5.7	0.5	412.7	409.8	2.9
ExpRAG	1.9	1.6	0.3	256.7	254.5	2.2
ReMem	1.6	1.2	0.3	357.6	355.0	2.7
MEMPROBE	2.5	2.1	0.4	220.9	218.9	2.0

B.4 COMPUTE RESOURCES

We use one NVIDIA A100 to deploy Qwen3-Embedding-8B, and at least two NVIDIA A100s to deploy Qwen3.5-35B-A3B or openai/gpt-oss-120b. The required time depends on the number of deployed GPUs, the size of the dataset, and the number of inference steps. We report the time costs of MEMPROBE in Table 5. In terms of runtime and the number of tool calls, reusing experience reduces the cost of solving complex tasks. We also report the token usage of evaluated methods on dependent streams in Table 6.

B.5 PROMPTS

Figure 7 and Figure 8 illustrate the prompting and rendering design used in MEMPROBE under the coding setting. After each task, the memory update prompt in Figure 7 takes the task, the candidate solution, and the agent's generated code as input, and asks an LLM to extract the episode into structured insight and skill memories, while the trajectory, i.e., generated code in this task, is retained as interaction memory. At inference time, as shown in Figure 8, retrieved memories from prior tasks are organized into a unified memory context that includes the task description, interaction record, extracted insights, and reusable skills, which are then provided to the agent to support solving the current task.

B.6 DETAILS OF MEMPROBE

To instantiate AGENTCL, we introduce a non-parametric memory probing method MEMPROBE for analyzing how agents accumulate and consolidate experience across a task stream. It aims for a balance between plasticity and stability. For plasticity, memory should capture complementary forms of experience, such as raw trajectories, concise summaries, and reusable skills, rather than biasing the agent toward a single representation that may only work in specific domains. For stability, stored experience should improve future decisions more often than it harms them, since past solutions can also bias the agent toward incorrect reasoning paths. Thus, we introduce MEMPROBE based on a retrieve–solve–consolidate loop.

Memory Structure. Rather than storing only raw trajectories or only compact summaries, MEMPROBE organizes experience into three complementary memory types. 1) **Interaction memory**: This

SYSTEM

```
You are a memory agent. Given a programming task and a candidate solution, create
a concise memory update. Output strict JSON with keys: insights, skills.
- The 'Task' is the only thing being solved/evaluated.
- Insights must be an object with keys: correctness, summary, complexity,
vulnerabilities, reusable.
  - correctness must be exactly one of: "true", "false", "unknown", consider all
  the specifications from the task description.
  - summary/complexity/vulnerabilities: string
  - reusable is a list of 1-4 short strings that are helpful even without the
  specific task background
- Skills must be reusable procedures: a list of 0-3 objects with keys: name,
snippet, description. Keep snippets short and self-contained.
```

USER

```
Task:
{TASK_QUERY}

Candidate solution:
{CANDIDATE_SOLUTION}

Agent trajectory (tool call/response trace):
{AGENT_TRAJECTORY}

Return the JSON now.
```

Figure 7: Memory update prompt in the coding setting. The interactive trajectory is stored as interaction memory, while an LLM extracts the current task episode into insight and skill memories.

memory records the concrete solving episode, including the agent's interaction trajectory and the final response. It is useful when future tasks benefit from recalling how a similar problem was operationally solved. 2) **Insight memory**: This memory stores extracted information from the episode, including a concise summary of the task-solving pattern, complexity-related observations, potential failure modes, and reusable takeaways. Compared with interaction memory, it is more abstract and compact, making it suitable for transferring task-relevant lessons without replaying the detailed trajectory. 3) **Skill memory:** This memory stores reusable procedures or workflow-level knowledge in a more explicit form, optionally accompanied by short code snippets. Skill memory is intended to capture higher-level strategies that may generalize across multiple tasks, rather than being tied to one specific instance. This decomposition reflects our view that reusable experience in continual learning is heterogeneous. Comprehensive representation allows the method to represent cross-task experience more flexibly than a single-memory format.

Memory Retrieval and Reuse. Given a new task τ_t, MEMPROBE first retrieves a small set of relevant past experiences from memory using embedding-based similarity search over previously stored task descriptions. Let M_{t-1} denote the current memory state and let $\mathcal{R}$ denote retrieval. The retrieved context is

$$c_t = \mathcal{R}(M_{t-1}, \tau_t),$$

where retrieval returns the top-k most similar prior tasks above a similarity threshold, together with the interaction, insight, and skill entries associated with those tasks. In our implementation, retrieval operates at the task level: the current task is matched against previously stored task descriptions via semantic similarity, and the associated interaction, insight, and skill entries are then rendered into the memory context. The agent then solves τ_t conditioned on the retrieved context. Importantly, retrieved memory is treated as reference context rather than authoritative guidance, so the model is encouraged to ground its response in the current task instead of reproducing superficially similar past solutions

MEMORY CONTEXT

```
### Retrieved memory (score: {SIMILARITY_SCORE})

#### Description
```text
{QUERY_OR_RETRIEVAL_QUERY}
```

Interaction trajectory 1
- Final code:
```python
{FINAL_CODE}
```

Insights 1
- Summary: {SUMMARY}
- Complexity: {COMPLEXITY}
- Vulnerabilities: {VULNERABILITIES}
- Reusable:
  - {REUSABLE_ITEM_1}
  - {REUSABLE_ITEM_2}

Skill 1
- Name: {SKILL_NAME}
- Description: {SKILL_DESCRIPTION}
- Snippet:
```python
{SKILL_SNIPPET}
```
```

Figure 8: Retrieved memory context in the coding setting. For each retrieved task, the solver is provided with the task description, interaction memory, extracted insights, and reusable skills.

or trajectories. A small retrieval budget bounds reuse by exposing only top-k semantically related entries, so that interference from weakly related tasks is reduced.

Quality-Aware Memory Consolidation. After completing task τ_t, the method consolidates the current episode into a new memory record:

$$M_t = \mathcal{U}(M_{t-1}, \tau_t, \xi_t).$$

During consolidation, the episode is converted into the three memory types described above. The updated memory is immediately available to subsequent tasks, enabling online accumulation of experience over the task stream. A central challenge in continual memory is that solved episodes vary in future utility. Poor solutions, malformed outputs, or misleading trajectories may be retrieved later and degrade decision-making. To mitigate this risk, MEMPROBE incorporates quality-aware controls during consolidation. First, consolidation is gated by basic solution validity. In the coding setting, the system skips consolidation if the extracted solution is not syntactically valid, preventing obviously unusable outputs from entering long-term memory. Second, the consolidation model is prompted to judge whether the candidate solution is correct, incorrect, or uncertain for the current task. This signal is stored in the insight memory and used conservatively during future reuse. In particular, memories judged clearly incorrect are not allowed to contribute full interaction-level traces as reusable context. This preserves compact evidence that a past attempt failed, while reducing the risk of replaying erroneous reasoning in later tasks. Overall, MEMPROBE provides a conservative continual learning mechanism that steadily accumulates structured experience from multi-turn agent episodes, while carefully restricting how experience is consolidated and reused so that memory is more likely to aid transfer than amplify noise.

Algorithm 1 MEMPROBE: Online Memory Update and Reuse

Require: Task stream $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_T)$; retrieval function $\mathcal{R}$; agent policy π; validation function $\mathcal{V}$; extraction function $\mathcal{D}$; memory update function $\mathcal{U}$; retrieval budget k
Ensure: Final memory M_T and task outputs $\{a_t\}_{t=1}^{T}$
1: Initialize memory $M_0 \leftarrow \emptyset$
2: **for** $t = 1, \ldots, T$ **do**
3: Retrieve relevant prior experience:

$$c_t \leftarrow \mathcal{R}(M_{t-1}, \tau_t, k)$$

4: Solve the current task:

$$(a_t, \xi_t) \leftarrow \pi(\tau_t, c_t)$$

5: Validate the episode, where the quality assessment $q_t \in \{\text{correct}, \text{unknown}, \text{incorrect}\}$:

$$q_t \leftarrow \mathcal{V}(\tau_t, a_t, \xi_t)$$

6: Extract structured memories for interaction, insights, and skills:

$$(m_t^{\text{int}}, m_t^{\text{ins}}, m_t^{\text{skill}}) \leftarrow \mathcal{D}(\tau_t, a_t, \xi_t, q_t)$$

7: Update persistent memory:

$$M_t \leftarrow \mathcal{U}(M_{t-1}, m_t^{\text{int}}, m_t^{\text{ins}}, m_t^{\text{skill}})$$

8: **end for**
9: **return** M_T and $\{a_t\}_{t=1}^{T}$

Table 7: Performance on CodeEval-Pro (BigCodeBench-Lite-Pro and HumanEval-Pro) with 95% bootstrap CIs. Each value is the average accuracy over three runs, with CIs shown as superscript and subscript.

	Dependent			Conventional		
Pass	1st	2nd	Indep.	1st	2nd	Indep.
Self-invoking	$60.4^{+8.3}_{-7.6}$	$60.4^{+8.3}_{-7.6}$	–	–	–	$79.2^{+4.2}_{-4.2}$
w/ subtask solution	$59.0^{+8.3}_{-8.3}$	$59.0^{+8.3}_{-8.3}$	–	–	–	$73.3^{+4.5}_{-4.7}$
ReAct	$44.8^{+8.3}_{-7.6}$	$44.8^{+8.3}_{-7.6}$	$\mathbf{72.5}^{+4.7}_{-4.7}$	$\mathbf{44.8}^{+8.3}_{-7.6}$	$\mathbf{44.8}^{+8.3}_{-7.6}$	$\mathbf{72.5}^{+4.7}_{-4.7}$
LangMem	$44.4^{+8.3}_{-8.3}$	$45.8^{+8.3}_{-8.3}$	$70.8^{+4.7}_{-4.7}$	$39.6^{+8.3}_{-7.6}$	$42.4^{+8.3}_{-7.6}$	$66.7^{+4.7}_{-5.0}$
Mem0	$43.8^{+8.3}_{-8.3}$	$44.4^{+8.3}_{-8.3}$	$70.8^{+4.7}_{-4.7}$	$43.8^{+8.3}_{-7.7}$	$42.4^{+8.3}_{-7.6}$	$69.2^{+4.7}_{-4.7}$
AWM	$39.6^{+7.6}_{-7.6}$	$38.2^{+8.3}_{-7.6}$	$70.0^{+4.7}_{-4.7}$	$44.5^{+8.3}_{-8.3}$	$41.0^{+8.3}_{-7.6}$	$70.0^{+4.7}_{-4.7}$
DC-RS	$48.6^{+7.6}_{-8.3}$	$50.7^{+8.3}_{-8.3}$	$60.8^{+5.0}_{-5.0}$	$35.4^{+7.6}_{-7.6}$	$39.6^{+7.6}_{-7.6}$	$65.8^{+5.0}_{-5.0}$
ExpRAG	$\underline{62.5}^{+7.6}_{-7.6}$	$\underline{62.5}^{+7.6}_{-7.6}$	$68.3^{+4.7}_{-4.7}$	$39.6^{+7.6}_{-7.6}$	$39.6^{+7.8}_{-7.8}$	$69.2^{+4.7}_{-4.7}$
ReMem	$58.3^{+7.7}_{-8.3}$	$56.3^{+8.3}_{-7.6}$	$67.5^{+5.0}_{-5.0}$	$41.7^{+8.3}_{-7.7}$	$43.8^{+7.7}_{-8.3}$	$70.0^{+4.7}_{-4.7}$
MEMPROBE	$\mathbf{66.7}^{+7.6}_{-7.6}$	$\mathbf{64.6}^{+7.6}_{-8.3}$	$\underline{71.7}^{+4.7}_{-4.7}$	$\underline{43.1}^{+7.6}_{-7.6}$	$\underline{44.5}^{+8.3}_{-7.6}$	$\underline{70.8}^{+4.7}_{-4.7}$

Algorithm. Algorithm 1 summarizes the online memory update and reuse process of MEMPROBE over a task stream $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_T)$, where τ_t denotes the t-th task, M_t denotes the memory state after finishing τ_t, and ξ_t denotes the interaction trajectory collected while solving τ_t.

B.7 STATISTICAL SIGNIFICANCE

We present mean values with 95% bootstrap confidence intervals (CIs) for the main results in Tables 7 and 8.

To further quantify whether different stream designs yield clearer separation among methods, we additionally perform pairwise comparisons on CodeEval-Pro. For each stream setting, we compute the pairwise accuracy difference between evaluated methods under bootstrap resampling and report

Table 8: Performance on BrowseComp+ (308 subtasks and 100 tasks). *Sub* denotes subtasks, and *Q* denotes complex tasks.

	Dependent				Conventional	
Method	1st-Sub	2nd-Sub	1st-Q	2nd-Q	1st-Q	2nd-Q
ReAct	$\underline{97.4}^{+1.6}_{-1.9}$	$97.4^{+1.6}_{-1.9}$	$50.0^{+10.0}_{-10.0}$	$50.0^{+10.0}_{-10.0}$	$50.0^{+10.0}_{-10.0}$	$50.0^{+10.0}_{-10.0}$
LangMem	$\mathbf{97.7}^{+1.6}_{-1.6}$	$\mathbf{98.4}^{+1.3}_{-1.6}$	$49.0^{+10.0}_{-10.0}$	$44.0^{+10.0}_{-9.0}$	$47.0^{+10.0}_{-10.0}$	$49.0^{+10.0}_{-10.0}$
Mem0	$\mathbf{97.7}^{+1.6}_{-1.9}$	$96.8^{+1.9}_{-1.9}$	$49.0^{+10.0}_{-10.0}$	$47.0^{+10.0}_{-10.0}$	$53.0^{+10.0}_{-10.0}$	$42.0^{+10.0}_{-9.0}$
AWM	$96.8^{+1.9}_{-2.3}$	$97.4^{+1.6}_{-1.9}$	$69.0^{+9.0}_{-9.0}$	$76.0^{+8.0}_{-9.0}$	$\underline{54.0}^{+10.0}_{-10.0}$	$53.0^{+10.0}_{-10.0}$
DC-RS	$96.4^{+1.9}_{-2.3}$	$\underline{97.7}^{+1.6}_{-1.6}$	$64.0^{+9.0}_{-9.0}$	$67.0^{+9.0}_{-10.0}$	$\mathbf{55.0}^{+10.0}_{-10.0}$	$\underline{57.0}^{+10.0}_{-10.0}$
ExpRAG	$\underline{97.4}^{+1.6}_{-1.9}$	$\mathbf{98.4}^{+1.3}_{-1.6}$	$\underline{82.0}^{+7.0}_{-8.0}$	$\underline{84.0}^{+7.0}_{-7.0}$	$51.0^{+10.0}_{-10.0}$	$56.0^{+10.0}_{-10.0}$
ReMem	$97.1^{+1.6}_{-1.9}$	$96.8^{+1.9}_{-2.3}$	$76.0^{+8.0}_{-9.0}$	$66.0^{+9.0}_{-9.0}$	$51.0^{+10.0}_{-10.0}$	$55.0^{+10.0}_{-10.0}$
MEMPROBE	$96.8^{+1.9}_{-1.9}$	$97.4^{+1.6}_{-1.9}$	$\mathbf{90.0}^{+5.0}_{-6.0}$	$\mathbf{89.0}^{+6.0}_{-6.0}$	$51.0^{+10.0}_{-10.0}$	$\mathbf{62.0}^{+9.0}_{-10.0}$

Table 9: Statistical significance of pairwise method comparisons on CodeEval-Pro. The first row denotes the proportion of pairwise comparisons whose 95% CI excludes zero. Higher values indicate that the stream yields clearer pairwise separation among evaluated methods.

	Dependent			Conventional		
Statistics	1st Pass	2nd Pass	Indep.	1st Pass	2nd Pass	Indep.
Pairwise CI excludes zero (%)	34.1	31.9	28.6	7.1	6.0	17.8
Mean absolute delta (pp)	11.2	10.3	4.0	4.8	3.7	4.2

two descriptive statistics: the proportion of pairwise comparisons whose 95% CI excludes zero, and the mean absolute pairwise delta in percentage points. As shown in Table 9, dependent streams yield substantially stronger pairwise separation than conventional streams. The same trend appears in the mean absolute delta. The independent setting remains comparatively weakly separated, consistent with its role as a robustness check rather than a setting with explicit transfer opportunities. These results provide complementary evidence that dependent streams make differences among memory methods more statistically visible than conventional streams. For CodeEval-Pro, CIs are computed using bootstrapping across all three-run results. For BrowseComp+, CIs are computed from task-level bootstrap results across 100 tasks.

B.8 DETAILED PERFORMANCE

We report the detailed results on CodeEval-Pro in Table 10, the accuracy on BrowseComp+ complex tasks in Table 11, and the accuracy on BrowseComp+ subtasks in Table 12.

AgentBoard ScienceWorld contains task families in which tasks share the same overall goal but differ in environment configurations. In addition to the original conventional stream, we evaluate a block stream that groups tasks from the same family together. This setting tests repeated exposure to near-duplicate goals rather than compositional reuse. As shown in Table 13, the block stream produces limited method separation, similar to the conventional stream. This contrasts with the larger dispersion observed in dependent coding and deep-research streams.

Since most evaluated memory methods distill memories from one or multiple tasks, and the resulting memory states may be updated over time, directly evaluating the retrieval step is less straightforward. Therefore, we primarily report the retrieval performance of MEMPROBE on the dependent streams of our two main datasets, as shown in Table 14. We measure Hits@k, defined as the proportion of the top-2 retrieved memory entries that correspond to relevant and reusable experience. The results show that retrieving the memory of the corresponding previous task is not a major bottleneck, suggesting that the more critical challenge lies in how memories are represented and utilized. We also observe that Complex $\rightarrow$ Sub of the second pass is lower than that of the first pass on both datasets. This is expected because Complex $\rightarrow$ Self memories in the second pass provide more directly relevant experience and thus compete with subtask memories during retrieval. Importantly, a high hit rate of Complex $\rightarrow$ Self does not constitute answer leakage: memories must be constructed by the agent itself in our setting, and the previous attempt is only available as memory context, serving as a reference rather than a guaranteed correct solution.

Table 10: Accuracy (%) on CodeEval-Pro complex tasks (BigCodeBench-Lite-Pro and HumanEval-Pro). BigCodeBench-Lite-Pro is used for conventional and dependent task streams. HumanEval-Pro is used as independent tasks after the dependent or conventional stream. In the self-invoking setting, each complex task is provided with its corresponding subtask as reference context, so no retrieval is required. PG denotes the difference from the memoryless agent on the corresponding stream, and SG denotes the difference between the 2nd pass and the 1st pass. Each value is the average over three runs. Bold and underline mark the highest and second-highest values for each metric, respectively. *Std. Dev.* denotes the population standard deviation across available non-self-invoking methods for each column.

	Dependent Stream			Conventional Stream		
Method	1st (PG)	2nd (SG)	Indep.	1st (PG)	2nd (SG)	Indep.
Self-invoking	60.4	60.4	–	–	–	79.2
w/ subtask solution	59.0	59.0	–	–	–	73.3
ReAct	44.8	44.8	**72.5**	**44.8**	**44.8**	**72.5**
LangMem	44.4 (−0.4)	45.8 (+1.4)	70.8 (−1.7)	39.6 (−5.2)	42.4 (+2.8)	66.7 (−5.8)
Mem0	43.8 (−1.0)	44.4 (+0.6)	70.8 (−1.7)	43.8 (−1.0)	42.4 (−1.4)	69.2 (−3.3)
AWM	39.6 (−5.2)	38.2 (−1.4)	70.0 (−2.5)	44.5 (**−0.3**)	41.0 (−3.5)	70.0 (−2.5)
DC-RS	48.6 (+3.8)	50.7 (**+2.1**)	60.8 (−11.7)	35.4 (−9.4)	39.6 (**+4.2**)	65.8 (−6.7)
ExpRAG	62.5 (+17.7)	62.5 (+0.0)	68.3 (−4.2)	39.6 (−5.2)	39.6 (+0.0)	69.2 (−3.3)
ReMem	58.3 (+13.5)	56.3 (−2.0)	67.5 (−5.0)	41.7 (−3.1)	43.8 (+2.1)	70.0 (−2.5)
MEMPROBE	**66.7** (**+21.9**)	**64.6** (−2.1)	71.7 (**−0.8**)	43.1 (−1.7)	44.5 (+1.4)	70.8 (**−1.7**)
Std. Dev.	9.4	8.8	3.5	3.0	1.9	2.0

Table 11: Accuracy (%) of 100 tasks sampled from the original BrowseComp+ dataset, evaluated with two passes.

	Dependent Stream		Conventional Stream	
Method	1st-Q (PG)	2nd-Q (SG)	1st-Q (PG)	2nd-Q (SG)
ReAct	50.0	50.0	50.0	50.0
LangMem	49.0 (−1.0)	44.0 (−5.0)	47.0 (−3.0)	49.0 (+2.0)
Mem0	49.0 (−1.0)	47.0 (−2.0)	53.0 (+3.0)	42.0 (−11.0)
AWM	69.0 (+19.0)	76.0 (**+7.0**)	54.0 (+4.0)	53.0 (−1.0)
DC-RS	64.0 (+14.0)	67.0 (+3.0)	**55.0** (**+5.0**)	57.0 (+2.0)
ExpRAG	82.0 (+32.0)	84.0 (+2.0)	51.0 (+1.0)	56.0 (+5.0)
ReMem	76.0 (+26.0)	66.0 (−10.0)	51.0 (+1.0)	55.0 (+4.0)
MEMPROBE	**90.0** (**+40.0**)	**89.0** (−1.0)	51.0 (+1.0)	**62.0** (**+11.0**)
Std. Dev.	14.9	16.0	2.3	5.7

Table 12: Accuracy (%) of 308 subtasks synthesized on BrowseComp+, evaluated with two passes.

	Dependent	
Method	1st-Sub (PG)	2nd-Sub (SG)
ReAct	97.4	97.4
LangMem	**97.7** (**+0.3**)	**98.4** (+0.7)
Mem0	**97.7** (**+0.3**)	96.8 (−0.9)
AWM	96.8 (−0.6)	97.4 (+0.6)
DC-RS	96.4 (−1.0)	97.7 (**+1.3**)
ExpRAG	97.4 (+0.0)	**98.4** (+1.0)
ReMem	97.1 (−0.3)	96.8 (−0.3)
MEMPROBE	96.8 (−0.6)	97.4 (+0.6)
Std. Dev.	0.4	0.6

Table 13: Performance (%) on AgentBoard ScienceWorld. *SR*: success rate. *PR*: progress rate.

	Conventional Stream				Block Stream			
Method	1-SR	1-PR	2-SR	2-PR	1-SR	1-PR	2-SR	2-PR
ReAct	64.4	87.1	64.4	87.1	**64.4**	87.1	64.4	87.1
ExpRAG	58.9	86.0	**67.8**	87.3	**64.4**	**87.8**	**65.6**	86.9
ReMem	59.3	83.3	62.9	87.1	**64.4**	85.7	62.2	85.0
MEMPROBE	**66.7**	**88.6**	66.7	**88.1**	60.0	85.2	62.2	**87.2**
Std. Dev.	3.3	1.9	1.9	0.4	1.9	1.0	1.5	0.9

Table 14: Hits@k retrieval performance on dependent streams. *Comp.* → *Sub* denotes retrieving the corresponding subtask memory given a complex task query, while *Comp.* → *Self* denotes retrieving the memory of the same complex task from the previous pass.

Dataset	**Pass 1 Comp. → Sub**	**Pass 2 Comp. → Sub**	**Pass 2 Comp. → Self**
BigCodeBench-Lite-Pro	95.8	91.7	98.8
BrowseComp+	87.0	77.0	98.0

Table 15: Accuracy (%) on CodeEval-Pro complex tasks using three backbone models. Each stream is evaluated with two passes; PG and SG are shown in parentheses. The Qwen3.5-35B-A3B results reproduce the corresponding three-run averages in Table 10.

	Dependent Stream		Conventional Stream	
Method	1st (PG)	2nd (SG)	1st (PG)	2nd (SG)
Qwen3.5-35B-A3B				
ReAct	44.8	44.8	44.8	44.8
ExpRAG	62.5 (+17.7)	62.5 (+0.0)	39.6 (−5.2)	39.6 (+0.0)
ReMem	58.3 (+13.5)	56.3 (−2.0)	41.7 (−3.1)	43.8 (+2.1)
MEMPROBE	66.7 (+21.9)	64.6 (−2.1)	43.1 (−1.7)	44.5 (+1.4)
gpt-oss-120b				
ReAct	45.8	45.8	45.8	45.8
ExpRAG	58.3 (+12.5)	58.3 (+0.0)	37.5 (−8.3)	35.4 (−2.1)
ReMem	56.3 (+10.4)	52.1 (−4.2)	43.8 (−2.1)	43.8 (+0.0)
MEMPROBE	52.1 (+6.3)	56.3 (+4.2)	35.4 (−10.4)	33.3 (−2.1)
Claude Sonnet 5				
ReAct	50.0	50.0	50.0	50.0
ExpRAG	62.5 (+12.5)	64.6 (+2.1)	47.9 (−2.1)	47.9 (+0.0)
ReMem	68.8 (+18.8)	64.6 (−4.2)	47.9 (−2.1)	45.8 (−2.1)
MEMPROBE	58.3 (+8.3)	56.3 (−2.0)	50.0 (+0.0)	50.0 (+0.0)

B.9 PERFORMANCE ACROSS BACKBONE MODELS

We replicate experiments on CodeEval-Pro using open-weight gpt-oss-120b and proprietary Claude Sonnet 5, while keeping all other settings fixed. Table 15 compares these results with the Qwen3.5-35B-A3B backbone presented in the main text. Despite different backbone models, our main conclusions remain.

B.10 SENSITIVITY TO RETRIEVAL BUDGET

We additionally evaluate MEMPROBE with $k \in \{1, 2, 4\}$, where k is the number of retrieved chunks during inference, keeping all other settings fixed. As shown in Table 16, the contrast between dependent and conventional streams remains regardless of the choices of k. Increasing k to 4 does not monotonically improve performance and slightly worsens conventional-stream plasticity, suggesting that retrieving more potentially irrelevant experience can introduce interference.

Table 16: Effect of retrieval depth k on MEMPROBE for CodeEval-Pro complex tasks. Accuracy (%) is reported with PG and SG in parentheses. 1st and 2nd denote the first and second passes over the same stream.

	Dependent Stream		Conventional Stream	
k	1st (PG)	2nd (SG)	1st (PG)	2nd (SG)
1	66.7 (+21.9)	60.4 (−6.3)	45.8 (+1.0)	45.8 (+0.0)
2	66.7 (+21.9)	64.6 (−2.1)	43.1 (−1.7)	44.5 (+1.4)
4	64.6 (+19.8)	64.6 (+0.0)	41.7 (−3.1)	43.8 (+2.1)

Table 17: Accuracy (%), PG (in parentheses), and task retrieval Hits@2 (%) on the CodeEval-Pro dependent stream (first pass), grouped by task distance (the number of intervening tasks between a subtask and its dependent complex task). PG is computed against ReAct on the same tasks within each interval.

Task Distance	7–26	27–46	47–66	67–86
# Task pairs	9	16	12	11
Mean distance	19.2	35.9	56.3	75.7
		Accuracy (PG)		
ReAct	33.3	68.8	33.3	36.4
ExpRAG	44.4 (+11.1)	81.2 (+12.5)	58.3 (+25.0)	54.5 (+18.2)
ReMem	44.4 (+11.1)	75.0 (+6.2)	50.0 (+16.7)	54.5 (+18.2)
MEMPROBE	44.4 (+11.1)	81.2 (+12.5)	58.3 (+25.0)	54.5 (+18.2)
		Hits@2		
ExpRAG	100.0	100.0	91.7	100.0
ReMem	88.9	93.8	100.0	90.9
MEMPROBE	100.0	93.8	91.7	100.0

B.11 SENSITIVITY TO SOURCE–TARGET DISTANCE

We examine whether transfer degrades as the distance between a subtask and its dependent complex task grows, i.e., whether there is a forgetting window for the evaluated memory designs. The dependent streams already contain substantial natural variation in this source-target task distance. On CodeEval-Pro, the 48 subtasks and the 48 complex tasks are shuffled independently before the two blocks are concatenated, so the number of intervening tasks between a subtask and its dependent task ranges from 7 to 86, with a mean of 47. We partition this range into four equal-width intervals and report accuracy, PG, and task retrieval Hits@2 within each interval. PG is measured against the memoryless ReAct result on the same tasks, which partly controls for differences in task difficulty across intervals.

As shown in Table 17, performance does not degrade monotonically with distance. In the longest-distance interval, where the source and target tasks are separated by 75.7 tasks on average, all three memory designs retain a PG of +18.2, and the relevant subtask is still retrieved in 90.9%–100% of cases. We therefore find no evidence of an effective forgetting window at the stream lengths, at least within the range observed in our experiments.

B.12 SENSITIVITY TO STREAM POSITION

We examine how PG evolves as a stream proceeds, i.e., whether memory methods improve at later positions as more experience is accumulated. To this end, we partition each stream into consecutive position bins without resetting memory, and report PG within each bin, together with its average across methods.

Table 18 shows that the trend depends on the stream type. On conventional streams, PG does not increase as more tasks are processed, and often declines. Averaged across methods, PG drops from +8.5 in the first bin to +0.5 in the last on MMLU-Pro, from −1.1 to −3.3 on ScienceWorld, and ends at −8.3 on CodeEval-Pro. Accumulating weakly related experience therefore does not translate into improved plasticity. In contrast, on the CodeEval-Pro dependent stream, PG is positive in every bin and increases toward later positions, with the method average rising from +8.3 in the first bin

Table 18: PG (%) by task position, with memory accumulated continuously across each stream. Bins are consecutive and of equal size: CodeEval-Pro 4×12 complex tasks, MMLU-Pro 4×25 tasks per domain, and ScienceWorld 3×30 tasks. The two CodeEval-Pro streams share the same complex tasks in the same order and bins; in the dependent stream, all 48 subtasks precede the complex tasks. *Average* denotes the mean over the three methods. PG is computed against ReAct on the same tasks within each bin. Results are from a single run; Table 10 reports three-run averages.

Method	Bin 1	Bin 2	Bin 3	Bin 4
CodeEval-Pro, conventional stream				
ExpRAG	−8.3	−8.3	+8.3	−16.7
ReMem	+0.0	−8.3	+0.0	−8.3
MEMPROBE	+0.0	−16.7	+0.0	+0.0
Average	−2.8	−11.1	+2.8	−8.3
CodeEval-Pro, dependent stream				
ExpRAG	+8.3	+8.3	+25.0	+25.0
ReMem	+8.3	+0.0	+25.0	+16.7
MEMPROBE	+8.3	+25.0	+8.3	+25.0
Average	+8.3	+11.1	+19.4	+22.2
MMLU-Pro, conventional stream				
ExpRAG	+10.7	−2.7	+2.7	+0.0
ReMem	+6.7	−4.0	+0.0	−1.3
MEMPROBE	+8.0	+4.0	+1.3	+2.7
Average	+8.5	−0.9	+1.3	+0.5
ScienceWorld, conventional stream				
ExpRAG	−3.3	−6.7	−6.7	–
ReMem	−3.3	−6.7	−3.3	–
MEMPROBE	+3.3	+3.3	+0.0	–
Average	−1.1	−3.4	−3.3	–

to +22.2 in the last. Since both CodeEval-Pro streams evaluate the same complex tasks in the same order and bins, this contrast is not due to task difficulty. Together with the distance analysis in Appendix B.11, these results suggest that whether memory benefits from a longer stream depends on the presence of reusable task relations.

B.13 DETAILS IN BENCHMARK CONSTRUCTION

We provide details on the construction of the AgentBoard BabyAI dataset. We constructed two task streams from this dataset. It originally had 112 tasks, organized into 28 BabyAI families, with four tasks per family. Tasks in each family share the same overall goal but differ in their environments. For the dependent stream, we selected seven high-confidence source-to-target task-family relations. Each relation maps a simpler source family to a target family that requires an additional or composed skill, such as single-object navigation to sequential navigation, navigation to pickup, single-door opening to ordered two-door opening, or pickup to unlock-then-pickup. For each selected pair, we kept all four source tasks and all four target tasks. The 28 source tasks and 28 target tasks were shuffled independently and randomly, then concatenated as a source half followed by a target half. This yields a deterministic 56-task bipartite stream designed to test whether earlier experience with simpler task families can transfer to later, compositionally related target families. Given that a special property of AgentBoard BabyAI is that its task families exhibit clear relations, which have already been exploited in constructing the dependent stream, we aim to exclude the influence of such relations when constructing the conventional stream. We selected ten easy-only BabyAI families covering diverse atomic skills, including navigation, target disambiguation, door opening, pickup, obstacle-aware pickup, location-grounded pickup, unlocking, and unlock-then-pickup. Each selected family was required to contain exactly 4 tasks. Families were ordered according to their first appearance in the original split, and within-family order was preserved, which yields a 40-task stream.

Table 19: Performance (%) on 36 held-out AgentBoard BabyAI tasks using frozen memory constructed from the dependent or conventional stream. *SR*: success rate. *PR*: progress rate.

	Dependent Stream		Conventional Stream	
Method	SR	PR	SR	PR
ReAct	36.1	58.8	36.1	58.8
ExpRAG	30.6	53.8	**47.2**	60.7
ReMem	36.1	53.2	41.7	61.9
MEMPROBE	**41.7**	**64.8**	44.4	**63.1**

Table 20: Case study on coding tasks. Retrieved memory context helps when it captures the reusable subtask structure, but hurts when a topically similar task has incompatible semantics.

Positive transfer: correct compositional relation (BigCodeBench/516; MEMPROBE pass, ReAct fail)

Target decisive condition. The target function receives multiple pairs of numeric lists and must return three lists: Euclidean distances, DataFrames, and Axes objects. The decisive test checks that each DataFrame has shape $(n, 2)$ with columns `A` and `B`, e.g., `pd.DataFrame({'A': [1,2,3], 'B': [4,5,6]})`.
Baseline behavior. The baseline misinterprets the DataFrame format and creates a one-column DataFrame with repeated row indices `A` and `B`, yielding shape $(2n, 1)$ instead of $(n, 2)$. The test fails with a shape mismatch: predicted $(6, 1)$ vs. expected $(3, 2)$.
Retrieved memory. The retrieved task is the single-pair version of the same computation: given two lists `a` and `b`, compute `distance.euclidean(a,b)`, construct `pd.DataFrame({'A': a, 'B': b})`, plot the two series, and return `(distance, df, ax)`.
Inter-task relation. The target is a list-level composition of the retrieved task: the same single-pair operation should be applied independently to each pair, and the corresponding outputs should be collected.
MEMPROBE behavior. The memory agent reuses the retrieved DataFrame construction, `pd.DataFrame({'A': a, 'B': b})`, inside a loop over pairs. This preserves the required $(n, 2)$ structure and produces the expected output lists.

Negative transfer: superficial task similarity (BigCodeBench/877; MEMPROBE fail, ReAct pass)

Target decisive condition. The target function must combine numeric DataFrames, return a covariance matrix and a Seaborn pair plot, and reject invalid inputs. The decisive test specifically passes a DataFrame with one non-numeric column, `{'A': ['a','b'], 'B': [3,4]}`, and expects a `TypeError`.
Baseline behavior. The baseline checks every column with `is_numeric_dtype`. It detects the non-numeric column `A` and raises `TypeError`, matching the target requirement.
Retrieved memory. The retrieved task also processes a list of DataFrames, but its goal is different: fill missing numeric values, standardize numeric columns, compute correlation matrices, and draw heatmaps.
Inter-task relation. The memory is topically similar but semantically mismatched. In the retrieved task, non-numeric columns can be ignored by selecting numeric columns. In the target task, any non-numeric column makes the input invalid.
MEMPROBE behavior. The memory agent follows the retrieved pattern and uses `select_dtypes(include=[np.number])`. For the mixed DataFrame, it keeps column `B`, drops non-numeric column `A`, and continues to compute covariance and pair plots instead of raising `TypeError`.

Table 19 reports results on 36 held-out AgentBoard BabyAI tasks that occur in neither stream, using memory constructed separately on each stream and frozen before evaluation. MEMPROBE improves both SR and PR over memoryless ReAct after either stream.

B.14 CASE STUDY

We show two coding cases in Table 20, as discussed in §5.4.

C LICENSE OF EXISTING ASSETS

We list the utilized assets and their licenses. The released dataset artifacts of CodeEval-Pro are labeled MIT. The BrowseComp-Plus dataset release is labeled MIT. The MMLU-Pro dataset is labeled MIT, while its official GitHub repository is Apache-2.0. ScienceWorld is Apache-2.0. These releases are usable for our academic research.

Table 21: The statistics of subtasks and tasks on BigCodeBench-Lite-Pro. Each cell contains the mean and standard deviation.

Statistics	Subtask	Task
Query tokens	304.8 ± 118.1	121.3 ± 34.4
Solution tokens	100.7 ± 46.2	87.2 ± 53.6
Solution Lines	11.9 ± 5.7	10.7 ± 5.2
Function Calls in Solution	7.4 ± 3.3	5.9 ± 4.0

Table 22: The statistics of subtasks and tasks on BrowseComp+. Each cell contains the mean and standard deviation.

Statistics	Subtask	Task
Query Tokens	29.7 ± 9.9	72.5 ± 12.0
Number of evidence documents	1.8 ± 0.8	5.0 ± 2.1
Tokens in evidence documents	$6.3\text{k} \pm 24.0\text{k}$	$22.7\text{k} \pm 52.0\text{k}$
Query-token recall in evidence documents	$83.3\% \pm 12.4\%$	$73.5\% \pm 14.8\%$
Answer-token recall in evidence documents	$99.9\% \pm 1.9\%$	$99.2\% \pm 6.0\%$

D New Assets

The only newly generated tasks are the synthesized BrowseComp+ subtasks. We also release derived benchmark artifacts, including stream orders and evaluation splits constructed from existing datasets.

D.1 Synthesis of BrowseComp+ Subtasks

BrowseComp+ is a challenging deep research benchmark with complex, artificially constructed multi-hop information-seeking tasks. Because independently sampled complex tasks may share little reusable structure, directly streaming them does not provide a continuous deep research workflow with reusable knowledge. We therefore construct a dependent stream whose purpose is to create known transfer opportunities for evaluation. Concretely, we use the original BrowseComp+ annotations only during offline benchmark construction to identify localized, evidence-supported intermediate questions for each parent task. We use GPT-5.2 as an LLM generator. It is given the parent task together with its final answer and supporting gold/evidence documents, and is asked to propose self-contained intermediate subtasks only when it believes a faithful decomposition is possible. This privileged information is never exposed to the evaluated agents or memory methods at test time. Each generated subtask is paired with a concise reference answer and supporting document identifiers solely for dataset verification and automatic evaluation. We retain a candidate only if its answer is evidence-supported, deterministic, and logically relevant to the parent task. Although the generation prompt encourages decompositions of 2–5 steps, this cardinality is not hard-enforced. The final stream keeps only the verified subset that survives filtering.

We enforce quality control of generated subtasks in two stages. First, we apply deterministic filtering to remove malformed candidates. A candidate subtask is retained only if both the subtask and its answer are non-empty, all cited document identifiers come from the original gold/evidence pools, both gold documents and evidence documents are non-empty, and the cited gold documents are a subset of the cited evidence documents. By default, we further require the proposed answer to appear verbatim in the supporting documents, which suppresses unsupported paraphrases and hallucinated answers. Second, each surviving candidate is re-verified by the model with a separate verification prompt conditioned on the original task and the cited supporting documents. The verifier checks five properties: explicit entity specification, logical relevance to the original task, evidence support for the required reasoning step, determinism of the answer, and answer completeness. Only candidates that pass all verifications are retained as final subtasks. Tasks for which the model declines decomposition, returns an invalid response, or yields no valid candidate after filtering are skipped.

Our focus is on evidence sharing within multi-hop logical chains, rather than on using subtasks to directly prompt the original task. Entities, timestamps, and other information mentioned in the original task are not required to appear in the subtasks.

D.2 The Statistics of Dependent Streams

Both Table 21 and Table 22 compare the statistics of subtasks and complex tasks, revealing a shared compositional pattern in which task-level instances aggregate more information or complexity than their constituent subtasks. In BigCodeBench-Lite-Pro, subtasks are more verbose than full tasks in terms of query length, and they also involve slightly longer solutions and more function calls. This suggests that the subtasks have a smaller problem scope but include more implementation details, whereas the complex tasks are often wrappers around these smaller problems. In contrast, BrowseComp+ exhibits the opposite trend for query length, where complex tasks are longer and require more evidence documents and many more supporting tokens than subtasks, indicating that composition primarily increases retrieval breadth and evidence integration demands. Across both coding and deep-research tasks, the transition from subtasks to tasks is associated with a change in information organization rather than a simple uniform scaling of all statistics.

We further examine whether the synthesized BrowseComp+ subtasks collapse into paraphrases of their parent task. The token-set Jaccard similarity between subtasks and their parent task is low, with a mean of 0.10, and no pair exceeds 0.30. Only 5.8% of individual subtasks cover the full parent evidence document set, and only 1.6% simultaneously have the same normalized answer and the same evidence set as the parent task. These diagnostics suggest that BrowseComp+ subtasks are semantically related localized evidence steps rather than full-task paraphrases or near-duplicate restatements of the complex tasks.