

Benchmarking Persistent Project Memory in Local Language Models: CAG vs RAG vs DAG over 30 Sequential Tasks

Seth Canfield
Guideboard Labs
guideboardlabs@proton.me

May 2, 2026

Abstract

We present CAG-Bench, a longitudinal benchmark for evaluating how well local language models maintain project context across a sequence of 30 interdependent software development tasks. We compare three context strategies: fresh source-only retrieval (RAG), fixed-workflow generation (DAG), and Context Accumulation Generation (CAG), in which validated project decisions are written to a persistent memory store and reused on subsequent tasks. On `qwen2.5-coder:7b` (3 trials), CAG achieves a composite score of 48.5 vs. 29.6 (RAG) and 28.5 (DAG), with continuity recall of 54.2% vs. 17.1% and 17.0% respectively; the effect is replicated directionally at 3B scale (10 trials). Within the CAG family we evaluate four memory-selection variants: an unbounded dump (`cag`), a label-free deployable retriever (`cag_scoped_promptonly`), and two diagnostic upper bounds that use answer-key metadata in their selection logic (`cag_scoped`, `cag_oracle_memory`). The deployable retriever still substantially outperforms RAG/DAG (composite 42.6, continuity 41.0), but trails the label-informed diagnostics by approximately 20 percentage points of memory recall—a real and previously hidden retrieval gap. We introduce `memory_usage_rate`, a diagnostic metric measuring whether selected memory concepts appear in the model’s final answer. Across CAG variants, memory usage declines over later project phases (62.6% → 45.8% → 38.1% for base CAG), suggesting that **memory uptake—not merely memory retrieval—is a central bottleneck**. Because our benchmark uses grounded task-defined memory rather than model-generated memory, these results isolate retrieval and uptake from memory-formation errors. Benchmark data, scoring code, raw outputs, and figures are released under AGPL-3.0-or-later.

1 Introduction

Large language models performing software development tasks face a challenge that single-turn benchmarks cannot capture: context continuity across a long-running project. A model asked to implement a “Review lane with ProgressSnapshot” on task 28 must recall that ProgressSnapshot was defined on task 11, that the persistence layer uses SQLite with soft-delete semantics from task 3, and that the project explicitly rejects cloud-sync from task 1. No single retrieved document contains all of this; it is accumulated project knowledge.

Existing benchmarks measure isolated task quality. They do not measure whether a model-assisted workflow degrades over time as context grows, whether retrieval-based approaches can compensate for that growth, or where the failure modes lie when they cannot.

CAG-Bench is designed to answer these questions precisely. It presents a model with 30 sequential, explicitly interdependent tasks—each task’s correct solution requires recalling decisions made in prior tasks. The benchmark compares three fundamentally different context strategies, instruments them with five distinct memory-quality metrics, and produces per-phase diagnostics to localize where each strategy succeeds and fails.

The benchmark is local-only (Ollama), deterministic in its scoring, and runs without external services. All memory is grounded: durable project facts are promoted from task-defined ground truth, not from model output. This design prevents the most common failure mode in memory-augmented generation—hallucinated facts becoming trusted context.

2 Background

2.1 RAG and its limits in iterative work

Retrieval-Augmented Generation retrieves relevant documents from a static corpus at each query. For iterative software development, the limitation is structural: prior model decisions are not documents. When the model chooses SQLite over PostgreSQL on task 3, that choice lives in a prior model response—not in a source file. RAG cannot retrieve it unless it is explicitly stored and indexed.

2.2 CAG: Context Accumulation Generation

CAG addresses this by maintaining a growing store of project decisions, accumulating accepted decisions after each task and providing them as context for subsequent tasks. The core principle: **accepted project decisions from ground-truth task metadata are promoted into persistent memory and reused on later tasks**. Model output is not directly promoted; only task-defined `promote_summary` and `promote_terms` fields enter durable memory. This grounding contract ensures the memory store contains only validated facts.

2.3 Selective retrieval within CAG (diagnostic variants)

As the memory store grows across 30 tasks, indiscriminate dumping of all accumulated context becomes expensive and may overwhelm a model’s effective context window. We evaluate two selective-retrieval variants designed to study the upper bound of selection quality. **Both variants use the task’s `continuity_terms` metadata**—the same field used for output scoring—as a retrieval feature. They are therefore diagnostic upper bounds, not deployable production retrievers:

- **cag_scoped**: Top-K scoring using $3.0 \times \text{concept_overlap} + 1.0 \times \text{tag_overlap} + 0.5 \times \text{task_text_overlap} + 1.0 \times \text{recency_weight}$, where `concept_overlap` counts how many task `continuity_terms` groups a memory row matches. This is label-informed scoring—a deployable retriever would not have access to `continuity_terms` at retrieval time.
- **cag_oracle_memory**: A stricter diagnostic that filters memory rows to only those whose `promoted_terms` intersect the task’s `continuity_terms`, then ranks by concept overlap. The model sees only the selected memory text; the scoring key itself is never inserted into the prompt.

The current `cag_scoped` variant is therefore diagnostic rather than production-realistic: it uses task-defined continuity metadata to approximate an ideal selector. We interpret the scoped/oracle comparison as an upper-bound analysis of capacity and uptake—not as evidence that a deployable retriever has solved memory selection. A true prompt-only retriever (using only task title, prompt, tags, and source documents—never `continuity_terms`) is described in Section 8.

3 Related Work

3.1 RAG evaluation frameworks

The RAGAS framework [1] provides the most widely adopted suite for evaluating retrieval-augmented pipelines. It defines four per-call metrics—faithfulness, answer relevancy, context precision, and context recall—computed independently for each query. RAGAS does not define a composite score with fixed weights; each metric is reported independently. Crucially, RAGAS is designed for single-turn or independent-query evaluation. It has no notion of accumulated context, no continuity across calls, and no mechanism for measuring whether prior decisions from query N are honored at query $N + 28$. CAG-Bench’s composite weighting (40% continuity recall, 34% checklist quality, 12% each source evidence and domain rule recall) reflects a deliberate choice to prioritize longitudinal coherence over single-call retrieval fidelity.

TruLens [2] extends similar principles with LLM-as-judge groundedness and relevance scoring. Like RAGAS, it is oriented toward retrieval quality on a per-call basis and does not address context accumulation across a task sequence.

3.2 Long-context comprehension benchmarks

LongBench [3] evaluates long-context comprehension across 21 tasks in six categories, including single-document QA, multi-document QA, and code completion. Reported means across tasks: GPT-3.5-Turbo 44.0, GPT-4 54.4, Llama-2-13B-Chat 24.4. LongBench v2 (2024) extends this to 503 challenging multiple-choice questions with contexts up to 2M words, including long-dialogue history understanding. Both versions establish that model capability degrades on tasks requiring information from early portions of long documents, particularly for smaller models. However, LongBench provides the long context as a given input—the model is a passive reader of a pre-assembled document. CAG-Bench differs fundamentally: there is no pre-assembled document. The model must perform tasks in sequence, with each task’s correct answer depending on what was decided in prior model interactions. The challenge is not reading a long document; it is accumulating a coherent project history across independent calls.

LongMemEval [4] is the most directly relevant long-context benchmark. It evaluates five memory abilities—information extraction, multi-session reasoning, temporal reasoning, knowledge updates, and abstention—across 500 questions embedded within scalable chat histories of up to 1.5M tokens. Key result: commercial chat assistants show a 30% accuracy drop on memorizing information across sustained interactions, and long-context LLMs including GPT-4o show a 30–60% performance decline compared to oracle retrieval settings. This directly parallels CAG-Bench’s finding that RAG/DAG continuity recall collapses from approximately 32% to approximately 6% across the three task phases. The critical difference: LongMemEval tests factual Q&A retrieval from chat history; CAG-Bench tests decision integration into actively generated software development plans.

The output is an action, not a lookup.

MemoryAgentBench [5] evaluates memory agents across four competencies derived from cognitive science: accurate retrieval, test-time learning, long-range understanding, and conflict resolution. It reconstructs existing long-context datasets into incremental multi-turn formats and introduces two new datasets (EventQA, FactConsolidation) for competencies not covered by existing data. This is the closest structural parallel to CAG-Bench in the memory literature—both benchmark memory across sequences of incremental inputs. The distinction is domain: MemoryAgentBench evaluates conversational and factual memory; CAG-Bench evaluates whether project-level architectural decisions made in prior development tasks are honored in subsequent ones. Additionally, MemoryAgentBench allows model-generated memory; CAG-Bench’s grounding constraint prevents unvalidated model output from entering durable memory, enabling cleaner isolation of retrieval and uptake as independent failure modes.

SCROLLS [6] and ∞ BENCH [7] address comprehension at greater length. ∞ BENCH reports GPT-4 performance degrading sharply beyond 100K tokens in retrieval tasks. These benchmarks treat context as externally provided input; none model the accumulation problem.

3.3 Memory-augmented generation systems

MemGPT [8] introduces OS-inspired hierarchical memory management for LLMs, with explicit paging between main context and external storage. On multi-session conversation and document analysis tasks exceeding a model’s context window, MemGPT substantially outperforms fixed-context baselines. The key architectural difference from CAG-Bench’s design: MemGPT allows the model to manage memory operations—deciding what to store, retrieve, and evict. CAG-Bench’s grounding constraint takes the opposite position: memory content comes from task-defined facts, not model output. This is a deliberate benchmark choice to isolate retrieval and uptake problems from memory formation quality; it does not imply model-managed memory is wrong for production use.

Generative Agents [9] introduce the retrieve-reflect-plan memory loop. MemoryBank [10] applies Ebbinghaus forgetting curve mechanics to LLM memory and reports approximately 15–20% improvement in personality and fact consistency over vanilla ChatGPT. Neither addresses the grounding problem—both allow model-generated content to enter persistent memory without validation. The downstream risk (confirmed empirically in earlier versions of this benchmark’s development) is that hallucinated technical choices—MongoDB, JWT, cloud sync—can become trusted project facts. PersistBench [11] directly quantifies these risks, reporting median failure rates of 53% on cross-domain memory leakage and 97% on memory-induced sycophancy across persistent memory systems. These failure modes arise specifically when model-generated memories are stored without validation, and motivate the grounding contract used in CAG-Bench.

3.4 Iterative software engineering benchmarks

SWE-bench [12] evaluates LLMs on resolving real GitHub issues against a test suite. Each issue is evaluated independently: the model receives the repository context as a snapshot without accumulated interaction history. Early GPT-4 results were approximately 1.7–2.7% resolution rate; top systems now exceed 49% on the Verified subset. SWE-bench measures capability to modify existing code; CAG-Bench measures capability to make consistent incremental decisions across a project lifecycle where no codebase exists yet.

SWE-EVO [13] is a significant recent advance: it benchmarks coding agents on long-horizon software evolution, comprising 48 tasks spanning an average of 21 files across seven mature open-source Python projects. GPT-5 with OpenHands achieves approximately 21% on SWE-EVO versus approximately 65% on SWE-Bench Verified, demonstrating that state-of-the-art agents fail dramatically on sustained multi-file reasoning compared to single-issue resolution. CAG-Bench and SWE-EVO are complementary: SWE-EVO tests evolution of existing codebases against real test suites; CAG-Bench tests accumulation of project decisions from scratch with concept-group scoring. Neither answers the other’s question.

Most directly parallel is **SlopCodeBench** [14], which evaluates how coding agents degrade over 20 problems and 93 checkpoints as specifications evolve and agents carry their own prior code forward. Key findings: no agent solves any problem end-to-end across 11 models; the highest checkpoint solve rate is 17.2%; agent code is $2.2\times$ more verbose than comparable open-source projects and degrades structurally faster than human code. This confirms CAG-Bench’s central finding—that iterative context degrades model performance—from the code quality angle. The distinction is dimension: SlopCodeBench measures structural code degradation (verbosity, complexity erosion); CAG-Bench measures decision continuity (whether project-level commitments are honored in subsequent outputs). Both findings are necessary; neither is sufficient alone.

HumanEval [15] and DevBench [16] evaluate isolated or single-pass code generation and are entirely stateless with respect to project history.

3.5 Where CAG-Bench stands

Recent work establishes the iterative degradation problem from multiple angles: LongMemEval shows 30–60% accuracy loss in conversational memory; SWE-EVO shows a 47-point gap between single-issue and multi-evolution performance; SlopCodeBench shows no agent completes an iterative specification sequence end-to-end. CAG-Bench contributes three things these works do not: (1) a controlled comparison of retrieval paradigms (RAG, DAG, CAG, scoped, oracle) on the same task sequence; (2) the `memory_usage_rate` metric that separates retrieval failure from uptake failure; and (3) a grounding contract that prevents model hallucinations from polluting the memory store, enabling clean measurement of retrieval and uptake independently of memory formation quality.

4 Benchmark Design

4.1 Task structure

The benchmark uses 30 sequential tasks describing an iterative software project (**PawLedger**, a local-first dog training workspace). Tasks are explicitly interdependent: each task from T02 onward contains `continuity_terms`—concepts that must appear in a correct answer, derived from decisions made in prior tasks. Task 1 has no continuity requirements. By task 30, a correct answer must reference 31 distinct prior concepts.

Each task also carries:

- `promote_summary`: the canonical one-sentence project decision that this task establishes
- `promote_terms`: the key technical terms associated with that decision

- `checklist_terms`: implementation checklist items expected in the answer
- `source_evidence_terms`: concepts from the source document corpus
- `domain_rule_terms`: domain-specific constraints (e.g., “humane training”, “no cloud sync”)
- `contradiction_terms`: terms that indicate a model invented something that conflicts with the project spec

4.2 Memory model

After each task runs, `ProjectMemory.add()` writes one row to a per-trial JSONL store using the task’s `promote_summary` as the memory text. This promotion is deterministic and auditable: a companion `_promotions.jsonl` file records every promotion decision with its reason (`supported_by_promote_summary` or `task_local`). No model output is ever promoted. In all runs reported here, 30/30 tasks produce a `promote` decision since all tasks carry a non-empty `promote_summary`.

4.3 Modes

Table 1: Mode definitions and token budgets at task 30.

Mode	Memory source	Selection strategy	T30 mean tokens
<code>rag</code>	None	Top-3 keyword source retrieval	418
<code>dag</code>	None	Fixed workflow prompt + top-3 retrieval	448
<code>cag</code>	Full accumulation	Top-K by keyword score, cap=25	1,850
<code>cag_scoped</code>	Full accumulation	Top-5 by composite score	692
<code>cag_oracle_memory</code>	Full accumulation	Top-5 by continuity-overlap filter	692

4.4 Scoring

All scoring is deterministic and concept-group based. Each term group has a canonical label and a set of accepted synonyms. A concept counts as a hit if any accepted synonym appears in the model answer; plain string terms are treated as singleton groups.

Composite score:

```
score = 0.34 * checklist_quality
      + 0.12 * source_evidence_recall
      + 0.12 * domain_rule_recall
      + 0.40 * continuity_recall
      + 0.01 * token_efficiency
      + 0.01 * latency_efficiency
      - contradiction_penalty
```

Continuity recall carries the highest weight (40%) because it measures the core benchmark property: are prior project decisions present in the current answer?

4.5 Memory diagnostic metrics (new in this work)

Four metrics instrument the memory pipeline at distinct stages:

Table 2: Memory diagnostic metrics introduced in this work.

Metric	Measures	Stage
<code>memory_recall</code>	% of task’s expected concepts in selected memory	Retrieval
<code>memory_precision</code>	% of selected memory concepts relevant to current task	Retrieval
<code>continuity_recall</code>	% of expected concepts in final answer	Output
<code>memory_usage_rate</code>	% of selected memory concepts in final answer	Uptake

`memory_usage_rate` is introduced in this work. It bridges the gap between `memory_recall` (what was retrieved) and `continuity_recall` (what appeared in the answer), enabling diagnosis of whether retrieval is the bottleneck or whether retrieved memory is being ignored by the model.

5 Experimental Setup

All experiments run locally via Ollama on the same hardware. Two models are evaluated:

- `qwen2.5-coder:7b`—primary model
- `qwen2.5-coder:3b-instruct`—secondary, lower-capacity model

Fixed parameters across all runs: temperature 0.1, context window 8192 tokens, source retrieval $K = 3$ (keyword-based). Memory top-K defaults: 5 for `cag_scoped/cag_oracle_memory`, 25 for base `cag`. All 30 tasks are run in sequence; each trial is a fresh independent pass through all 30 tasks with a fresh memory store.

Table 3: Run configurations.

Run	Model	Suite	Tas
Baseline 7B	<code>qwen2.5-coder:7b</code>	rag, dag, cag	30
Baseline 3B	<code>qwen2.5-coder:3b-instruct</code>	rag, dag, cag	30
CAG Ablation 7B	<code>qwen2.5-coder:7b</code>	cag, cag_scoped, cag_scoped_promptonly, cag_oracle_memory	30

6 Results

6.1 RAG vs DAG vs CAG: baseline comparison

The `rag` baseline should be interpreted as a source-only retrieval system without durable project memory—its purpose is to test whether fresh document retrieval alone can substitute for accumulated context. We do not claim it is a state-of-the-art retrieval system; we claim that source-only retrieval is structurally insufficient for tasks whose correct answers depend on prior model interactions. The `dag` baseline tests the same hypothesis under a fixed workflow prompt instead of free generation, isolating whether prompt structure alone can compensate for missing memory.

Table 4: Overall means across 30 tasks (qwen2.5-coder:7b, 3 trials).

Mode	Composite	Continuity	Checklist	Src. Ev.	Dom. Rule	Contra.	Tokens
RAG	29.6	17.1	33.1	37.3	47.2	1.50	374
DAG	28.5	17.0	33.7	40.0	41.7	2.00	404
CAG	48.5	54.2	42.4	38.1	56.4	1.00	1,087

CAG outperforms RAG by +18.9 composite points and +37.1 continuity points in this run. The effect is replicated directionally at 3B scale (RAG 28.0, DAG 28.6, CAG 45.7) over 10 trials, suggesting the ordering is not specific to a single model size, though formal hypothesis testing is left to future work given the small trial counts ($n = 3$ for the 7B baseline).

Two findings stand out beyond the headline numbers. First, **source evidence recall is essentially equivalent across all three modes** (37–40%), confirming that all modes retrieve relevant documentation similarly well—CAG’s advantage is not from better source access. Second, **domain rule recall is substantially higher for CAG** (56.4 vs. 47.2 for RAG on the 7B model; 59.8 vs. 40.3 on 3B). Domain rules are project-specific constraints that are encoded in the accumulated memory decisions. The model is more reliably applying them when they are explicitly carried in context.

DAG does not improve over RAG. The structured workflow prompt adds token overhead (+8%) without improving any recall metric, and shows the highest mean contradiction penalty (2.00). This suggests that fixed-structure prompting may interfere with task-specific reasoning without compensating through better context access.

Table 5: Phase breakdown—composite score and continuity recall (qwen2.5-coder:7b).

Phase	RAG score	DAG score	CAG score	RAG cont.	DAG cont.	CAG cont.
T01–T10	44.6	40.5	57.8	31.8	31.0	70.7
T11–T20	31.8	30.5	56.7	15.1	15.8	59.0
T21–T30	12.4	14.6	30.9	5.9	5.6	34.6

Two patterns are notable here. First, **RAG and DAG degrade sharply and symmetrically**—by the final 10 tasks, both modes score below 15 and continuity drops to approximately 6%. Fresh retrieval cannot compensate for the growth in expected prior concepts: at task 28, a correct answer needs to reference 28+ decisions that live exclusively in the accumulated project history.

Second, **CAG’s largest relative advantage occurs in the middle phase (T11–T20)**, not at the end. CAG leads RAG by +13.2 points in T01–T10, widens to +24.9 in T11–T20, then narrows to +18.5 in T21–T30. The middle-phase peak suggests that accumulated context is most effective once a meaningful project history has formed but before the memory store becomes large enough to dilute the model’s attention.

At task 30 specifically—the final implementation handoff task—the gap is stark: CAG achieves 46.0% continuity recall vs. 9.2% for RAG and 4.6% for DAG. This is the task where all 30 prior decisions are simultaneously required. Even CAG does not reach the theoretical maximum, but it is the only mode that provides a practically useful response.

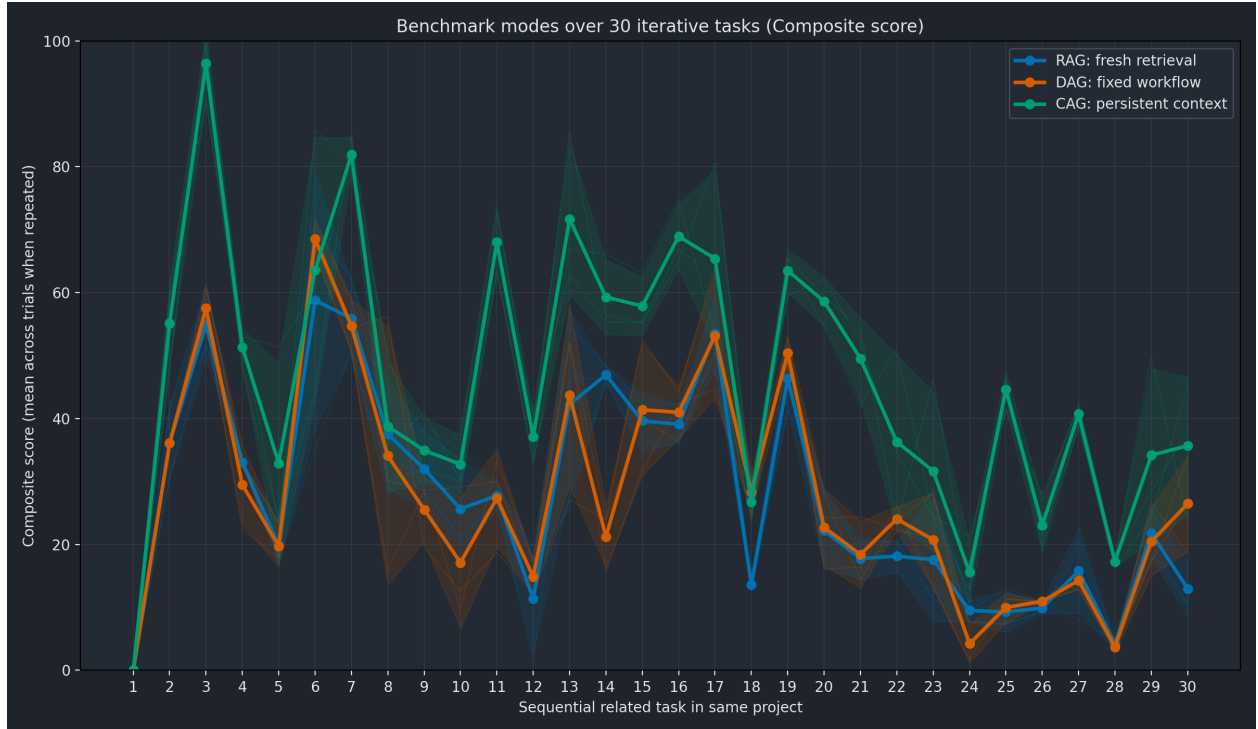


Figure 1: Composite score by task index—RAG and DAG collapse monotonically while CAG holds a substantially higher floor through all 30 tasks.

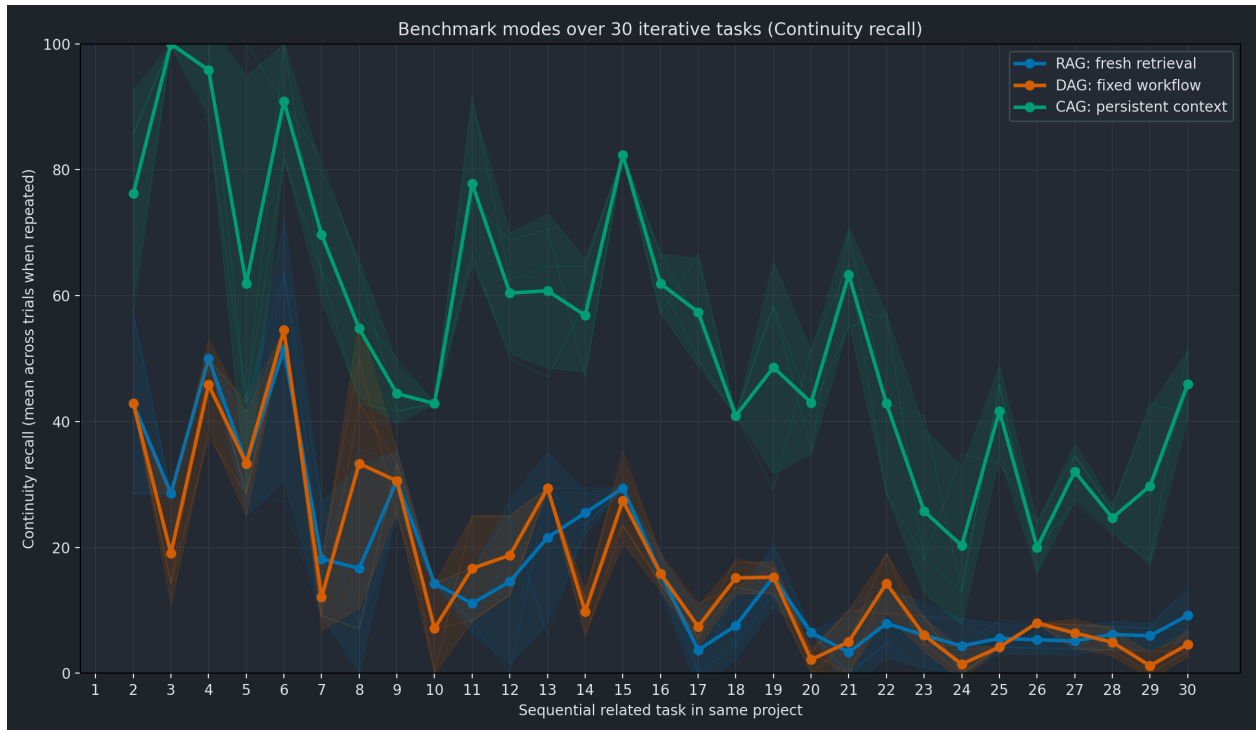


Figure 2: Continuity recall by task index—the metric most directly measuring prior-decision coverage. RAG and DAG approach zero by task 20; CAG holds a meaningful floor.

6.2 CAG internal ablation: leakage gap, capacity, and the deployable retriever

Having established CAG’s advantage over retrieval-only approaches, we evaluate four CAG variants on the same memory store: base `cag` (unbounded dump capped at 25), `cag_scoped` (label-informed top-K), `cag_oracle_memory` (label-informed filter, diagnostic ceiling), and `cag_scoped_promptonly` (label-free top-K—the only variant whose retrieval would be implementable in production). All variants except `cag_scoped_promptonly` consult the task’s `continuity_terms` field—also the answer-key field used in scoring—and are therefore upper bounds, not deployable retrievers. `cag_scoped_promptonly` uses only the task title, prompt, tags, source documents, and recency.

Table 6: CAG ablation—overall means (qwen2.5-coder:7b, 3 trials, $K = 5$).

Mode	Composite	Continuity	Mem Recall	Mem Prec.	Usage Rate	Mem Tokens
<code>cag</code> (cap=25)	48.9	53.3	100.0	39.3	48.4	675
<code>cag_scoped</code> (label-informed)	46.2	49.9	80.7	53.0	54.1	210
<code>cag_scoped_promptonly</code> (deployable)	42.6	41.0	60.9	47.7	54.0	222
<code>cag_oracle_memory</code> (ceiling)	45.3	50.6	80.7	53.0	54.2	210

The label-leakage gap. The two label-informed variants (`cag_scoped`, `cag_oracle_memory`) produce identical aggregate memory recall and memory precision (80.7% / 53.0%) across all phases. This is partly structural: both use `continuity_terms` to drive selection, and they identify substantially the same rows by different routes. When that label is removed, retrieval quality drops materially: `cag_scoped_promptonly` reaches 60.9% memory recall vs. 80.7% for the label-informed variant—a 19.8 percentage-point gap. Continuity recall in the answer drops correspondingly (49.9 → 41.0). The original framing of `cag_scoped` as a deterministic non-oracle retrieval baseline was incorrect; the variant should be interpreted only as an upper-bound diagnostic. Reporting these numbers side-by-side with the deployable variant makes the size of the ceiling-vs-deployable gap explicit.

The deployable retriever still beats RAG/DAG. Despite the leakage gap, `cag_scoped_promptonly` substantially outperforms the no-memory baselines: composite 42.6 vs. 29.6 (RAG), continuity 41.0 vs. 17.1, memory recall 60.9 vs. 0. The advantage of accumulated project memory does not depend on label leakage. It does depend on retrieval, however—see Section 7.4 below.

Capacity is the binding constraint at the upper bound. Within label-informed variants, both `scoped` and `oracle` plateau at 62% memory recall in T21–T30 because $K = 5$ cannot cover the 20+ concepts late tasks require. Improvements to the scoring formula at this capacity, given the same labels, will not change outcomes—the two variants already select essentially the same rows. The remaining headroom under label-informed selection comes from larger K , not better scoring. Whether the same conclusion holds for the deployable variant—where retrieval has more headroom to improve—is an open question; preliminary evidence in Section 7.4 suggests retrieval scoring matters more without labels.

The phase breakdown shows the leakage gap widening with task complexity. In T01–T10, all four variants are roughly comparable on memory recall (96–100%)—early tasks have few enough continuity concepts that label-free retrieval can find them through ordinary lexical and tag overlap. By T11–T20 the gap opens (promptonly 59.9 vs. `scoped` 82.6), and by T21–T30 promptonly reaches half the recall of the label-informed variants (30.5 vs. 62.1). The same pattern appears in continuity recall, where promptonly drops from competitive in T01–T10 (67.4) to a sharp deficit in T21–T30 (15.6 vs. `scoped` 37.7).

Table 7: Phase breakdown—CAG ablation (memory recall and continuity, `qwen2.5-coder:7b`). Promptonly = `cag_scoped_promptonly`.

Phase	Memory recall				Continuity recall			
	CAG	Scoped	Promptonly	Oracle	CAG	Scoped	Promptonly	Oracle
T01–T10	100.0	99.2	95.8	99.2	71.2	64.7	67.4	72.8
T11–T20	100.0	82.6	59.9	82.6	54.9	48.9	42.8	48.8
T21–T30	100.0	62.1	30.5	62.1	35.4	37.7	15.6	32.5

Note also that `cag_scoped` continuity recall in T21–T30 (37.7) modestly exceeds base CAG (35.4) despite holding only 62% of the memory recall—the smaller, more focused selection yields slightly better answer-side continuity at the cost of significantly lower retrieval coverage. The “more is better” assumption from a generous dump does not hold once a focused selection is available, even at substantially lower memory recall.

Token cost: base CAG uses 675 tokens of memory context on average, growing to 1,850 at task 30. The $K = 5$ variants (scoped, oracle, promptonly) all hold approximately 216–222 tokens regardless of phase—a 67% reduction in context cost relative to the unbounded dump. If memory recall under deployable retrieval can be improved through a higher K or richer signal, scoped variants become clearly preferable on efficiency grounds.

This suggests that focused memory selection can outperform larger context dumps in efficiency terms—sustaining comparable continuity output at one-third the token cost—even when absolute recall is lower, highlighting a trade-off between coverage and usability that the `continuity_per_memory_token` metric captures directly (Figure 8).

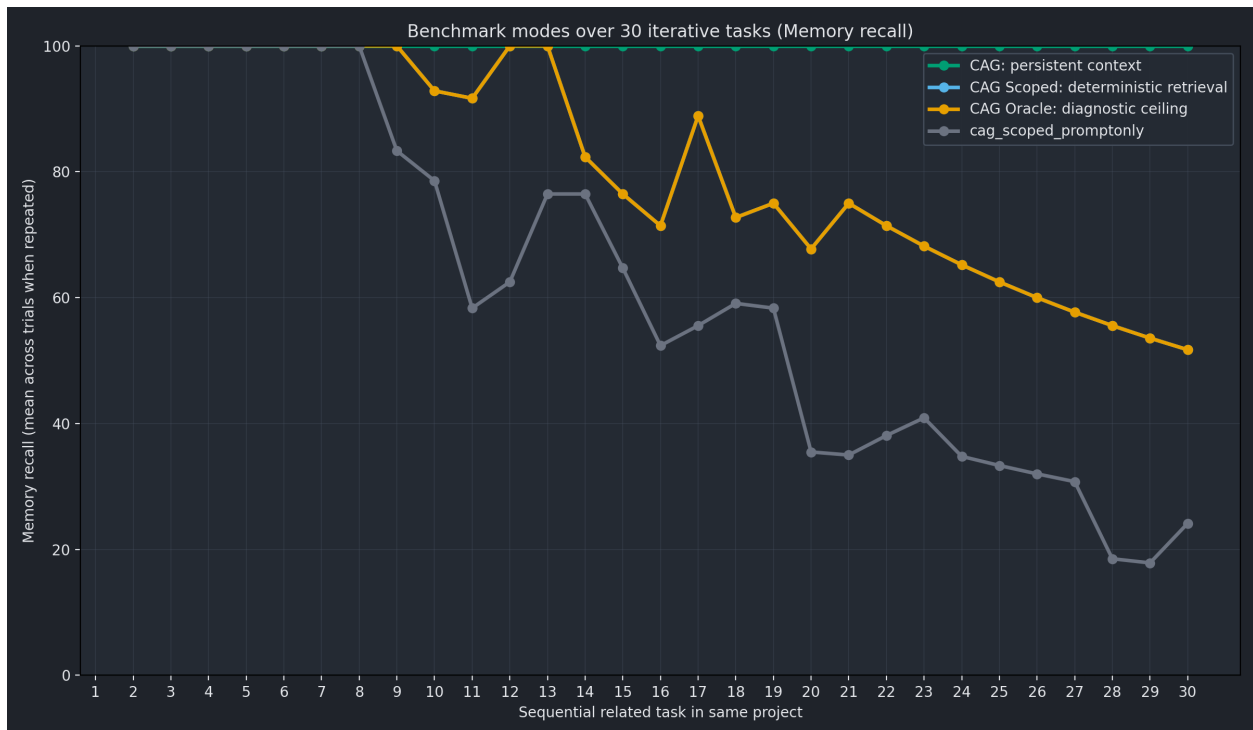


Figure 3: Memory recall by task index across CAG variants—base CAG holds 100% throughout; the two label-informed variants (scoped, oracle) converge at approximately 62% by tasks 21–30; the label-free deployable variant (`cag_scoped_promptonly`) drops to approximately 30%.

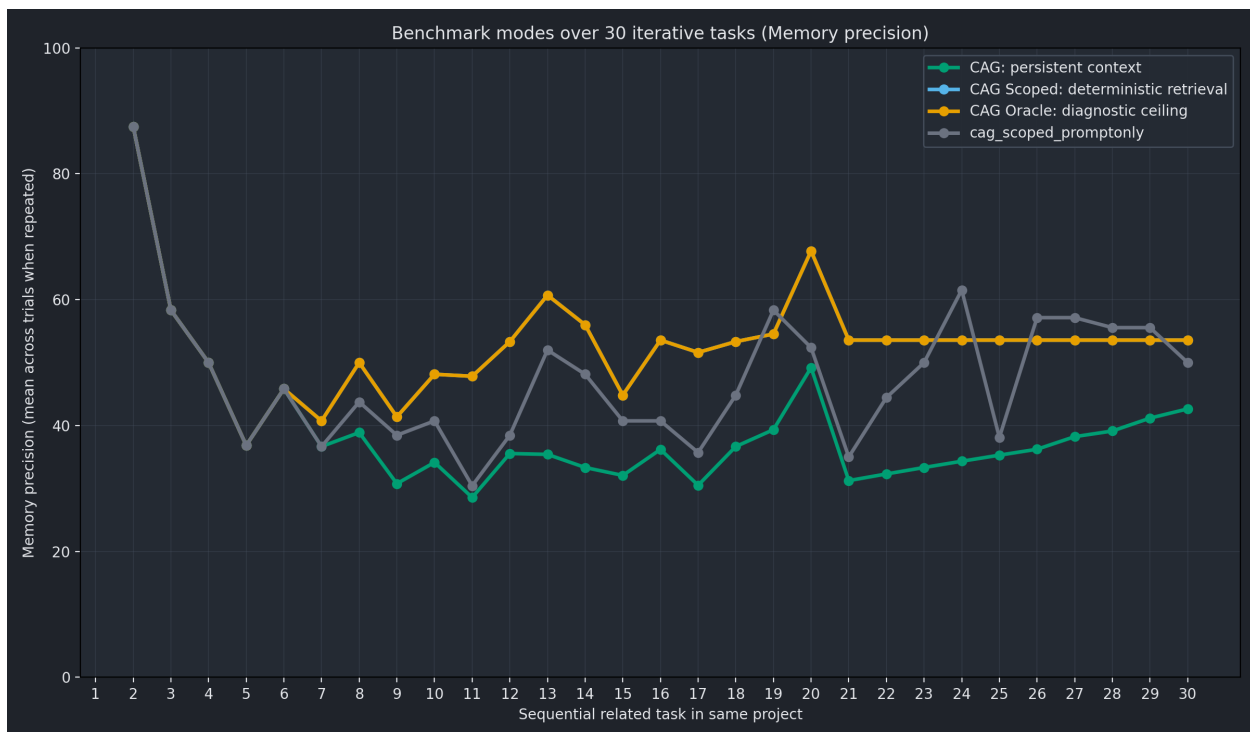


Figure 4: Memory precision by task index—scoped/oracle maintain approximately 53% precision; promptly trails at approximately 48% with substantially lower coverage.

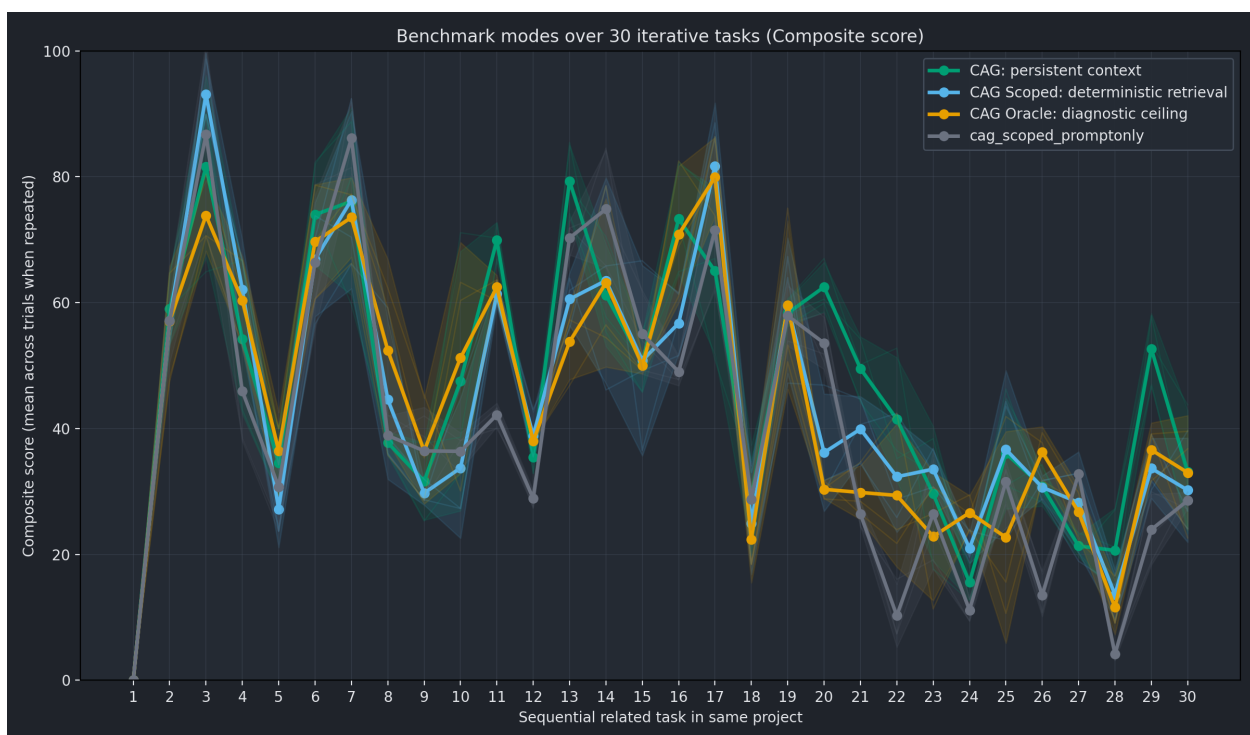


Figure 5: Composite score by task index across CAG variants—the deployable variant (promptonly) tracks closely to the label-informed variants in early phases and falls increasingly behind in late phases.

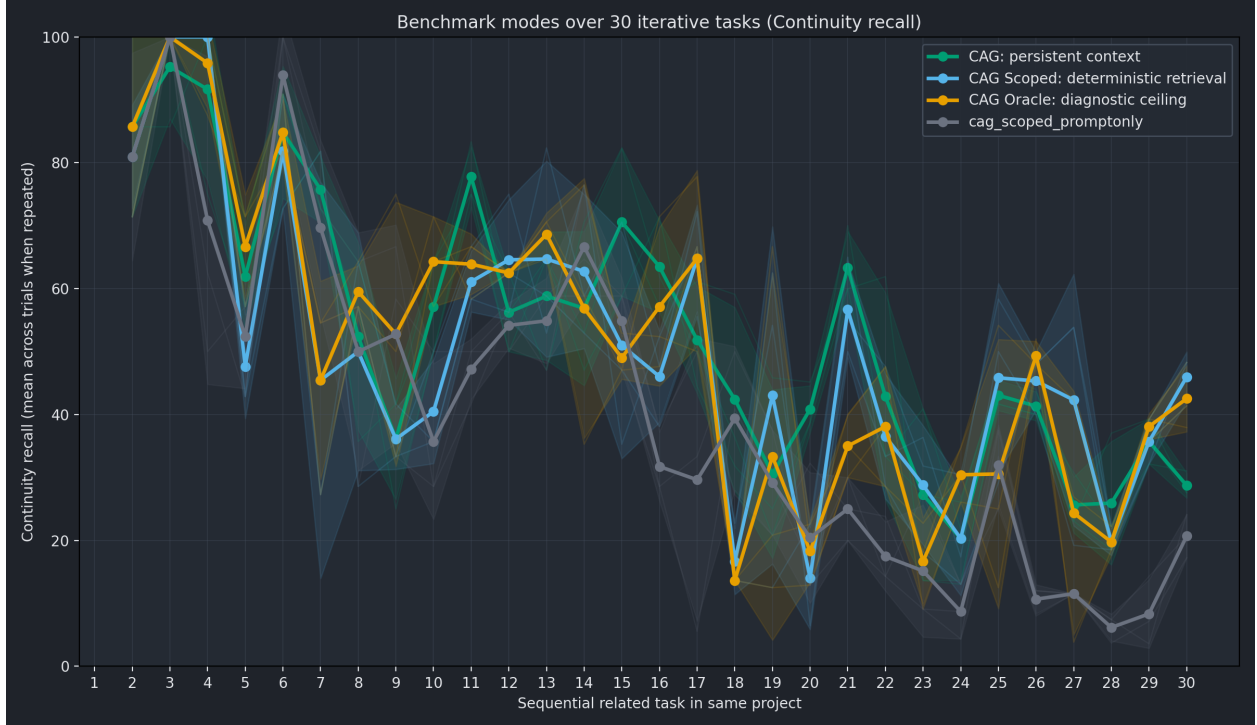


Figure 6: Continuity recall by task index across CAG variants—phase breakdown of the leakage gap.

6.3 Memory uptake: a new diagnostic axis

The most actionable finding from this benchmark concerns `memory_usage_rate`: the fraction of selected memory concepts that actually appear in the final answer.

Table 8: Memory usage rate by phase (`qwen2.5-coder:7b`, CAG ablation).

Phase	CAG	Scoped	Promptonly	Oracle
T01–T10	62.6%	56.8%	63.2%	61.9%
T11–T20	45.8%	48.1%	53.2%	49.1%
T21–T30	38.1%	57.7%	46.5%	52.3%

Memory usage rate declines across all four modes as task complexity increases. For base CAG, which holds 100% memory recall, less than 40% of retrieved concepts make it into the final answer in the late phase. Even when the correct prior decisions are present in the prompt, the model does not consistently use them.

The gap between `memory_recall` and `continuity_recall` quantifies the uptake deficit at the aggregate level: -47 points for base CAG, -31 for label-informed scoped, -20 for label-free promptonly. The smaller gap under label-free retrieval is a numerical consequence of its lower memory recall—when fewer concepts are retrieved, the absolute uptake gap is smaller even if the rate is similar. **Even under favorable or label-informed retrieval, models fail to reliably incorporate available memory into outputs, indicating that memory uptake is a primary bottleneck alongside retrieval rather than retrieval alone.** The deployable label-free retriever shows that retrieval, too, is a real bottleneck once label leakage is removed: the two failure modes operate together, and addressing only one is insufficient.

Surprise finding: label-free retrieval drives higher uptake in early and middle phases.

`cag_scoped_promptonly` achieves the highest memory usage rate of any variant in T01–T10 (63.2%) and T11–T20 (53.2%), exceeding both label-informed scoped and the oracle. The plausible interpretation: a label-free retriever selects memories whose surface text matches the current task prompt and tags. The model finds those memories more obviously relevant to its current reasoning and incorporates them more readily. Label-informed retrieval, by contrast, can pull in rows the answer-key says should be relevant but whose surface text the model does not naturally recognize as connected to the task. This advantage reverses by T21–T30, where `cag_scoped` (label-informed, 57.7%) outperforms `promptonly` (46.5%)—late tasks have so many continuity concepts that any selection covering them helps, regardless of whether the model immediately recognizes the connection.

This interpretation is consistent with the observation that `promptonly` retrieval selects memory rows with higher lexical overlap to the current task prompt, but we do not directly measure semantic alignment between selected memory and downstream model attention; confirming the mechanism would require attention-pattern instrumentation and is left to future work (Section 8.2). As a bounded hypothesis, the finding suggests that production retrieval systems may have a structural uptake advantage over oracle-style upper bounds in early phases of a project’s lifecycle, before the concept space becomes too large for surface matching to suffice.

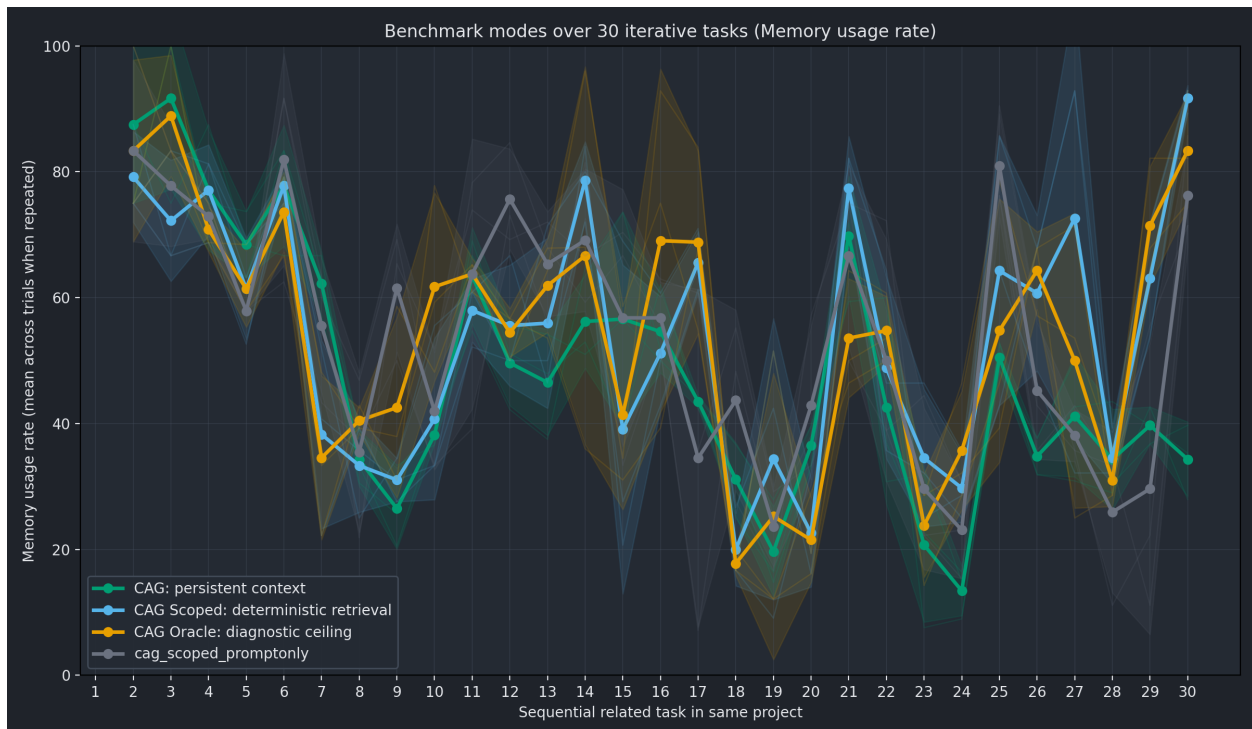


Figure 7: Memory usage rate by task index—the fraction of selected memory concepts that appear in the final answer. All modes decline over time; base CAG shows the steepest fall despite holding 100% memory recall, consistent with attention dilution under growing context load.

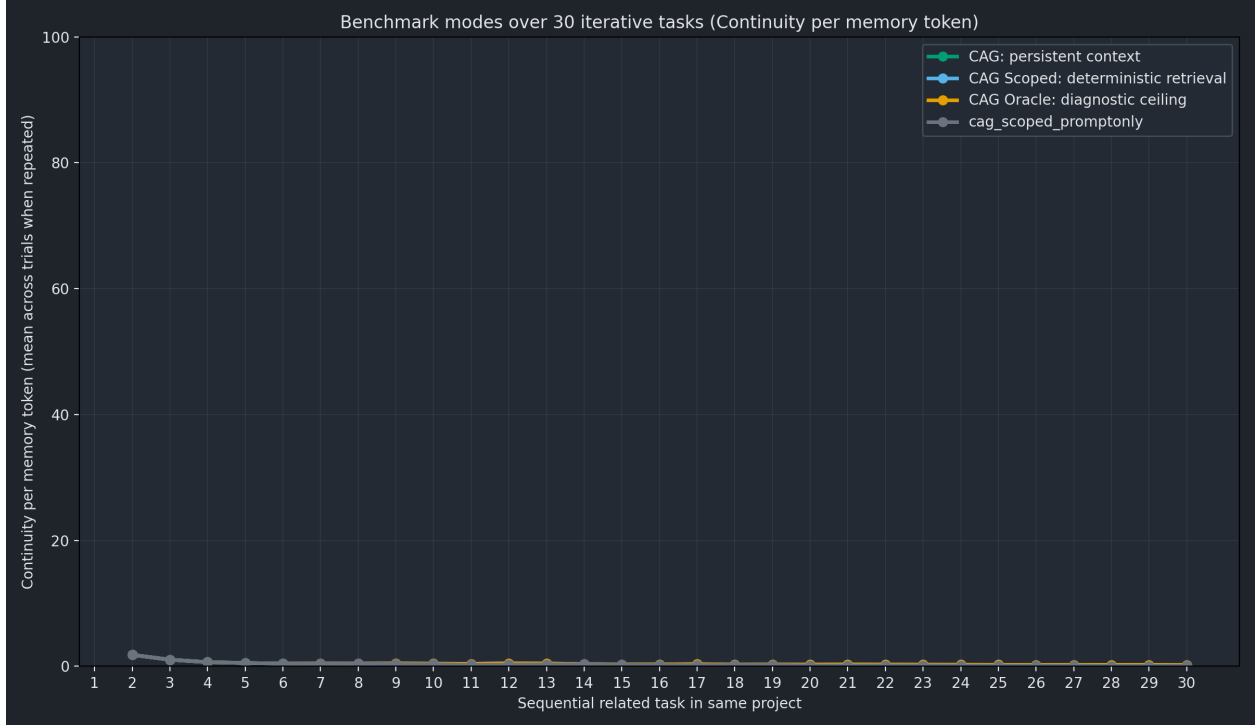


Figure 8: Continuity per memory token—efficiency of context spend. Scoped retrieval is roughly $3\times$ more efficient than base CAG in the late phase, sustaining similar continuity output at one-third the token cost.

7 Discussion

7.1 The value of accumulated project memory

The primary question CAG-Bench was designed to answer—does accumulated project memory meaningfully improve model performance over a long iterative task sequence?—receives a clear positive answer in this dataset. A $+18.9$ composite point advantage for 7B models ($n = 3$ trials), replicated directionally at 3B scale ($n = 10$) with comparable magnitude, is large enough to be unlikely to flip with formal testing, though we explicitly defer significance claims pending higher-trial-count replication. The advantage does not appear to be an artifact of prompt design: source retrieval recall is flat across modes, and checklist quality (which measures immediate task-specific content, not continuity) shows a smaller but consistent CAG advantage ($+9.3$ points).

The T30 result is particularly illustrative. At the final task—produce a complete implementation handoff for a 30-decision project—RAG achieves 9.2% continuity recall and a composite score of 13.0. CAG achieves 46.0% continuity recall and 30.7 composite. This is not a marginal improvement. It represents the difference between an assistant that can describe the project from its accumulated context and one that can only describe what appears in a single retrieved source document.

7.2 Why RAG and DAG fail on long sequences

RAG’s failure mode is structural. By task 20, the benchmark expects a model answer to reference 20 prior decisions, none of which are in any source document. Fresh retrieval cannot produce what was never written. RAG’s continuity recall of 5.9% in T21–T30 is effectively near-zero—the model

has no mechanism to access those decisions.

DAG’s failure is more subtle. The structured workflow prompt does not degrade faster than RAG (both reach approximately 6% continuity at T21–T30), but DAG consistently shows higher contradiction penalties (2.00 vs. 1.50 for RAG) and does not improve source evidence recall over RAG despite using a more structured prompt. Workflow structure without memory cannot compensate for the absence of project context.

7.3 Where CAG falls short

CAG’s late-phase degradation (from 70.7% continuity at T01–T10 to 34.6% at T21–T30) is meaningful. Two mechanisms drive this. First, **context saturation**: as the memory store grows, prompts become increasingly long (1,850 tokens of memory context at task 30 vs. 209 at task 2), and the model’s effective use of that context—as measured by memory usage rate—declines. Second, **concept density**: late tasks require recalling many more concepts simultaneously. Even when all 25 memory rows are present (100% memory recall), the model consistently references only about 35–46% of them in its answer.

The uptake gap is the most important unsolved problem identified by this benchmark. Improving retrieval—either through better scoring formulas or higher K —does not address the fundamental challenge that retrieved context must be actively incorporated into model outputs.

7.4 Retrieval matters more than the upper bound suggested

At $K = 5$, the two label-informed retrievers (`cag_scoped`, `cag_oracle_memory`) converge on the same memory recall and memory precision, indicating that further refinement of label-informed scoring at this capacity will not improve retrieval. However, the deployable label-free retriever (`cag_scoped_promptonly`) reaches only 60.9% memory recall versus 80.7% for the label-informed variants—a 19.8 percentage-point gap that widens to 31.6 points in T21–T30 (30.5% vs. 62.1%). Under deployable retrieval, retrieval scoring matters significantly. The “scoring formula isn’t the bottleneck” conclusion holds only when the formula has access to answer-key metadata; without it, retrieval becomes a meaningful bottleneck alongside uptake.

The capacity argument also holds for the deployable variant: $K = 5$ leaves no room to cover the 20+ continuity concepts required by late tasks, regardless of formula. A natural follow-up is to evaluate `cag_scoped_promptonly` at $K = 8$, $K = 12$, or higher, with diversity-aware selection to maximize concept coverage within the budget. We hypothesize the deployable retriever benefits more from capacity expansion than the label-informed one does—preliminary evidence in this benchmark suggests label-informed selection saturates around $K = 5$, while the deployable variant clearly does not.

7.5 Domain rules as persistent project constraints

A potentially important finding is CAG’s apparent advantage on domain rule recall: 56.4% for 7B CAG vs. 47.2% for RAG, and 59.8% vs. 40.3% at 3B. Domain rules in this benchmark encode project-specific constraints such as “never diagnose,” “no cloud sync,” “use humane training methods only,” and “implement local passcode.” These rules, once established in memory, persist across all subsequent tasks.

Caveat. Our scoring code mirrors source-evidence behavior into `domain_rule_recall` when one side is absent and vice versa, meaning some reported `domain_rule_recall` values include source-evidence scoring on tasks without domain-rule terms. The aggregate numbers reported here are therefore best understood as an evidence-split aggregate rather than a pure measurement of domain-rule application. A focused re-analysis filtered to tasks with non-empty `domain_rule_terms` is left to follow-on work; if the CAG advantage holds after that filter, it becomes a substantially stronger result. We flag this here so that readers do not over-interpret the current numbers.

7.6 Limitations

Several limitations constrain the generalizability of these findings.

Single task domain. All 30 tasks describe the same fictional application (`PawLedger`). Whether the CAG advantage generalizes to other domains—data pipelines, infrastructure engineering, API design—is an open question. The benchmark design is domain-agnostic in principle; we release it to enable replication with other task sequences.

Grounded memory only. Durable memory in this benchmark comes exclusively from task-defined ground truth, not from model output. Real-world CAG deployments would need to decide what model outputs are trustworthy enough to promote, a problem this benchmark does not address. The grounding constraint makes the benchmark cleaner but means it does not test the hardest real-world case.

Two local models. Experiments are limited to `qwen2.5-coder:3b-instruct` and `qwen2.5-coder:7b`. Larger models, models from other families, and API-hosted models may show different uptake rates and saturation behaviors. The relative ordering ($\text{CAG} > \text{RAG} \approx \text{DAG}$) is likely to hold but the magnitude of the advantage may vary.

Deterministic scoring. The benchmark measures concept presence in output text, not semantic correctness or code quality. A model that lists prior decisions verbatim as a preamble scores identically to one that integrates those decisions naturally into its implementation plan. Supplementing deterministic scoring with LLM-as-judge evaluation is left for future work.

8 Future Work and Research Agenda

The findings from this benchmark define a sequence of open questions that we intend to pursue as a series of follow-on studies. We outline them here both as a research agenda and as an invitation for replication by others using the released benchmark.

8.1 Higher-K and diversity-aware selection for the deployable retriever

The label-free `cag_scoped_promptonly` variant evaluated in Section 6.2 was tested at $K = 5$ to match the existing baseline. At that capacity, it reaches 60.9% memory recall overall and 30.5% in T21–T30. Whether a higher K (8, 12, or 20)—combined with diversity-aware selection that maximizes concept coverage rather than greedy top-K scoring—can close the gap to the label-informed upper bound is the most important open question this paper raises. We hypothesize that at $K = 12$ with diversity selection, deployable retrieval may approach 80% memory recall in T21–T30, but this is an empirical question we have not yet answered.

A complementary baseline worth implementing in the same study is **rag_memory_index**—fresh top-K retrieval from the accumulated memory store using only the task prompt/title/tags, with no scoring formula at all. This separates “having an accumulated store” from “selecting accumulated store with a tuned formula,” and would establish whether the CAG advantage requires the scoped scoring or merely the existence of a memory store.

8.2 Solving the uptake problem (near-term)

The dominant bottleneck identified in this work is not retrieval—it is uptake. Only 38–58% of retrieved concepts appear in model outputs, and this rate declines monotonically as task complexity grows. The benchmark as designed cannot distinguish between two possible causes: (a) the model attends to the memory context but fails to incorporate it into its output, or (b) the model’s attention mechanism deprioritizes memory context when the prompt is long. These produce the same observable symptom but require different interventions.

A follow-on study would instrument this at the attention level using open-weight models where attention patterns are accessible, directly measuring whether retrieved memory rows are attended to before scoring whether they appear in outputs. This would also evaluate whether diversity-aware selection (choosing K memory rows that cover maximum concept variety rather than the top-K by score) improves uptake by reducing redundancy in the context block.

8.3 Model-generated memory formation (near-term)

CAG-Bench deliberately uses grounded memory—only task-defined facts enter the store. This was the right design choice for a controlled first study, but it defers the harder real-world question: how should a system decide which model outputs are trustworthy enough to promote into durable memory?

A follow-on benchmark would allow model-generated memory and evaluate several validation strategies: (a) pure model output with no validation (the baseline, expected to pollute quickly with hallucinations); (b) model output filtered by overlap with source documents; (c) model output filtered by contradiction with existing memory; (d) model output validated against concept-group scoring before promotion. The grounded-memory runs reported here would serve as the oracle ceiling for this study.

8.4 Multi-domain task sequences (near-term)

All 30 tasks in CAG-Bench describe a single application domain (dog training software). Whether the measured advantage of CAG generalizes across domains is unknown. A data engineering pipeline, an API design sequence, an infrastructure provisioning workflow, and a compliance documentation project all have different prior-decision structures and different kinds of continuity dependencies. We plan to release additional 30-task sequences in at least three domains and evaluate whether the CAG advantage, the phase degradation pattern, and the uptake bottleneck are consistent across domains or domain-specific.

8.5 Larger models and API-hosted systems (near-term)

All experiments reported here use 3B and 7B local models via Ollama. We expect larger models to show better uptake rates—their stronger instruction-following may improve concept integration—but we also expect the fundamental retrieval capacity problem ($K = 5$ insufficient for late tasks requiring 20+ concepts) to persist regardless of model size, since it is a function of architecture constraints, not capability. A study comparing 7B, 13B, 34B, and 70B models on the same task sequence would establish whether uptake scales with model size or whether it requires architectural changes to improve.

8.6 Structured vs. unstructured memory representations (medium-term)

CAG-Bench stores memory as free-text summaries tagged with concept terms. An alternative representation—structured key-value stores, knowledge graph triples, or decision trees—might improve both retrieval precision and model uptake by presenting information in a format the model finds easier to reason over. A comparative study of memory representation formats on the same task sequence would quantify this trade-off. This connects directly to the SlopCodeBench finding that agents produce $2.2\times$ more verbose outputs than human developers: a structured memory format might reduce prompt bloat while improving concept coverage.

8.7 Scoring beyond concept presence (medium-term)

CAG-Bench’s deterministic scoring measures whether benchmark concepts appear in model outputs, not whether they are used correctly. A model that lists “SQLite, migration_version, soft_delete” as a preamble scores identically to one that integrates those decisions naturally into an implementation plan. This limitation was made explicit by an experiment in this work: a carry-forward prompt that instructed the model to list prior decisions verbatim dramatically improved measured continuity recall while degrading answer actionability as assessed by human review.

A follow-on study would develop an LLM-as-judge scoring layer calibrated against human developer ratings of answer actionability and decision fidelity. The calibration dataset would require human annotation of a sample of task answers across modes, producing human-validated scores that the judge model is trained to replicate. This would establish whether deterministic concept-presence scoring is a reliable proxy for practical usefulness—or systematically diverges from it in identifiable ways.

8.8 Longer task sequences and memory decay (longer-term)

Thirty tasks is a starting point. Software projects run for months or years, accumulating hundreds of decisions. A 100-task or 200-task sequence would answer: does CAG’s advantage continue to grow, plateau, or eventually invert as the memory store becomes too large to retrieve from effectively? This is the natural boundary condition for any accumulation-based approach. Related work on memory forgetting (MemoryBank, PersistBench) suggests that indiscriminate accumulation eventually becomes a liability; the task sequence length at which this crossover occurs has not been empirically established for software development contexts.

8.9 Human baseline (longer-term)

No human developer baseline exists for this benchmark. A developer given the same source documents and task sequence—without the ability to look at prior answers—would produce a natural lower bound. A developer who can review all prior outputs would produce a natural upper bound on what CAG could theoretically achieve. Establishing both would anchor the absolute scale of CAG-Bench scores in human terms and clarify whether the 48.5 composite we report for 7B CAG represents a practically useful assistant or a system still far below the human floor for professional software development.

9 Conclusion

Current evaluation practice for LLM-assisted software development treats tasks as independent. CAG-Bench demonstrates that this assumption breaks down over the course of a real project: RAG and DAG continuity recall collapses from approximately 32% in early tasks to approximately 6% in tasks 21–30, while a model with access to accumulated project memory holds a 34.6% floor at the same stage. The gap is not marginal—at task 30, a final implementation handoff, RAG produces answers with 9.2% continuity recall. CAG produces 46.0%. That is the difference between an assistant that can describe a project and one that cannot.

Three findings from this work warrant particular attention. First, the CAG advantage is **directionally consistent across model sizes** in our runs—the same relative ordering appears at 3B ($n = 10$) and 7B ($n = 3$) parameters—suggesting the effect is not specific to a particular scale, though we defer formal robustness claims to future replication. Second, an honest evaluation of label-free retrieval (`cag_scoped_promptonly`) reveals a substantial gap between the deployable retriever and the label-informed upper bound: approximately 20 percentage points of memory recall overall, widening to 32 points in late tasks. Retrieval is genuinely a bottleneck without label leakage. The deployable retriever nonetheless substantially outperforms RAG/DAG, confirming that the CAG advantage does not depend on label leakage—but the room for improvement under honest retrieval is larger than label-informed diagnostics suggested. Third, and most importantly: this paper introduces `memory_usage_rate` as a diagnostic metric that makes the uptake problem visible for the first time in this context. Even with favorable memory selection, models fail to reliably incorporate available memory into outputs—only 38–63% of selected memory concepts appear in final answers, and the rate declines as task complexity grows. That is a different problem from retrieval failure, and it requires different interventions.

The grounding contract—durable memory derived only from task-defined ground truth, never from raw model output—is both a design choice and a finding. Earlier development of this benchmark confirmed empirically that models will hallucinate plausible-sounding but fabricated architectural decisions (cloud sync, external APIs, incompatible data models) and that without a validation gate, those inventions compound across tasks until the memory store is corrupted. Grounded memory is not a limitation of this benchmark; it is the correct baseline from which model-generated memory formation should be evaluated as a separate, harder problem.

The benchmark, task suite, scoring code, figures, and committed paper assets (run configs, summary CSVs, promotion audit logs, sampled raw outputs) are released under AGPL-3.0-or-later at <https://github.com/GuideboardLabs/cag-bench>. Full per-trial raw outputs are available as a GitHub Release artifact in the same repository. The central question this work leaves open—and the question

we consider most important for the field—is not how to retrieve memory better. It is how to make models use the memory they already have.

References

- [1] Es, S., James, J., Espinosa-Anke, L., & Schockaert, S. (2023). RAGAS: Automated Evaluation of Retrieval Augmented Generation. *arXiv preprint arXiv:2309.15217*.
- [2] Truera (2023). TruLens: Evaluation and Tracking for LLM Experiments. <https://github.com/truera/trulens>.
- [3] Bai, Y., Lv, X., Zhang, J., Lyu, H., Tang, J., Huang, Z., Du, Z., Liu, X., Zeng, A., Hou, L., Dong, Y., Tang, J., & Li, J. (2023). LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. *arXiv preprint arXiv:2308.14508*.
- [4] Wu, D., Wang, H., Yu, W., Zhang, Y., Chang, K.-W., & Yu, D. (2024). LongMemEval: Benchmarking Chat Assistants on Long-Term Interactive Memory. *ICLR 2025*. arXiv:2410.10813.
- [5] Hu, Y., Wang, Y., & McAuley, J. (2025). Evaluating Memory in LLM Agents via Incremental Multi-Turn Interactions. *ICLR 2026*. arXiv:2507.05257.
- [6] Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., & Levy, O. (2022). SCROLLS: Standardized Comparison Over Long Language Sequences. *EMNLP 2022*.
- [7] Zhang, X., Chen, Y., Hu, S., et al. (2024). ∞ BENCH: Extending Long Context Evaluation Beyond 100K Tokens. *arXiv preprint arXiv:2402.13718*.
- [8] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560*.
- [9] Park, J. S., O’Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). Generative Agents: Interactive Simulacra of Human Behavior. *UIST 2023*.
- [10] Zhong, W., Guo, L., Gao, Q., Ye, H., & Wang, Y. (2023). MemoryBank: Enhancing Large Language Models with Long-Term Memory. *arXiv preprint arXiv:2305.10250*.
- [11] PersistBench: When Should Long-Term Memories Be Forgotten by LLMs? (2026). *arXiv preprint arXiv:2602.01146*.
- [12] Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., & Narasimhan, K. (2023). SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770*.
- [13] SWE-EVO: Benchmarking Coding Agents in Long-Horizon Software Evolution Scenarios (2024). *arXiv preprint arXiv:2512.18470*.
- [14] SlopCodeBench: Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks (2026). *arXiv preprint arXiv:2603.24755*.
- [15] Chen, M., Tworek, J., Jun, H., Yuan, Q., et al. (2021). Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*.

- [16] Li, B., Wu, W., Tang, Z., Shi, L., Yang, J., et al. (2024). DevBench: A Comprehensive Benchmark for Software Development. *arXiv preprint arXiv:2403.08604*.

A Full Metric Grids

B Composite Score Weights

```

checklist_quality:      0.34
source_evidence_recall: 0.12
domain_rule_recall:     0.12
continuity_recall:      0.40
token_efficiency:       0.01
latency_efficiency:     0.01
minus contradiction_penalty

```

C Retrieval Formula (`cag_scoped`)

```

score = 3.0 * concept_overlap
       + 1.0 * tag_overlap
       + 0.5 * task_text_overlap (Jaccard)
       + 1.0 * recency_weight (idx / store_size)

```

`concept_overlap` counts the number of `continuity_terms` groups (from the task definition) that the memory row matches. Because `continuity_terms` is also the answer-key field used in scoring, the current `cag_scoped` formula is **label-informed** and should be treated as a diagnostic upper bound rather than a deployable retriever. A label-free variant using only task title, prompt, tags, and source-document overlap is described in Section 8.

D Memory Schema

```

{
  "memory_id": "M01_00003",
  "task_id": "T03",
  "task_title": "Define local persistence",
  "scope": "project",
  "memory_type": "decision",
  "text": "Persistence uses SQLite with UTC timestamps,
          migration_version, soft_delete, and
          deterministic IDs.",
  "tags": ["persistence", "storage", "sqlite"],
  "promoted_terms": ["SQLite", "migration_version",
                    "soft_delete", "deterministic IDs"]
}

```

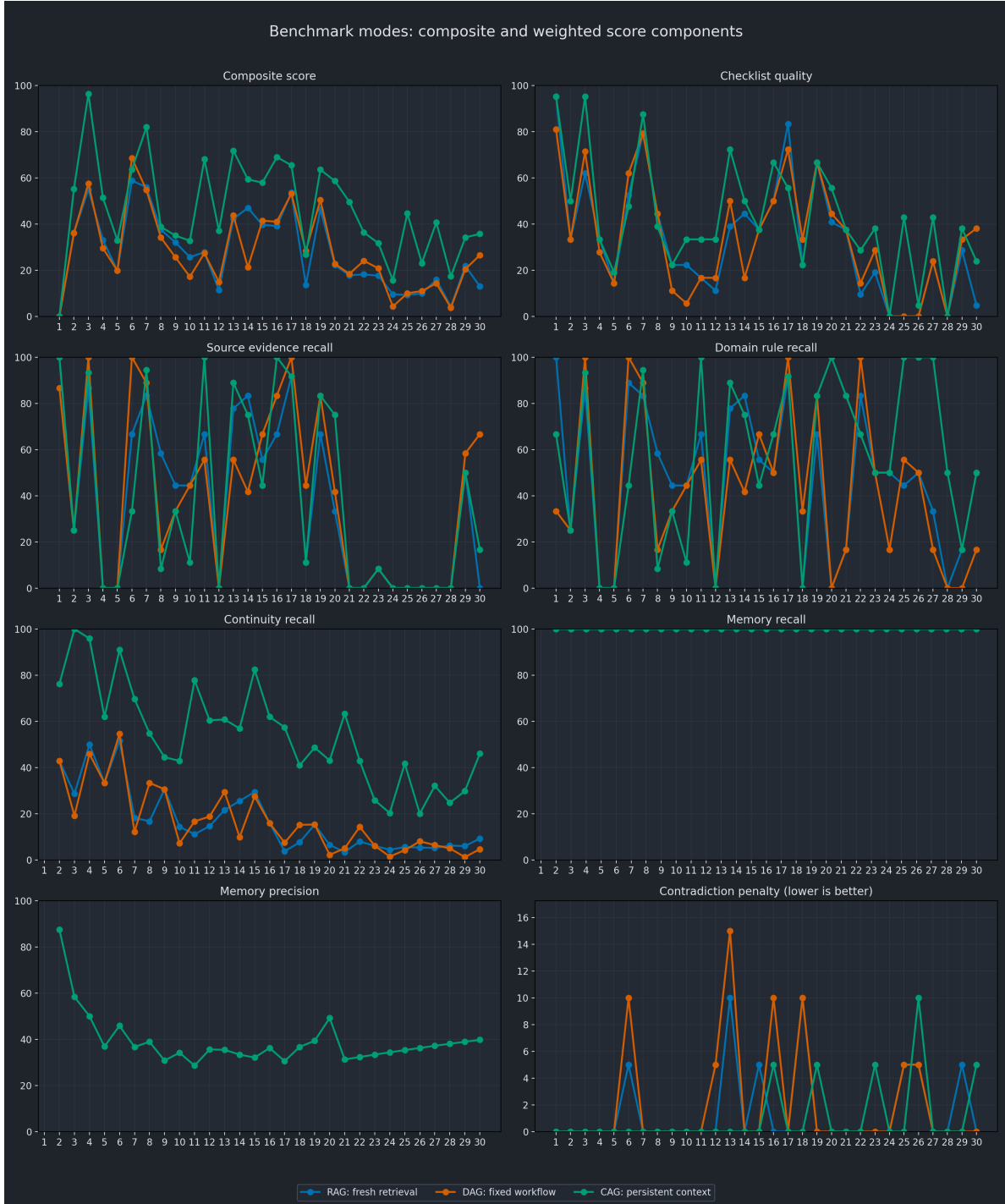


Figure 9: All metrics grid—baseline run (qwen2.5-coder:7b, RAG / DAG / CAG, 30 tasks, 3 trials). Composite score, checklist quality, evidence recall, continuity recall, memory recall, memory precision, and contradiction penalty across all 30 tasks.



Figure 10: All metrics grid—CAG ablation run (qwen2.5-coder:7b, four CAG variants including cag_scoped_promptonly, 30 tasks, 3 trials). Same metric set across the four variants, showing the leakage gap, the scoped/oracle convergence, and base CAG token cost.

E Run Configuration Reference

Table 9: Fixed run configuration parameters.

Parameter	Value
Temperature	0.1
Context window	8192 tokens
Source retrieval K	3
Memory top- K (scoped/oracle)	5
Memory cap (base CAG)	25
Ollama server	localhost:11434