PROJECT PLAN

Cookbook Memory

AI Agent Memory Harness — persistent memory for long-running coding agents

Codename **Cookbook Memory** · Team of 4 (Keith · Ken · Brent · Scott B.) · Two-week sprint · Five public benchmarks

Repository: github.com/kenhuangus/agent-memory-harness · Live: kenhuangus.github.io/agent-memory-harness

1 The problem

There is **no standard, model-agnostic layer** that decides **what to remember**, **where to store it**, **how to retrieve it fast**, and **how to keep it consistent over time** for long-running agents.

DECISION	WHAT IT MEANS
What	signal vs. noise
Where	the right store
Fast	no context flood
Consistent	dedup, no conflicts

2 Hypothesis & success criteria

> **Hypothesis.** With the memory harness, **Haiku can close the gap to Opus 4.8** (which runs without memory) on established memory benchmarks.

Success criterion. On **at least two of the five** benchmarks, *Haiku + harness* beats the *Opus 4.8 no-memory* baseline across the four metrics, **without** blowing the efficiency budget (memory adds **< ~10%** context-token overhead).

METRIC	DEFINITION
Recency	Is the freshest relevant memory ranked first?
Efficiency	Tokens per retrieval; target under ~10% overhead.
Relevancy	Retrieved items actually relate to the query.
Accuracy	Task correctness, memory on vs. off.

3 Technical approach

A **pluggable memory harness**: four modules over three indexed storage backends, wired into the open-source **OpenCode** agent and measured on public benchmarks.

Four modules

- **Persistence layer (write).** Decides what/when/where to save; tags each item (timestamp, relevancy, session); versioning + dedup hints.
- **Intelligent router (dispatch).** Classifies each query and hits the *single best* backend. Rule-based, then learned.
- **Retrieval orchestrator (read).** Unified API: route, rank by `recency × relevancy`, de-dup, return a tight context.
- **Dreaming component (async).** Runs **while other agents sleep**: de-dup, resolve contradictions, session governance (must-know / must-do / blacklist), retention + pruning.

Three storage backends (each indexed)

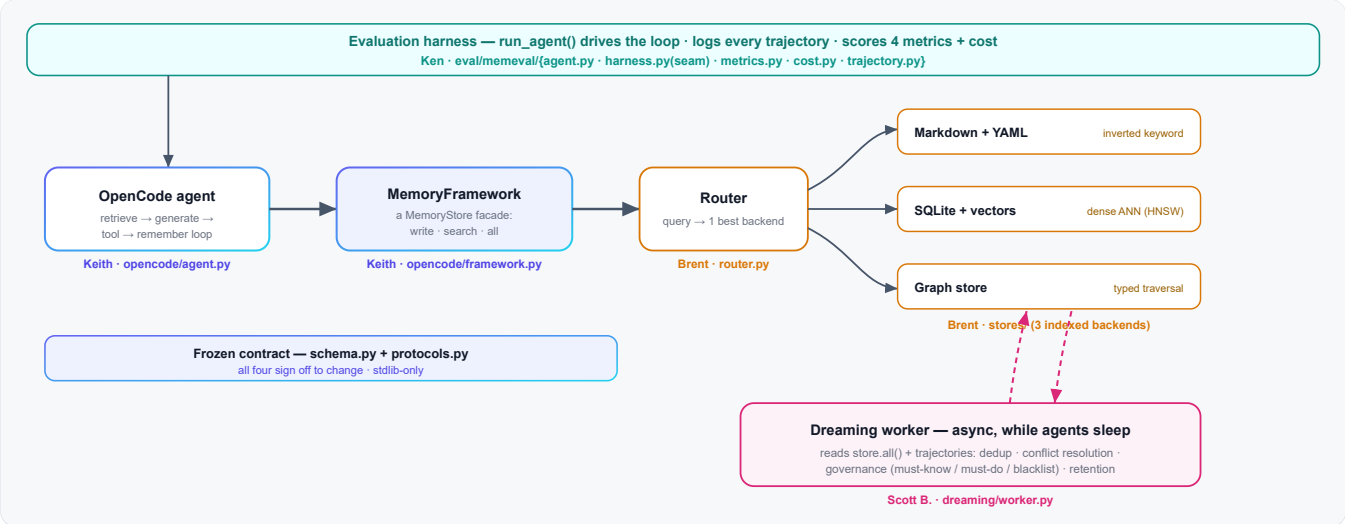
BACKEND	BEST FOR	INDEX
Markdown + YAML	Literal recall	Inverted keyword map
SQLite + vectors	Semantic search	Dense ANN (HNSW / FAISS)
Graph store	Relationships & conflicts	Typed traversal

Evaluation

Public benchmarks only — no home-grown evals. Per benchmark, a controlled grid runs Haiku + harness vs. Haiku, Opus 4.8 and Sonnet — prompts and task order fixed, every trajectory logged.

4 System design (final architecture)

The design is **contract-first**: two stdlib-only files — `schema.py` (data model) and `protocols.py` (interfaces) — are **frozen on Day 3**, and every component is built against those `typing.Protocol` seams, never against each other's code. That is what lets four people work in parallel without collisions. The full technical contract lives in `architecture.md`.



Data flow: the eval harness (Ken) drives OpenCode (Keith), which reads/writes through the MemoryFramework (Keith) → Router (Brent) → one of three indexed backends (Brent); the Dreaming worker (Scott B.) consolidates the store asynchronously. Every box implements a frozen `protocols.py` seam.

Frozen public interfaces (the seams)

PROTOCOL	KEY METHODS	IMPLEMENTED BY
MemoryStore	`write(item)` · `get(id)` · `search(query, k) → RetrievedItem[]` · `all()`	Brent's backends + Keith's `MemoryFramework`
AgentAdapter	`solve(task, ctx)` — the multi-step loop	Keith's `OpenCodeAgent`
ModelAdapter	`generate(prompt) → (text, tok_in, tok_out)`	Keith's `models.py` (+ EchoModel)
Loader	`load(path_or_id) → Task[]`	Ken's `loaders/` (five benchmarks)

How a request flows

- **Drive:** `run_agent(benchmark, OpenCodeAgent, memory=True, store=MemoryFramework)` hands the agent one task at a time and a per-task `AgentContext` (centralizes cost + trajectory + grading).
- **Retrieve:** the agent calls `ctx.retrieve(query)` → framework asks the **Router** for the one best backend → that backend's `search` returns ranked, deduped hits (each carrying its token count).
- **Generate:** `ctx.generate(prompt)` calls the model, charges cost, logs a step.
- **Remember:** `ctx.remember(fact)` writes back to the shared store, so a later task in the same group can retrieve it.
- **Dream (async):** between sessions the **Dreaming worker** reads `store.all()` + trajectories and consolidates — dedup, conflict resolution, governance, pruning.

Module → directory → owner

PATH	ROLE	OWNER
`schema.py` , `protocols.py`	Frozen contract (data model + seams)	**All four**
`opencode/`	OpenCode agent + MemoryFramework (store/retrieve + dreaming wiring)	**Keith**
`harness.py` , `models.py` , `cli.py`	Single-shot harness · model adapters · CLI	**Keith**
`stores/` , `router.py`	Three indexed backends + the dispatch router	**Brent**
`dreaming/`	Async consolidation engine	**Scott B.**
`loaders/` , `metrics.py` , `cost.py` , `trajectory.py` , `agent.py` , `results.py`	Eval infrastructure + the AgentAdapter seam	**Ken**

> **Load-bearing invariants** (the contract tests can't catch): `search` returns items sorted by descending score with `rank` and `tokens` set (the efficiency metric needs them); retrieval respects the query's "as-of" time — never surface a memory from the future; the offline path imports **no** third-party package at module top level.

5 Scope

In scope

- The memory harness: all four modules, end to end.
- Three indexed storage adapters behind one interface.
- The intelligent router (rule-based; learned-classifier hook).
- The async dreaming component: dedup, conflict resolution, governance, retention.
- OpenCode integration for memory write/read on each step.
- Eval harness over five benchmarks; baselines for Haiku, Opus 4.8, Sonnet.
- Metric definitions, a reproducible protocol, results dashboard.

Out of scope (non-goals)

- Building our own benchmark or dataset — we use public ones.
- Training or fine-tuning any model.
- Production hardening: multi-tenant infra, auth, SLAs, polished UI.
- Distributed / scaled deployment — single-node is enough for the sprint.
- Validating non-coding agents — design generalizes, but not tested now.

6 Team & ownership — who owns what

Four parallel workstreams, locked behind a shared storage interface **frozen on Day 3**.

The team: P1 Keith (harness, OpenCode) · **P2 Ken** (benchmark) · **P3 Brent** (store, retrieve) · **P4 Scott B.** (dreaming).

AREA / DELIVERABLE	OWNER	SUPPORTING
Storage interface & memory-item schema	**P1 · Keith**	P3
Persistence layer (write path, tagging, versioning)	**P1 · Keith**	—
Retrieval orchestrator (rank · dedup · return)	**P1 · Keith**	P3
OpenCode integration (write/read on each step)	**P1 · Keith**	—
Three storage backends + per-backend indexes	**P3 · Brent**	P1
Intelligent router (rules → learned)	**P3 · Brent**	P1
Backend performance testing	**P3 · Brent**	P2
Dreaming worker (dedup, conflict, governance, retention)	**P4 · Scott B.**	P1, P3
Memory semantics ("what is good memory")	**P4 · Scott B.**	P2
Datasets, loaders & trajectory logging	**P2 · Ken**	P4
Metric defs, shared run harness & cost gates	**P2 · Ken**	P4
Results + cost dashboard, stats, aggregation	**P2 · Ken**	—
Run SWE-Bench-CL eval (own key)	**P1 · Keith**	—
Run LongMemEval eval (own key)	**P2 · Ken**	—
Run SWE-ContextBench eval (own key)	**P3 · Brent**	—
Run MemoryAgentBench eval (own key)	**P4 · Scott B.**	—
Run ContextBench eval (own key)	**P3 · Brent**	—
Final end-to-end integration	**All**	—

- **P1 · Keith — Harness Architecture & OpenCode Integration.** Critical path — the contracts everyone builds against.
- **P2 · Ken — Evaluation Infrastructure & Coordination.** Shared runner, metrics, dashboard — and captains one benchmark.
- **P3 · Brent — Storage Implementation & Router.** Three fast backends + the dispatch layer; captains the two retrieval benchmarks.
- **P4 · Scott B. — Dreaming Component & Memory Governance.** The offline engine that keeps memory clean.

Dividing the eval runs (API cost & time)

~20 runs (5 benchmarks × 4 configs) is too much cost and time for one person on one key. Each teammate **captains** the benchmark(s) that stress their own component, on their **own API budget**. Any collaborator can also launch a run from the **GitHub Actions** tab (free offline by default).

BENCHMARK	CAPTAIN	WHY THEM
SWE-Bench-CL	**Keith (P1)**	Drives the coding agent end-to-end — his OpenCode wheelhouse.
LongMemEval	**Ken (P2)**	Eval lead; recency / temporal reasoning.
SWE-ContextBench	**Brent (P3)**	Cross-task context reuse exercises his retrieval + router.
MemoryAgentBench	**Scott B. (P4)**	Tests conflict resolution — his dreaming component.
ContextBench	**Brent (P3)**	Retrieval-quality (gold contexts) — the retrieval/router he owns.

Ken owns the shared runner — `run(benchmark, model, memory) → metrics` — and aggregates all results. Controls: sharded keys (one per captain); cheapest-first + early-exit (Opus last, confirm-only); dev-slice → full; cache + resume; hard per-benchmark budget gates; no-memory baselines in week 1.

7 Two-week timeline

Person 1 · Keith — Harness Architecture & OpenCode Integration

- **Week 1:** Three-module abstraction & storage interface. Persistence layer (write, versioning, tagging). Instrument OpenCode to write memory each step.
- **Week 2:** Retrieval orchestrator (rank → dedup → return); wire in Brent's router; end-to-end task. **Captain the SWE-Bench-CL runs.**

Person 2 · Ken — Evaluation Infrastructure & Coordination

- **Week 1:** Parse all five datasets; loaders + trajectory logging (with Scott B.). Define the metrics; build the shared run harness + cost tracker.
- **Week 2: Captain the LongMemEval runs.** Aggregate every captain's results; stats + results & cost dashboard; document the protocol.

Person 3 · Brent — Storage Implementation & Router

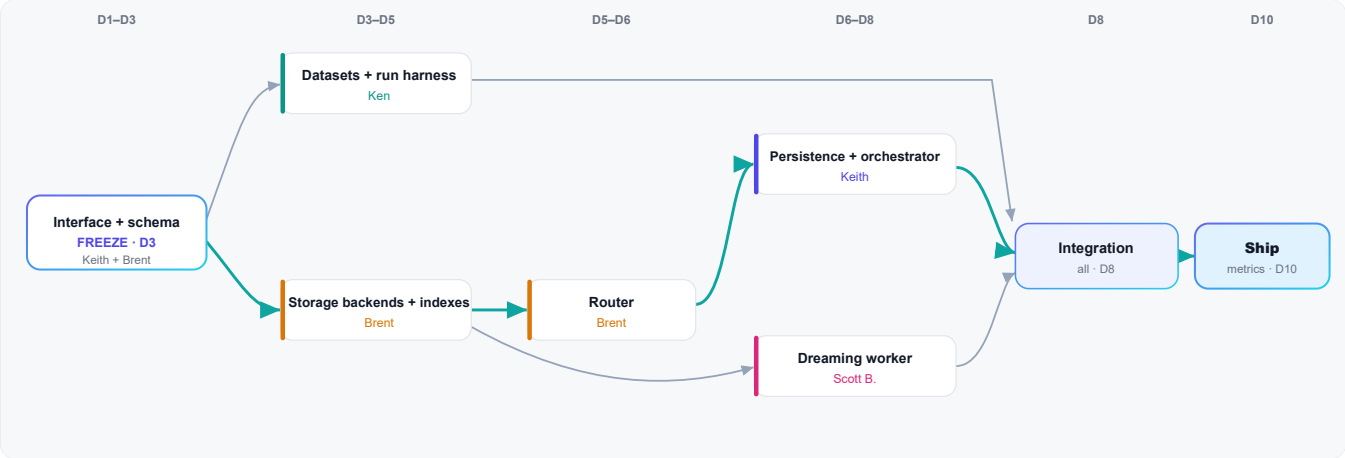
- **Week 1:** SQLite + vector pipeline (HNSW/FAISS). Markdown store + inverted index. Graph store (Neo4j) + traversal index.
- **Week 2:** Adapters behind Keith's interface; implement the router; performance-test each backend. **Captain the SWE-ContextBench & ContextBench runs.**

Person 4 · Scott B. — Dreaming Component & Memory Governance

- **Week 1:** Dedup (exact / semantic / near-dup). Conflict detection + reconciliation. Session-filter & blacklist. Co-build trajectory logging with Ken.
- **Week 2:** Async scheduler that runs while agents sleep; must-know / must-do extraction; retention & pruning; observability; integration with Keith & Brent. **Captain the MemoryAgentBench runs.**

8 Dependency map & critical path

The Day-3 interface freeze unblocks every stream; the bold chain is the critical path to ship.



Teal = critical path (freeze → backends → router → orchestrator → integration → ship). Grey = supporting dependencies.

- **Keith + Brent (D1–D3):** co-author and freeze the storage interface + schema, so persistence and adapters share one contract.
- **Brent → Keith:** the router lands before the orchestrator can route in week 2.
- **Ken → all (by D5):** datasets, logging and the shared run harness ready, so baselines can start ~D4–D6.
- **Keith + Brent → Scott B.:** the dreaming worker integrates once real storage exists (D8+).
- **Sharded keys:** each captain on a separate API budget — baselines week 1, treatment week 2.

> **Top risk is semantic, not structural:** defining *what counts as "good memory."* Ken and Scott B. must align on memory semantics in week 1, or the harness stores everything and retrieves nothing useful.

9 Milestones

DAY	MILESTONE
D3	**Contract freeze** — storage interface + memory-item schema locked; metric definitions agreed.
D5	**End of week 1** — three backends read/write; shared run harness + loaders ready; no-memory baselines started on sharded keys.
D8	**Integration start** — router + orchestrator connected; dreaming worker running against real stores.
D10	**Ship** — all benchmark shards complete (Haiku + harness); four metrics aggregated vs. baselines.

10 Benchmarks & metrics

BENCHMARK	PRIMARY METRICS	REFERENCE
MemoryAgentBench	Relevancy, Accuracy, conflict resolution	arXiv 2507.05257
LongMemEval	Recency, Accuracy, temporal reasoning	arXiv 2410.10813
SWE-ContextBench	Efficiency, Relevancy, Accuracy	arXiv 2602.08316
SWE-Bench-CL	Relevancy, Accuracy, continual learning	arXiv 2507.00014
ContextBench	Relevancy, Efficiency, retrieval recall/precision	arXiv 2602.05892

Full descriptions, dataset links and the metric-mapping matrix live on the project site's **Benchmarks** page.

11 Collaboration & deliverables

Four developers, one repo, no conflicts: **prd.md / architecture.md / plan.md** are the contracts; GitHub Flow + CODEOWNERS + branch protection enforce one-owner-per-directory and all-owner sign-off to change a frozen interface. See `CONTRIBUTING.md` and the site's **Collaborate** page.

- A model-agnostic memory harness (4 modules) integrated with OpenCode.
- Three indexed storage backends behind one interface.
- The async dreaming component with governance & pruning.
- A reproducible eval harness (shared runner + cost gates) + baselines on five benchmarks.
- A results scoreboard testing the hypothesis (logged via `results.json` / GitHub Actions).
- The project site (GitHub Pages) documenting all of the above.