ChatGPT

ANVCS and CRDT: Introducing a Shared “Memory” Space

ANVCS already uses **CRDT-based rooms** for conflict-free, real-time editing and snapshots (“commits”) into a Git-compatible DAG [1] [2]. We propose adding a parallel **“memory room”**: a CRDT-backed, shared state that all agents can read/write. In practice this means treating a special memory store (e.g. one or more JSON/Markdown files in the repo) as a live CRDT document. Agents collaboratively update this memory via CRDT ops, which converge automatically (no conflicts). When an agent reaches a semantic milestone (like completing a task or hitting a goal), it **snapshots the memory CRDT** into a new Git commit alongside code changes – just as current ANVCS does for code [1] [2]. This design leverages ANVCS’s existing model: CRDT operations flow into a DAG commit stream, preserving **immutability and history**. New agents pulling the repo simply load the latest memory file (the “current” view) and can also trace its evolution via Git diffs, ensuring every agent benefits from earlier work.

[1] [3] In this model, memory is part of the repo. For example, a file like `memory.json` (or a folder of topical memory notes) is managed by the CRDT engine. Agents add facts or context to it via high-level operations (e.g. “set(person:Alice:age, 30)”), and those operations propagate to all peers. Because the CRDT ensures convergence, any agents (even offline) merging later will see a consistent combined memory [1] [3]. This resembles open-source **memX**, which uses a CRDT-like layer for shared agent memory [3]. Unlike separate vector stores, the memory stays human-readable and versioned in Git. Agents can query it directly (using simple search or BM25) to extend their context beyond the LLM prompt window, while heavy semantic search can optionally overlay (we keep Git as the source of *present truth* and history, per DiffMem’s advice [4] [5]).

Architecture & Workflow: The ANVCS stack would gain a **Memory Manager** alongside the existing CRDT collaboration core. This manager exposes an in-repo memory CRDT “room” (e.g. via the ai-vcs CLI or IDE integration). Agents **read** from memory at startup (pulling the latest repo) and **listen** for live updates via the CRDT protocol. They **write** by emitting CRDT ops, which the local or cloud relay merges. When tasks finish, an agent invokes `ai-vcs commit` (as normal) to snapshot *both* code and memory state. Under the hood this “dual-write” appends a new DAG node with two trees: code and memory (the memory file content is captured in that commit) [1] [2]. Each commit carries AI metadata as before (prompts, model versions, etc.), and memory changes appear as part of the tree diff. The design still uses Git operations (in .git/objects and .git/refs) so any Git-compatible workflow (PRs, CI checks) works seamlessly. Importantly, **merging memory is automatic** thanks to CRDT; semantic merging is only needed when multiple branches concurrently updated code. Memory, treated like any other CRDT document, never “loses” data due to a merge conflict.

Leveraging Git’s Versioning: Storing memory in Git offers auditability and continuity [4] [5]. By default agents work with the “present” memory (a small file or folder) to build prompts. The Git history preserves *how* the memory changed over time. For instance, an agent updating a person’s age will overwrite the current value (e.g. `Alice.age = 30`), but the previous value (e.g. 29) remains in past commits. If later needed, an agent or human can inspect the commit history (or use `git blame`) to see the change context

[4]. This addresses limited LLM context: agents always get the latest compact memory (avoiding overwhelming prompts), but can also reconstruct earlier facts by reading historical versions if needed. In practice we follow the "DiffMem" pattern: keep memory documents small and use Git to trace history [4]. Agents might apply lightweight indexing (like a BM25 index on the current memory files) for fast retrieval of relevant facts. If memory grows large, we can prune older facts into an archive branch (e.g. `.ai/history/`), as suggested by DiffMem [4], to keep the working memory lean while preserving full history.

Multi-Agent Dynamics: CRDT ensures *concurrency* is safe. Two agents can update memory at once (say one adds a goal, another corrects a detail) without locking or conflicts: the CRDT merge rule merges their ops deterministically. Agents treat the repo as a "shared contract" [5]: they can even open pull requests for manual review of significant memory changes, just like code changes. For critical coordination, agents can tag memory updates with provenance (added in commit metadata), so any human or agent can audit why a fact changed. We enforce policies (via small-commit, clear-message guidelines) to avoid "stomping" in multi-agent races [6]. In effect, memory commits become another stream in the DAG graph. Because ANVCS uses a DAG, multiple branches of memory can exist if agents diverge offline; semantic merging handles code, but memory CRDT auto-merges so we avoid dangling divergent memory states. This design readily scales from single-repo to multi-repo. In the future, we could extend memory scope to the org level by a "meta-memory" repo: using ANVCS's cross-repo commit grouping [7], one could commit the combined state or links of multiple repo memories to a central place, giving agents a global context when needed.

Offline and Adoption: This approach remains **fully offline-capable**. ANVCS is local-first by design [8], so agents can use memory even without network. In offline mode agents accumulate CRDT ops in the memory room and commit locally to Git just as usual [8]. Once reconnected, ops sync with peers automatically. No extra "offline memory" mode is needed – memory simply behaves as another CRDT-tracked file. By using standard technologies (Git, JSON/Markdown, Yjs/Automerge-style CRDTs) we aim for wide adoption. Developers and agents use familiar Git workflows (push, pull, PRs) to manage memory commits. As R. Thompson notes, teams can "treat the repo like a shared contract" [5]. This lowers the barrier: any Git host (GitHub, GitLab) can store the memory, and any tool that reads the repo can access it. Importantly, no proprietary vector DB is required; the repo itself is the source of truth. Teams can still augment with semantic search if needed, but the core memory remains portable and human-inspectable in Git.

Summary: In summary, we enhance ANVCS by adding a *per-repo memory room* – a CRDT-synchronized state stored as repository files. Agents update this memory live and snapshot it into commits when it matters [1] [2]. This persistent shared memory lets new agents immediately benefit from past work (avoiding cold-start context) and greatly extends effective context beyond the LLM's window. All while preserving Git's offline, immutable history and fitting into existing Git/GitHub workflows [8] [5].

Sources: Key ideas come from ANVCS's own design (CRDT rooms and commit DAG [1] [2]) and from recent approaches like DiffMem (using Git for agent memory [4]) and memX (real-time CRDT memory for agents [3]). These principles guide a solution that is both powerful and widely adoptable.

---

[1] [2] anvcs_ai_native_version_control_system_problem_design_vision_v_0.md
file://file-SoKnu21Q6Epd1ZepAnkNdq

3 GitHub - MehulG/memX: A real-time shared memory layer for multi-agent LLM systems.
https://github.com/MehulG/memX

4 5 6 DiffMem: Why a Git Repo Might Be the Most Honest Memory Your AI Can Have | by R. Thompson (PhD) | Data Science Collective | Medium
https://medium.com/data-science-collective/diffmem-why-a-git-repo-might-be-the-most-honest-memory-your-ai-can-have-1951203912ed

7 anvcs_multi_agent_extensions_live_observability_v_0.md
file://file-NGaQKNTKv9MfJ9ZKxmdC8v

8 CRDT-Disconnected (Offline) Mode.pdf
file://file-7JXhxK5agU8pLSP5zFzxzB