

# ANVCS — Multi-Agent Extensions & Live Observability (v0.3 Addendum)

**Purpose.** Extend the ANVCS spec with research-backed improvements focused on: (1) additional Git shortcomings for AI multi-agent workflows, (2) optimizations to the current CRDT + Git hybrid, (3) secure deployment topologies (local, Codespaces/remote, P2P, relay), and (4) real-time human visibility (live viewing, playback, and recording of repository evolution).

---

## 1) Additional Git Shortcomings in the AI Multi-Agent Era

1. **History opacity & local state**
  2. Reflogs, stashes, and soft resets are **local-only**; they don't propagate or audit well across teams or agents.
  3. Automated agents performing non-linear history ops (amend/reset) can create gaps in the shared narrative.
  4. **Commit churn & review fatigue**
  5. High-throughput agents emit thousands of small commits. Human reviewers can't parse intent; meaningful milestones are lost in noise.
  6. **Merge/cherry-pick fragility at scale**
  7. Line-based merges fail under overlapping automated edits; cherry-picks become brittle when the codebase shifts rapidly.
  8. **Context-free blame**
  9. `git blame` lacks prompt/model/agent context; accountability and debugging suffer when code is AI-authored or AI-refactored.
  10. **Cross-repo drift**
  11. Git has no native atomic multi-repo commit. Multi-service edits (API + consumer) easily desynchronize.
  12. **Non-text asset friction**
  13. Models/datasets/notebooks/images are awkward in vanilla Git; reproducibility requires exact artifact pinning and versioned storage.
-

## 2) Optimizing the CRDT + Git Hybrid

1. **Operation-journal first; Git mirror second**
2. Treat per-file CRDT ops as the canonical stream. Periodically snapshot to a Git commit (dual-write).  
Benefits: conflict elimination, fine-grained playback, efficient P2P sync.
3. **DAG as the truth; branches as pointers**
4. Keep a directed acyclic commit graph. Branchesstreams are named refs to nodes. This removes rebasing pressure and encodes parallelism explicitly.
5. **Rich provenance attached to commits**
6. Per-commit metadata: prompt, model version, agent id, tests, coverage, environment, artifact pins  
`{key, version_id, sha256}`.
7. Store redundantly in `.ai/meta/<commit>.json` and in Git (notes or structured commit message block).
8. **Semantic merge at commit boundaries**
9. Use AST/test-guided merge (optional LLM proposals) only when creating merge nodes in the DAG.  
Live CRDT ops remain light/fast.
10. **Cross-repo atomic groups**
11. Represent multi-repo updates as a **meta-commit** that pins child commit IDs from multiple repos.  
Push a signed manifest to each repo; CI validates group consistency.
12. **Churn control**
13. Commit policy knobs: per-task default, time-boxed, CI-validated. Collapse trivial cosmetic changes into a formatting lane or batch window.
14. **Modality-aware diffs**
15. Built-in merge drivers for JSON/YAML (schema-aware), notebooks (cell graph), proto/IDLs (symbol diffs), media manifests (metadata diffs), and binary pass-through.

---

## 3) Real-Time Human Visibility & Playback

1. **Live CRDT diff viewer**

2. VS Code/Monaco panel that overlays agent cursors and streaming edits (per-agent colors). Toggle lanes (feature/refactor/format) to reduce noise.

### 3. Commit preview mode

4. Two-step: preview → confirm. Show semantic summary (changed symbols/APIs), risk flags, and test dry-run results before creating a DAG node.

### 5. Time-lapse & playback

6. Persist timestamped CRDT ops. Provide a timeline scrubber to replay repository evolution. “Time-warp checkout” can materialize the worktree at any op index.

### 7. Live DAG view

8. Animated graph of new nodes/merges as agents finish tasks. Filters by agent, path, test status, or lane.

### 9. Screen recording vs code-level replay -

10. **Pixel recording**: optional ffmpeg-based capture during sessions for human-friendly demos.

11. **Code-level replay**: deterministic reconstruction from CRDT ops + commit snapshots; smaller, exact, machine-diffable.

### 12. Activity dashboard

13. Per-agent status, latest ops/sec, pending snapshots, failing tests, and artifact pins. Notifications when attention is needed.

---

## 4) Do I Need a CRDT Server?

**Short answer:** Not always.

**Modes:** 1. **Local-only (single user)**: No server; CRDT state is in-process. You can still snapshot to Git and pin artifacts. 2. **Ephemeral local relay (recommended default)**: A lightweight process inside your devcontainer (or laptop) acts as a room broker while you collaborate; it exits when you do. 3. **P2P WebRTC**: Direct, encrypted peer-to-peer between your machine and remote agents. Needs STUN; TURN used if strict NAT. 4. **Relay/broker service**: A modest WebSocket/gRPC service (could be self-hosted or serverless) brokers CRDT ops when P2P is blocked. End-to-end encryption keeps payload opaque to the relay. 5. **Fully centralized**: For enterprises, a managed cluster provides rooms at scale with auth, quotas, and observability.

**Decision guide:** - Solo / offline → *Local-only*. - Small team / Codespaces → *Ephemeral relay in devcontainer*. - Cross-org or NAT-constrained → *P2P with TURN fallback*. - Large org governance → *Managed relay cluster*.

---

## 5) Secure Remote Connectivity (Local $\rightleftarrows$ Cloud Agents)

- **Transports:** WebSocket (TLS), WebRTC (DTLS/SRTP), or gRPC (mTLS).
  - **AuthN/Z:** JWT/OIDC, SSH-sig, or mTLS client certs. Scopes: `read`, `write`, `commit`, `merge`; path-level allow/deny.
  - **Key management:** Per-agent keypairs; periodic rotation; revocation lists; short-lived tokens.
  - **NAT traversal:** ICE with STUN/TURN; enterprise proxy awareness via `HTTPS_PROXY/NO_PROXY`.
  - **Zero-trust posture:** End-to-end crypto for CRDT payloads; relays are untrusted pass-through.
  - **Audit:** Sign merge/snapshot events; attach chain-of-custody to DAG metadata.
- 

## 6) Observability, Resilience & Debuggability

1. **Eventual consistency by design**
  2. CRDT guarantees convergence even with partitions; persist op logs locally; replay on reconnect. Garbage-collect tombstones over time.
  3. **Immutable audit graph**
  4. Every snapshot/merge is an append-only DAG node, mirrored to a signed Git commit. Rollbacks are just ref updates to earlier nodes.
  5. **Metrics & tracing**
  6. Emit Prometheus/OTel: op latency, ops/sec, queue depth, merge success rate, artifact hydration times, agent heartbeats.
  7. **Session journal**
  8. Structured log of join/leave, ops batch IDs, snapshot requests, merge attempts, CI/test results, artifact registrations. Correlate with commit IDs.
  9. **Debug tools**
  10. Dump live CRDT state; inspect pending ops; “time-travel replay” to a target moment; redaction for PII in logs.
- 

## 7) Extensibility for Non-Code Modalities

- **JSON/YAML:** Schema-aware merge; preserve array identity; stable key ordering.
- **Notebooks:** Cell-graph diff/merge; preserve execution metadata; gate on re-exec tests.
- **IDLs/Protos:** Symbol-aware rename/move; API compatibility checks.

- **Media:** Store manifests + metadata diffs; binaries pinned via `{key, version_id, sha256}`.
  - **Models/Datasets:** Version via S3-versioning/IPFS; dataset fingerprints; data contracts as tests.
- 

## 8) Human Visibility Features — Implementation Notes

- **Live viewer:** The VS Code extension subscribes to room ops; renders cursors and a live diff heatmap; shows per-lane filters.
  - **Preview/confirm:** The `snapshot` API supports dry-run → present AST/test summary → require confirmation → persist DAG node + Git commit.
  - **Playback:** The room persists an op-journal (compacted). The UI offers a timeline with play/pause/scrub; “checkout at t” reconstructs files.
  - **Recording options:** Toggle pixel capture (ffmpeg) or code-replay capture (compact). Store captures under `.ai/sessions/` and pin with artifacts.
- 

## 9) Deployment Topologies (Secure)

Topology	When to use	Pros	Cons
Local-only	Solo dev, offline	No infra, simple	No live collaboration
Ephemeral relay (devcontainer)	Small team, Codespaces	Easy, low-latency, no firewall ops	Relay process lifecycle mgmt
WebRTC P2P	Cross-site agents	E2E, low relay cost	NAT/TURN complexity
Managed relay cluster	Enterprise scale	Governance, quotas, SSO, audits	Operate/maintain service

---

## 10) Updated Milestones (v0.3)

- **A. Local CRDT + Git mirror** (done in v0.1 plan)
  - **B. S3-versioned adapter** (done in v0.1 plan)
  - **C. Semantic merge (AST + tests)** (v0.1 plan)
  - **D. VS Code live viewer + Preview/Confirm** (new)
  - **E. Playback & Session Journal** (new)
  - **F. P2P/Relay connectivity + auth** (new)
  - **G. Cross-repo meta-commit groups** (new)
-

## 11) FAQs — Live Rooms & Static Git

**Q: A local Git repo is static files. Do I need a CRDT server to see live changes?**

**A:** No. For solo work: CRDT runs in-process and you commit locally. For live collaboration: run an **ephemeral local relay** or use **P2P WebRTC**. A full-time central server is optional, not required.

**Q: How do remote agents connect securely to my local machine?**

**A:** Use WebRTC (STUN/TURN) or a temporary TLS relay with token/mTLS auth. All CRDT payloads are end-to-end encrypted; relays can be untrusted.

**Q: Can a human watch the repo “grow” in real-time?**

**A:** Yes. The VS Code extension shows streaming edits (cursors/diffs) and an animated DAG. You can also enable pixel or code-level recording for playback.

**Q: Does ANVCS still create Git commits?**

**A:** Yes. Every snapshot/merge creates a Git-compatible commit with structured AI metadata. Git remains interoperable with your existing hosting/CI.

---

**Summary.** These additions harden ANVCS for the multi-agent future: live visibility, resilient sync without mandatory servers, secure remote connectivity, and rich provenance on an immutable DAG mirrored to Git. The design remains incremental, enterprise-friendly, and extensible across code and non-code artifacts.