ANVCS — AI-Native Version Control System

Subtitle: Problem Statement, Design Scope, Architecture & Prototype Plan for AI-Native, Git-Compatible Version Control (v0.1)

0) Executive Summary

AI coding agents now contribute code continuously and in parallel. Git was designed for thoughtful, human-paced workflows. The mismatch manifests as conflict storms, brittle rebases, and cross-repo drift. **ANVCS** (AI-Native Version Control System) is a **Git-compatible** overlay that adds:

- **CRDT live collaboration** for conflict-free concurrent editing
- **DAG (graph) history** instead of strictly linear branches
- **AI-aware provenance** (prompts, model versions, test outcomes) captured per commit
- **Semantic merge engine** (AST, test-guided, optional LLM assist)
- **Pluggable artifact storage** for any file type (S3 versioning aware, IPFS, NFS, etc.)
- **Adoptable incrementally**: runs alongside Git; existing Git/GitHub workflows continue to work

ANVCS preserves Git's **immutability and content addressability** while introducing AI-native collaboration and metadata.

1) Problem Statement

1. **Conflict explosion**: Many agents edit the same codebase concurrently; traditional 3-way merges and rebases fail frequently and block progress.
2. **Strict merge order & divergence**: Agents must serialize on `main`. By the time merges complete, parallel branches have diverged further.
3. **Context loss**: Git doesn't record *why* changes were made (prompt, model, tests). Review and reconciliation lack intent.
4. **Multi-repo sprawl**: Modern systems span multiple repos/assets (code, models, data, design). Git has no native atomic cross-repo versioning.
5. **Non-code assets**: Large binaries, datasets, models, notebooks: Git alone is inefficient. Reproducibility needs precise artifact pinning.

2) Goals & Non-Goals

Goals - Preserve Git's **immutability**, **integrity**, and **interoperability**. - Enable **24/7 multi-agent** collaboration with **minimal conflicts**. - Make intent **traceable**: prompts, model versions, test results captured per commit. - Support **any file type**; provide **pluggable, version-aware storage adapters** (e.g., S3 object versioning, IPFS) and precise artifact retrieval. - Provide **incremental adoption**: keep Git commands, CI/CD, hosting (GitHub) working.

Non-Goals (v0.1) - Replacing Git server infrastructure wholesale. - Blockchain/consensus layers for global truth (future option). - Automatic merging of inherently semantic conflicts without tests/policies.

3) Core Principles

1. **Git-Compatibility First**: Maintain `.git/`; map ANVCS commits to Git commits and refs; use Git notes/messages for metadata; push/pull over Git.
2. **Immutability & Reproducibility**: Content-addressed commits; artifact metadata includes content hashes and storage **version IDs**.
3. **CRDT for Edits, DAG for Versions**: Continuous, conflict-free edits; snapshot meaningful states into a commit DAG.
4. **Provenance as a First-Class Citizen**: Prompt, model, toolchain, tests captured per commit.
5. **Semantic Merging**: Prefer AST/test-guided merges; escalate to review only on irreconcilable semantics.
6. **Local-First, Cloud-Ready**: Fully local dev loop; scalable sync to remotes/peers/cloud.
7. **Pluggable Everything**: Storage adapters, merge strategies, commit policies, identity providers.

4) Conceptual Model

- **CRDT Room**: Editors/agents collaborate in real time. Edits are operation-based CRDTs per file; operations commute and converge.
- **Commit = Semantic Snapshot**: When an agent reaches a logical milestone (task complete + tests pass), it snapshots the current state into a DAG commit.
- **DAG (Directed Acyclic Graph)**: Commits form a graph (multiple parents allowed). Branches/streams are **named pointers** to DAG nodes.
- **Agent Context**: Each commit carries structured metadata: prompt, model version, test report, coverage, environment.
- **Semantic Merge Engine**: Combines changes via AST and tests; optionally LLM-assisted; creates merge commits in the DAG.
- **Artifact Store**: Large/non-text assets are offloaded via adapters (e.g., S3 with VersionId; IPFS CIDs). Commits pin exact versions.

5) Architecture Overview

```
┌────────────────────────────────────────────────────────────────────────────┐
│                         IDEs / Agents / CLI (local)
│
│  VS Code ext (CRDT cursors) • Agent runners • ai-vcs CLI • Pre-commit
hooks│
└──────────────┬─────────────────────────────────────────────────────────────┘
               │ CRDT ops (WebSocket/WebRTC/gRPC)
┌──────────────▼──────────────┐    ┌──────────────────────────────┐
│   CRDT Collaboration Core   │    │     Commit DAG Manager       │
│  (per-file op logs, merge)  │    │  (graph store, refs/streams) │
└──────────────┬──────────────┘    └──────────────┬───────────────┘
               │                                  │
        live buffers                          DAG snapshots
               │                                  │
```

```
┌──────────────▼──────────────┐    ┌──────────────▼───────────────┐
│  Semantic Merge Engine      │    │  Provenance & Metadata Store │
│ (AST/test/LLM strategies)   │    │ (prompts, models, tests, env)│
└──────────────┬──────────────┘    └──────────────┬───────────────┘
               │                                  │
         merged diffs                        commit annotations
               │                                  │
┌──────────────▼──────────────┐    ┌──────────────▼───────────────┐
│   Git Compatibility Layer   │    │   Artifact Storage Adapters  │
│  (.git objects/notes/refs)  │    │ (S3 ver., IPFS, NFS, local)  │
└──────────────┬──────────────┘    └──────────────┬───────────────┘
               │                                  │
       push/pull over Git                   upload/download blobs
               │                                  │
       Git remotes / GitHub             S3 buckets / IPFS / NFS / etc.
```

6) Data Model (v0.1)

6.1 Commit Object (conceptual)

```
commit_id: <content-hash>
parents: [<commit_id>, ...]
author: { name, email, time }
message: |
  Human summary…

  ai-meta:
    prompt: "…"
    model: "gpt-4o-2025-10-xx"
    tests_passed: true
    test_report_ref: ".ai/meta/commit/<id>/tests.json"
    coverage: 87.1
    env: { python: "3.12", node: "24", os: "ubuntu-24.04" }
    agent_id: "agent-foo"
    artifacts:
      - type: model
        key: "models/agent_v1.pt"
        storage: s3
        bucket: "anvcs-artifacts"
        region: "eu-west-1"
        version_id: "3HL4IxNZA..."
        sha256: "4f34...9a1c"
        size: 104857600
```

> **Note:** AI metadata may be embedded in the commit message, or attached via `git notes`, and mirrored in `.ai/meta/<commit_id>.json`.

6.2 CRDT Ops (per file)

```
file: "src/app.py"
ops:
  - id: "op-123"
    ts: 1730569200
    type: insert
    pos: 210
    text: "def foo(): ...\n"
    origin: "agent-foo"
  - id: "op-124" # ...
```

6.3 DAG Store

- Nodes: commits (immutable)
- Edges: parent links (append-only)
- Refs/Streams: named pointers to nodes (mutable pointers only)

---

7) Commit Autonomy & Policies (Agent-Side)

Commit decision protocol (per agent): 1. Task completion ✅2. CRDT integration ✅(pull/merge recent ops) 3. Tests pass / quality gates ✅4. Redundancy/supersession check ✅5. Semantic completeness ✅ (avoid half-thought commits) 6. Optional human/policy gate ✅

Policy examples - *Per-task (default):* commit when job complete + tests pass. - *Time-boxed:* commit every N minutes of material change. - *CI-validated:* only commit if CI green.

---

8) Branches/Streams & Graph Semantics

- ANVCS keeps **branches** as named references to DAG nodes (aka *streams*).
- CRDT room is shared live state; **commits** capture coherent snapshots at agent-chosen boundaries.
- Merges create new DAG nodes with multiple parents; rebases are minimized (graph is the truth).

---

9) Semantic Merge Engine (v0.1)

- **Primary**: AST-level merge (Tree-sitter/Babel/Roslyn/etc.).
- **Guards**: run tests, static analysis, type checks on merged result.
- **LLM assist (optional)**: propose resolutions when AST diffs conflict semantically; include rationale in metadata.
- **Escalation**: route to human review on policy or test failure.

---

10) Storage Adapters (Any File Type)

10.1 Design

- **Interface**

  ```
  class BlobStore:
      def upload(local_path: str, key: str) -> dict: ...
      def download(key: str, version_id: Optional[str], dest_path: str) ->
  None: ...
      def exists(key: str, version_id: Optional[str] = None) -> bool: ...
      def delete_version(key: str, version_id: str) -> None: ...
  ```

- **Adapters**: `S3VersionedBlobStore`, `IPFSBlobStore`, `LocalBlobStore`, `NFSBlobStore`, `CompositeBlobStore` (hybrid orchestration).

- **Commit-time flow**: match files against `blobs.track` patterns → hash → upload via adapter → record `{key, version_id, sha256, size}` in commit metadata.

- **Checkout-time flow**: resolve metadata per commit/branch → hydrate local cache by fetching exact versions (supports lazy fetch and offline mode).

10.2 S3 Versioning (Prototype Scope)

- Buckets with **Versioning Enabled**; single stable key per artifact; each update produces a new `VersionId`.
- Commit metadata pins `key` + `version_id` + `sha256` to guarantee reproducibility.

Example metadata

```
{
  "artifact": {
    "type": "model",
    "key": "models/agent_v1.pt",
    "storage": "s3",
    "bucket": "anvcs-artifacts",
    "region": "eu-west-1",
    "version_id": "3HL4IxNZAvtZ...",
    "sha256": "4f34...9a1c",
    "size": 104857600
  }
}
```

Example config (`ai-vcs.config.yaml`)

```
storage:
  mode: hybrid
  primary: s3
  fallback: local

blobs:
  track:
    - "*.pt"
    - "*.onnx"
    - "datasets/**/*.csv"
  upload_on_commit: true
  compress: true

adapters:
  s3:
    provider: aws
    bucket: "anvcs-artifacts"
    region: "eu-west-1"
    endpoint_url: "https://s3.eu-west-1.amazonaws.com"
    access_key_id: "${S3_KEY}"
    secret_access_key: "${S3_SECRET}"
```

---

11) Git Interop Strategy

- **Dual-write**: ANVCS commit → also materialize a standard Git commit (tree, parent(s), author). AI metadata is stored in `git notes` and/or a structured block in the commit message; mirror in `.ai/meta/`.
- **Refs**: DAG heads map to Git refs; `ai-vcs push` calls `git push`.
- **Pull/Clone**: `ai-vcs pull` calls `git pull` + fetches `.ai/` metadata; artifacts rehydrated via adapters.
- **Migration**: `ai-vcs init` on an existing repo bootstraps `.ai/` without rewriting history; legacy commits get empty metadata.

---

12) Security, Identity, Governance

- **Commit signing**: Ed25519/GPG signing for human and agent identities.
- **Agent identity**: keypairs per agent; map to policy roles (read/write/merge).
- **Branch/stream protections**: policy gates (tests required; codeowners; model version allowlist).
- **Audit trails**: immutable logs of merges, policy decisions, and LLM-assisted resolutions.

---

13) CLI, API & VS Code UX (v0.1)

13.1 CLI (selected)

```
ai-vcs init
ai-vcs collab start|join <room>
ai-vcs add <paths>
ai-vcs commit -m "feat: X"
  --prompt "Implement X" --model gpt-4 --test-report ./.ai/meta/tests.json
ai-vcs merge <src> into <dst> [--auto --test-check]
ai-vcs graph --since 2w --filter agent:foo
ai-vcs push origin main
ai-vcs pull origin main
```

13.2 API (sketch)

- `POST /crdt/ops` : append ops
- `POST /commits` : create commit (payload: tree hash, metadata)
- `POST /merge` : request semantic merge
- `GET /graph` : query DAG nodes/edges
- `GET /artifacts/:key?version_id=` : retrieve artifact

13.3 VS Code Extension (concept)

- **CRDT Cursors** (per agent color), **Live Diff Heatmap**
- **Task Timeline** with per-agent commit readiness (🟠pending / ✅committed)
- **DAG Graph View** with filters (by agent, file, tests)
- **Artifacts Pane** showing pinned models/datasets with version IDs

---

14) Prototype Plan (MVP → v0.1)

Milestone A — Local CRDT + Git Mirror - File-level CRDT with local room (single machine / multi-process) - Snapshot to DAG → dual-write Git commit with AI metadata in notes - Basic CLI: `init`, `commit`, `graph`, `push/pull`

Milestone B — S3 Versioned Adapter - Implement `S3VersionedBlobStore` (upload returns `VersionId`) - `blobs.track` patterns, commit-time upload, metadata pinning - Checkout rehydration by `key+version_id`

Milestone C — Semantic Merge (AST + Tests) - Integrate Tree-sitter for AST merges (JS/TS/Python first) - Test-gated merges; CLI `ai-vcs merge --test-check`

Milestone D — VS Code Extension (Preview) - CRDT cursors; DAG view; commit metadata panel; artifact thumbnails/links

Milestone E — Remote Sync - WebSocket relay for CRDT ops; push/pull via Git remote; metadata fetch; artifact hydration

15) Future Directions

- **Federated multi-repo graph** (atomic cross-repo commits)
- **Speculative execution branches** (agents propose branches auto-benchmarked in sandboxes)
- **Non-code semantic diffs** (designs, notebooks, datasets)
- **Policy DSL** for governance (merge/test/deploy gates as code)
- **Deterministic env pinning** (Nix/containers recorded per commit)
- **P2P sync** (IPFS/libp2p for offline teams)

16) Risks & Mitigations

- **Tooling complexity** → *Incremental adoption; Git remains source of truth; clear escape hatches to plain Git.*
- **Merge misresolutions** → *Tests first; conservative policies; human escalation.*
- **Artifact availability** → *Hybrid adapters; local caching; content hash verification.*
- **Vendor lock-in** → *Open formats, adapter interfaces, Git wire-protocol compatibility.*

17) Glossary

- **CRDT**: Conflict-Free Replicated Data Type enabling auto-merge of concurrent edits.
- **DAG**: Directed Acyclic Graph for commit history; merges add nodes with multiple parents.
- **Provenance**: Metadata describing *why/how* a change was made (prompts, models, tests, env).
- **Artifact**: Non-text asset (models, datasets, images, videos, notebooks, binaries).

18) Appendix — Code & Schemas (Extracts)

18.1 BlobStore Interface & S3 Versioned Adapter (Python-style)

```python
from abc import ABC, abstractmethod
from typing import Optional, Dict

class BlobStore(ABC):
    @abstractmethod
    def upload(self, local_path: str, key: str) -> Dict: ...
    @abstractmethod
    def download(self, key: str, version_id: Optional[str], dest_path: str) -> None: ...
    @abstractmethod
    def exists(self, key: str, version_id: Optional[str] = None) -> bool: ...
    @abstractmethod
    def delete_version(self, key: str, version_id: str) -> None: ...
```

```python
import boto3, hashlib, os
from typing import Dict, Optional

class S3VersionedBlobStore(BlobStore):
    def __init__(self, bucket: str, region: str, endpoint_url: Optional[str]
= None):
        self.bucket = bucket
        self.client = boto3.client("s3", region_name=region,
endpoint_url=endpoint_url)

    def _sha256(self, path: str) -> str:
        h = hashlib.sha256()
        with open(path, "rb") as f:
            for chunk in iter(lambda: f.read(8192), b""):
                h.update(chunk)
        return h.hexdigest()

    def upload(self, local_path: str, key: str) -> Dict:
        with open(local_path, "rb") as f:
            resp = self.client.put_object(Bucket=self.bucket, Key=key,
Body=f)
        return {
            "key": key,
            "version_id": resp.get("VersionId"),
            "sha256": self._sha256(local_path),
            "size": os.path.getsize(local_path)
        }

    def download(self, key: str, version_id: Optional[str], dest_path: str) -
> None:
        params = {"Bucket": self.bucket, "Key": key}
        if version_id:
            params["VersionId"] = version_id
        obj = self.client.get_object(**params)
        with open(dest_path, "wb") as f:
            f.write(obj["Body"].read())

    def exists(self, key: str, version_id: Optional[str] = None) -> bool:
        try:
            params = {"Bucket": self.bucket, "Key": key}
            if version_id:
                params["VersionId"] = version_id
            self.client.head_object(**params)
            return True
        except self.client.exceptions.ClientError:
            return False

    def delete_version(self, key: str, version_id: str) -> None:
        self.client.delete_object(Bucket=self.bucket, Key=key,
VersionId=version_id)
```

18.2 Example `ai-vcs.config.yaml`

```yaml
agent:
  name: "agent-synth-x"
  model: "gpt-4o"
  commit_policy: "task-complete"

storage:
  mode: hybrid
  primary: s3
  fallback: local

blobs:
  track:
    - "*.jpg"
    - "*.png"
    - "*.pt"
    - "*.onnx"
    - "datasets/**/*.csv"
  upload_on_commit: true
  compress: true

adapters:
  s3:
    provider: aws
    bucket: "anvcs-artifacts"
    region: "eu-west-1"
    endpoint_url: "https://s3.eu-west-1.amazonaws.com"
    access_key_id: "${S3_KEY}"
    secret_access_key: "${S3_SECRET}"
  ipfs:
    gateway: "https://ipfs.io"
    pinning_service: "web3.storage"
    api_token: "${IPFS_TOKEN}"
  nfs:
    mount_path: "/mnt/anvcs_blobs"
```

18.3 Minimal Commit Metadata (JSON)

```json
{
  "commit_id": "abc123...",
  "ai_meta": {
    "prompt": "Implement validation",
    "model": "gpt-4o-2025-10",
    "tests_passed": true,
    "coverage": 91.2,
    "agent_id": "agent-validate",
    "artifacts": [
      {
        "type": "model",
        "storage": "s3",
        "bucket": "anvcs-artifacts",
        "region": "eu-west-1",
        "key": "models/validator.pt",
        "version_id": "3HL4IxNZ...",
        "sha256": "4f34...9a1c",
        "size": 104857600
      }
    ]
  }
}
```

---

19) Conclusion

ANVCS brings **AI-native collaboration** to version control without abandoning Git. CRDT live editing, graph histories, and rich provenance make continuous multi-agent development practical. S3-versioning-aware artifacts ensure reproducibility and storage efficiency across any file type. The MVP plan enables a concrete, end-to-end demo while keeping a clear path to scale across teams and enterprises.

> **Call to Action:** Begin with Milestone A (CRDT + Git mirror) and Milestone B (S3-versioned adapter) to validate the end-to-end flow locally, then layer semantic merges and the VS Code extension.

---

20) Codespaces / VS Code Server & Storage Readiness (v0.2 Addendum)

This addendum guarantees ANVCS works identically **locally** and in **remote VS Code Server environments (e.g., GitHub Codespaces)**, with resilient connectivity to storage providers.

20.1 Run-Anywhere Guarantees

- **Pure CLI + local files**: `.git/` + `.ai/` on disk; no always-on daemon required.
- **Git-compatible push/pull**: Keep Git remotes, PRs, and CI unchanged.
- **Editor-agnostic CRDT transport**: WebSocket/WebRTC/gRPC that runs in a devcontainer.
- **All storage via HTTPS adapters**: S3/MinIO/Spaces/IPFS/NFS through outbound 443.

20.2 S3 Adapter: Reliability & Enterprise Considerations

- **Native versioning**: Commits pin `{ key, version_id, sha256, size }`. `download()` always fetches the exact `VersionId`.
- **Retries & timeouts**: Exponential backoff with jitter for PUT/GET/HEAD; bounded concurrency.
- **Multipart uploads**: Threshold & part size tuned for large files (e.g., 64MB+ threshold).
- **Proxy support**: Honors `HTTPS_PROXY/HTTP_PROXY/NO_PROXY` automatically.
- **Endpoint flexibility**: Configurable `endpoint_url` for MinIO/Spaces; supports virtual-host & path styles.

- **Clock-skew guard**: Preflight skew check; clear error if SigV4 fails due to time drift.
- **Idempotency/dedup** (optional): Write `x-amz-meta-sha256`; skip re-upload if same content already exists.
- **Auth models**: Static keys, **OIDC/STS**, or **presigned URLs** for restricted egress/IP policies.
- **Secrets hygiene**: Read from Codespaces/Org/Repo secrets → env; no plaintext keys in images.

20.3 Devcontainer (Codespaces) Setup

`.devcontainer/devcontainer.json`

```json
{
  "features": {
    "ghcr.io/devcontainers/features/aws-cli:1": {},
    "ghcr.io/devcontainers/features/python:1": { "version": "3.12" }
  },
  "containerEnv": {
    "AWS_REGION": "eu-west-1",
    "AI_VCS_S3_BUCKET": "anvcs-artifacts",
    "HTTPS_PROXY": "${containerEnv:HTTPS_PROXY}"
  },
  "secrets": ["AWS_ACCESS_KEY_ID","AWS_SECRET_ACCESS_KEY"],
  "postCreateCommand": "ai-vcs storage check"
}
```

`ai-vcs.config.yaml` (works local & Codespaces)

```yaml
storage:
  mode: hybrid
  primary: s3
  fallback: local

adapters:
  s3:
    bucket: ${AI_VCS_S3_BUCKET}
    region: ${AWS_REGION}
    endpoint_url: ${S3_ENDPOINT_URL:-}
    use_presigned: true

blobs:
  track: ["*.pt","*.onnx","datasets/**/*.csv","**/*.ipynb","**/*.png","**/*.mp4"]
  upload_on_commit: true
  compress: true
```

20.4 Optional Local/Offline: MinIO Sidecar

`docker-compose.minio.yml`

```
services:
  minio:
    image: quay.io/minio/minio:latest
    command: server /data --console-address ":9001"
    ports: ["9000:9000","9001:9001"]
    environment:
      MINIO_ROOT_USER: minio
      MINIO_ROOT_PASSWORD: minio123
    volumes: ["./.minio:/data"]
```

Enable versioning once

```
mc alias set local http://127.0.0.1:9000 minio minio123
mc mb local/anvcs-artifacts
mc version enable local/anvcs-artifacts
```

Set `S3_ENDPOINT_URL=http://127.0.0.1:9000` and credentials to `minio/minio123`.

20.5 Connectivity Matrix

Environment	Auth best practice	Endpoint	Notes
Local laptop	~/.aws SSO or static keys	AWS S3	Works offline with local cache; sync later.
Codespaces	Repo/Org secrets → env, or presigned URLs	AWS S3	Avoid IP-allowlists; prefer OIDC/STS or presigned.
Codespaces + MinIO	Local endpoint	`http://minio:9000`	Great for demos; versioning supported.
Self-hosted VS Code Server	IAM Role/SSO	VPC Endpoint	Private S3 via VPC endpoint; no public egress.

20.6 Operational Commands

- `ai-vcs storage check` → DNS/TLS/bucket existence/versioning/PUT-GET roundtrip/proxy detection.
- `ai-vcs storage doctor --verbose` → prints proxy, DNS, clock skew, TLS chain.
- `ai-vcs pull --artifacts=lazy|eager` → control hydration strategy.
- `ai-vcs commit --no-upload` → force offline, queue uploads.
- `ai-vcs retry-uploads` → drain deferred queue with backoff.

20.7 Branch/Stream Switching Behavior in Remote Dev

Switching refs triggers, in order: 1) Git checkout (fast, Git-native) 2) CRDT buffer swap (local) 3) Artifact hydration using pinned S3 `VersionId`s (lazy by default for snappy UX)

20.8 Guardrails / Best Practices

- **Bucket policy**: Prefer IAM roles/conditions over IP allowlists (Codespaces egress IPs vary).
- **Time sync**: Ensure container clock is correct (SigV4 time-sensitive).
- **Proxy envs**: Set `HTTPS_PROXY/NO_PROXY` in devcontainer if required by enterprise networks.
- **Large files**: Tune multipart thresholds/concurrency; bound memory during uploads.
- **Git LFS interop**: Choose whether ANVCS or LFS owns a pattern to avoid double handling.

This section formalizes the portability, reliability, and security posture for running ANVCS in Codespaces/VS Code Server and local environments without connectivity surprises.