Arc: Towards Structured, Concurrent, and Provenance-Aware Collective AI Memory for Software Engineering

Jarrod Barnes
Arc Computer
Jarrod@arc.computer

May 2025

Executive Summary

As AI generates exponentially more code, the critical bottleneck in software engineering is shifting from *generation* to *understanding, provenance, and coordination.* Arc addresses this fundamental challenge by transforming every codebase into a continuously evolving world model—a live, executable graph of architecture, rationale, and provenance that both humans and AI agents can query, update, and learn from. This memory layer is the missing link between today's siloed copilots and tomorrow's coordinated, autonomous software factories. For engineering leaders, Arc delivers increased velocity, enhanced trust through transparent provenance, improved resilience during incidents, and true AI augmentation through contextual awareness. Our initial focus on AI code review provides immediate value while building toward our vision of coordinated multi-agent systems that can safely evolve complex software in parallel.

Abstract

Despite their ability to resolve real-world GitHub issues and ingest vast context windows, frontier AI coding agents struggle with a "transient-context ceiling"—preserving causal rationale behind long-running changes. Complex software evolution requires memory that is persistent, structured, and supports semantically-aware concurrent modifications.

Arc is an architecture enabling multiple reinforcement learning agents to collaboratively manage codebases as continuously-evolving world models through three integrated components: (i) **Temporal Knowledge Graphs (TKGs)** providing structured, queryable history of code artifacts and decisions; (ii) **Causal CRDT layer** extending Conflict-free Replicated Data Types with semantic invariant checks derived from the TKG; and (iii) **Provenance-Driven RL** rewarding agents for generating and consuming causal context, fostering coordination and auditable reasoning.

This approach transforms memory from passive retrieval into an active, shared world model for both human and AI developers. We outline the Arc architecture, introduce the SYNAPSE benchmark for empirical validation, and present a research agenda for this critical challenge in AI-assisted software engineering.

Keywords: Temporal Knowledge Graphs, Causal CRDTs, Reinforcement Learning, Software Engineering, Multi-Agent Systems, Provenance

1 Introduction

Software engineering is undergoing a profound transformation driven by AI-powered code generation [? ?]. Despite their capabilities, frontier AI coding agents face a critical challenge: preserving causal rationale behind long-running changes. This ”transient-context ceiling” persists even with multi-million token contexts, as models struggle to retain deep causal relationships and design rationale over time.

North Star. Arc transforms codebases into continuously-evolving world models—live, executable graphs of architecture, rationale, and provenance that both humans and AI agents can query and update. As AI generates more code, the bottleneck shifts from generation to *understanding, provenance, and coordination.* Arc serves as the essential memory layer for verified provenance and coherent multi-agent collaboration, supporting a paradigm where engineering work evolves from implementation to strategic oversight.

When crucial context—like API design reasoning—is lost, its connection to future edits becomes tenuous, impeding collaborative tasks like security upgrades and cross-language migrations. Complex software evolution demands memory that is *persistent*, *concurrently writable*, and *semantically consistent* with software rules. Without such a substrate, development suffers from duplicated work, integration conflicts, opaque AI decisions, and architectural decay—a phenomenon we term ”context collapse.”

Core Thesis: Treat the codebase as a living graph of artifacts, dependencies, and decisions, then enforce semantic rules when agents propose changes. With this foundation, we can train agents to act as reliable, accountable teammates, not reckless auto-complete engines. Lifting the transient-context ceiling and addressing the context collapse necessitates the co-design of three pillars:

1. **Remembering History & Rationale:** Beyond simple file diffs, agents require structured, queryable history. **Temporal Knowledge Graphs (TKGs)** encode the evolving state of artifacts, design decisions (including ingested Architectural Decision Records - ADRs), requirements, and agent actions, allowing queries like ”What requirement led to this API change?” [**? ? ?**].

2. **Enabling Safe Parallel Work:** To allow concurrent modifications by multiple human and AI actors without breaking code semantics, we propose **Causal CRDTs**. These generalize Conflict-free Replicated Data Types (CRDTs) with software-specific semantic checks derived from the TKG, preserving crucial dependencies (e.g., build order, API contracts) and preventing logical conflicts [**?**].

3. **Learning from Collective Experience:** To encourage effective coordination, documentation, and accountability, we employ **Provenance-Driven Reinforcement Learning (RL)**, building on frameworks that synergize reasoning and acting [**? ?**]. Agents are rewarded not only for task progress but critically for generating and consuming causal context within the TKG—linking changes to requirements and documenting decisions—thereby fostering auditable and collaborative AI systems.

This paper details the Arc architecture, discusses related work, outlines the SYNAPSE benchmark for empirical validation, and presents a research agenda inviting community collaboration to build this essential memory layer for the future of software engineering.

Key Takeaway

Arc transforms every codebase into a continuously-evolving world model—a live, executable graph of architecture, rationale, and provenance that both humans and AI agents can query, update, and learn from.

2 Arc at a Glance

Arc provides a shared, persistent, and semantically consistent foundation for collaborative AI software development. It integrates three core components, previously introduced, which work in concert to create an active world model of the software project:

- **Temporal Knowledge Graph (TKG):** The system's collective memory, representing the evolving state of software artifacts, design decisions, and agent actions as a queryable, time-stamped graph.
- **Causal CRDT Layer:** Enables safe concurrent modifications by enforcing software-specific semantic dependencies derived from the TKG, preventing merges that would lead to invalid states.
- **Provenance-Driven RL:** Incentivizes agents not only for task completion but also for enriching and utilizing the TKG, fostering collaborative behavior and a deep understanding of context.

The illustrative example of upgrading OpenSSL across services highlights Arc's potential: Agent A refactors wrappers, Agent B updates API calls. Using Arc, Agent B can query the TKG for Agent A's progress. The Causal CRDT layer buffers Agent B's commit if dependencies (e.g., "Wrapper tests pass") aren't met. Both agents are rewarded for linking their work in the TKG, creating an understandable history.

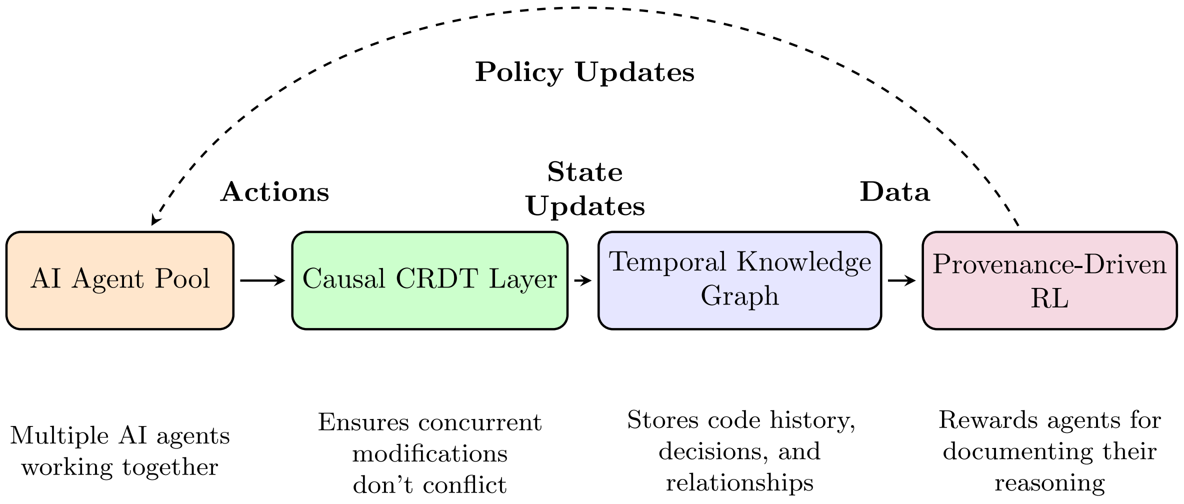
Policy Updates
Actions
State Updates
Data
AI Agent Pool
Causal CRDT Layer
Temporal Knowledge Graph
Provenance-Driven RL
Multiple AI agents working together
Ensures concurrent modifications don't conflict
Stores code history, decisions, and relationships
Rewards agents for documenting their reasoning

Figure 1: Arc Architecture: A simplified view of how Arc's components work together. AI agents make changes through the Causal CRDT layer, which safely updates the Temporal Knowledge Graph. The Provenance-Driven RL component learns from this data and improves agent policies, creating a continuous improvement cycle.

Figure: Arc Architecture - Integrating Temporal Knowledge Graphs, Causal CRDTs, and Provenance-Driven RL

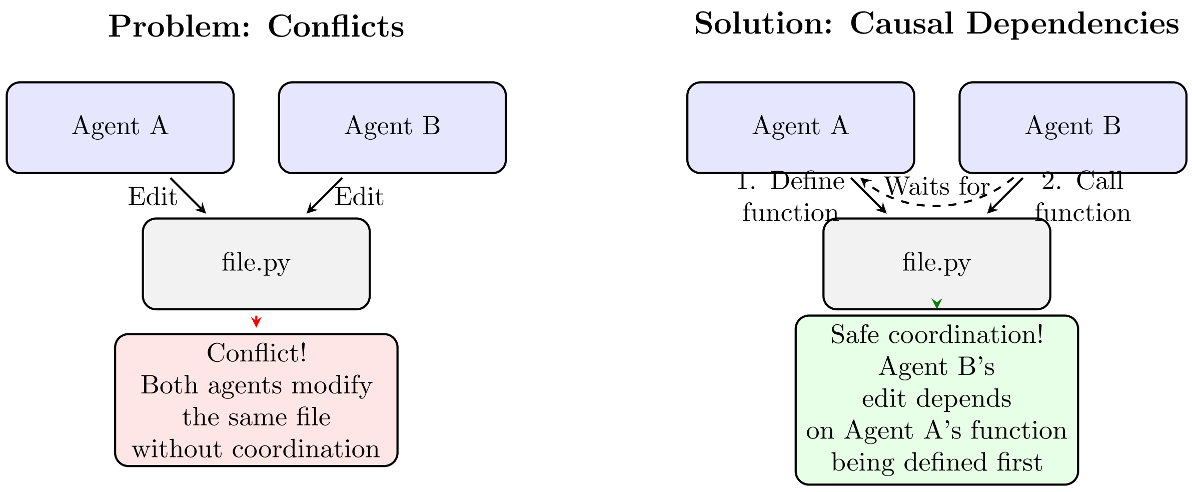
Problem: Conflicts
Agent A
Agent B
Edit
Edit
file.py
Conflict!
Both agents modify the same file without coordination
Solution: Causal Dependencies
Agent A
Agent B
1. Define function
Waits for
2. Call function
file.py
Safe coordination!
Agent B's edit depends on Agent A's function being defined first

Figure 2: Causal Dependency Management: On the left, we see the problem when multiple agents edit the same file without coordination, leading to conflicts. On the right, Arc's solution uses causal dependencies to ensure that operations happen in the correct order - Agent B waits for Agent A to define a function before trying to call it.

Figure: Causal Dependency Management in Arc - Ensuring semantic correctness during parallel development

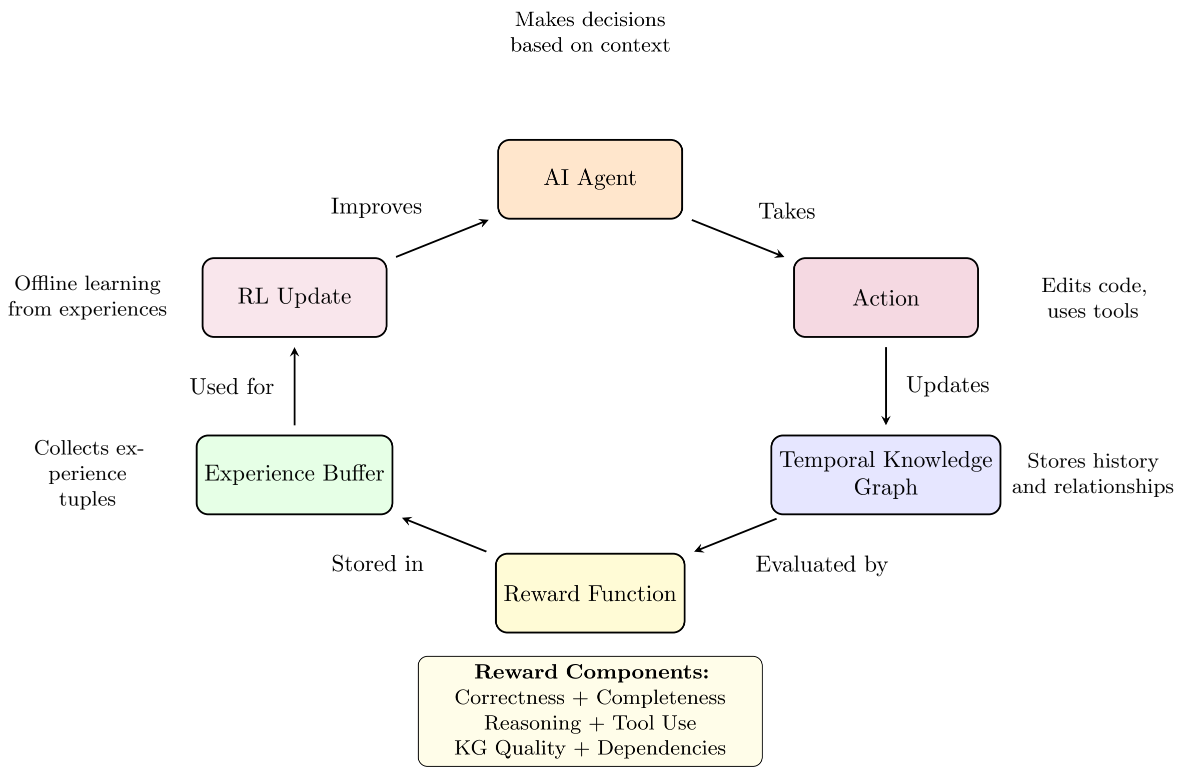

Figure 3: Reinforcement Learning Flow in Arc: AI agents take actions that update the Temporal Knowledge Graph. These actions are evaluated by a multi-faceted reward function. The experiences are collected and used to improve agent policies through offline reinforcement learning.

Figure: Reinforcement Learning Flow in Arc - Training agents to collaborate effectively

3 Problem Statement & Research Gap

The challenge is enabling multiple AI agents to collaboratively remember, reason about, and evolve software systems while maintaining architectural integrity and decision provenance.
Research Question (RQ): How can multiple RL coding agents maintain long-term architectural coherence while concurrently modifying a live codebase?

Existing approaches either (a) treat memory as passive RAG chunks, losing structural semantics, or (b) lock files to serialize edits, throttling parallelism. There is no system that offers structured, concurrent, provenance-aware memory tailored to code evolution. We formalize the gap as:

- **P1 (Structure):** Absence of a canonical, query-efficient model linking commits, decisions, artifacts, and agent executions over time.
- **P2 (Concurrency):** Lack of write protocols (beyond basic CRDTs or locking) that respect higher-level semantic causal dependencies inherent in software evolution.
- **P3 (Learning):** Missing reward signals and learning frameworks that explicitly train agents to utilize shared historical context effectively and value the generation of explainable, historically consistent actions.

4 Background and Related Work

We synthesize advancements in distributed systems, knowledge representation, and reinforcement learning [?]. We build on established concepts, proposing novel integrations for collaborative software evolution [?].

4.1 Core Concepts

Temporal Knowledge Graphs (TKGs) TKGs enhance knowledge graphs by adding temporal data, representing facts as quadruples (h, r, t, τ) with timestamps or intervals τ [?]. Representation learning techniques model these temporal dynamics for forecasting and link prediction [? ? ? ?]. TKGs offer a structured way to capture software evolution history over time [?].

Conflict-Free Replicated Data Types (CRDTs) CRDTs are data structures designed for distributed systems that guarantee eventual consistency without requiring complex consensus protocols, making them suitable for collaborative applications under network partitions [?]. They achieve this through merge functions that are commutative, associative, and idempotent.

Reinforcement Learning (RL) for Code RL has been applied to various software engineering tasks, including code generation, optimization, and refactoring [? ?]. Step-wise or offline RL approaches [? ? ?] are particularly relevant for learning from interaction traces, potentially augmented with synthetic data [?], to train policies that map system states to actions.

4.2 Related Work and Differentiation

- **Long Context LLMs & Sequential Collaboration:** Frameworks like Chain-of-Agents [?] and larger context windows help with long inputs but lack structured, concurrent workflows [? ?].

- **Causal Consistency Models:** Research on causal consistency in distributed systems [? ? ?] informs our Causal CRDT design.
- **Graph-Based Agent Frameworks:** LangGraph [? ?] and Graph-RAG [? ?] use graphs for orchestration and retrieval; Arc uses the TKG as the shared, mutable world model and RL environment [?].
- **Graph-of-Thoughts(GoT**:GoT [?] utilizes graph structures internally within a single LLM's reasoning process to explore different solution paths for complex problems. While powerful for individual agent reasoning, it doesn't address the shared, persistent, and concurrently modified memory challenges inherent in multi-agent software evolution that Arc targets.
- **Graph CRDTs:** Prior work on graph CRDTs and probabilistic causal contexts [? ? ?] provides a foundation for our Causal CRDT.
- **RL on Dynamic/Temporal Graphs:** RL for temporal KGs [? ?] and GNN representations [? ? ?] guide our state-encoding choices.
- **GitTemporalAI**: This concurrent work [?] also leverages TKGs derived from Git history for repository intelligence, primarily focusing on analysis and querying by LLMs. Arc differentiates by using the TKG as a *live, mutable environment* for *concurrent agent action* mediated by Causal CRDTs and learned via RL, going beyond historical analysis to enable active, collaborative evolution.
- **Provenance Tracking:** Provenance mechanisms in storage systems [? ? ?], Software Heritage [?], and supply-chain frameworks like SLSA [? ?] inspire our provenance integration.
- **Multi-Agent Collaboration & Memory:** Dialogue-based frameworks (ChatDev [?], CAMEL [?]), Generative Agents [?], and shared-memory MARL [? ?] point to the need for a structured substrate.

5 Proposed Approach: High-Level Architecture

As illustrated in Figure 1, Arc integrates three key components that work together to create a cohesive memory layer for software engineering.

5.1 Temporal Knowledge Graph (TKG) as Environment

The TKG serves as the shared world model and RL environment. *Think of it as the project's collective, structured memory or logbook, capturing not just the code but the relationships between components, changes, and decisions over time.*

- **Structure:** Software entities (services like `auth-service`, `payment-service`, files such as `openssl_wrapper.c`, functions, commits, build dependencies, API contracts, agent assignments, and decisions) are nodes. Relationships (e.g., `depends_on`, `calls_api`, `modified_by`, `tested_by`) are typed edges with valid time intervals [?], representing the system's state. [?].
- **Query Capabilities:** Agents interact with the TKG via temporal graph queries [?].
  - *Example Query (Agent A):* "What services currently depend on `libssl` and were last successfully built before my refactoring task started?" (Temporal snapshot query).
  - *Example Query (Agent B):* Show the history of changes to the `PaymentAPI::process()` signature, and which agent initiated them." (Historical path query). Efficient indexing and partitioning techniques are crucial.

  [? ? ?].

- **RL State Representation:** Relevant TKG subgraphs, like Agent A's dependency graph, are encoded with GNNs for temporal graphs [? ? ? ?], producing vector states for policy networks.

World-model view. We treat the pair (TKG, f_{merge}) as a symbolic world model. The TKG provides the observable state s, while the Causal-CRDT merge function $f_{\text{merge}}(s, a)$ deterministically computes the next state if action a is semantically valid. Because f_{merge} is queryable before committing, an agent can ask counter-factual questions—e.g., "If I bump OpenSSL to 3.0, which services will violate dependency tests?". While the goal is to receive answers rapidly (e.g., within milliseconds for common queries), achieving this for massive, evolving TKGs relies on ongoing research into highly scalable graph database technologies, efficient indexing, partitioning techniques [? ? ?], and query optimization. This elevates Arc beyond large-context retrieval toward model-based planning in software-engineering domains.

5.2 Causal CRDT Layer

We propose an operation-based Causal CRDT to manage concurrent writes while preserving semantic dependencies. Standard CRDTs ensure concurrent text edits eventually merge, but they don't understand if merging breaks the build or violates an API contract. Our Causal CRDT layer adds software-specific "guardrails," derived from the TKG, to prevent merges that, while textually valid, would be semantically incorrect in the context of the software project.

- **Design:** Each operation has dependency metadata ('deps') and a vector clock. Merges happen when all dependencies and causal predecessors are satisfied [? ? ? ?], otherwise they are buffered.
  - *Example Operation (Agent B):* `Op:UpdateAPISignature(PaymentAPI)` (ensuring Agent A's foundational refactor is committed before the API depending on it is changed). Merges (applying operations to the shared state) occur only when all vector clock predecessors *and* explicit TKG-derived dependencies are satisfied [? ? ? ?]. Operations with unmet dependencies are buffered (See Figure 2).
- **Differentiation:** Its novelty is in enforcing deterministic, application-specific causal invariants critical for software evolution (e.g., "a service cannot deploy if dependent libraries fail tests," "an API change requires updating all callers"). Unlike structural CRDTs or probabilistic approaches, Arc leverages the live TKG state during the merge protocol to define and check semantic invariants via metadata, ensuring correctness beyond eventual consistency. While fine-grained Abstract Syntax Tree (AST) level CRDTs pose additional challenges, our initial focus is on file-level operations guided by TKG semantics. Furthermore, complex semantic checks like full test suite execution are infeasible during the synchronous merge. Our protocol thus prioritizes *efficiently checkable invariants*. For example, an invariant like "API signature (name, arity, parameter types) must match the TKG-defined contract for `PaymentAPI::process()`" can be checked synchronously by querying the TKG. In contrast, a full behavioral test asserting "all existing callers of `PaymentAPI::process()` still function correctly post-change" would typically be deferred to asynchronous CI gates, which are themselves informed by TKG provenance. This approach allows concurrent edits while respecting the rules of software development.

5.3 Step-Wise Reinforcement Learning (RL) Loop

Agents learn optimal policies for evolving codebases within the TKG environment using offline RL from interaction traces [? ?]. *The goal isn't just to teach agents to write code, but to teach them how to be effective members of a collaborative team, using the shared memory to coordinate and understand context.* Each trace step contains the state, action, reward, next state, and causal dependencies: (s,a,r,s′,deps). As illustrated in Figure 3, this process creates a learning flywheel.

- **Multi-Component Reward:** The reward function R guides agents to become effective collaborators by considering multiple factors beyond task completion. $R = R_{\text{corr}} + R_{\text{comp}} + R_{\text{reas}} + R_{\text{tool}} + R_{\text{kg}} + R_{\text{causal}}$

  It consists of several components:

  – (Correctness): Reward for code changes passing tests, static analysis, and type checks, with penalties for build failures or regressions. Penalties for build failures or introducing regressions.

  – (Completion): Reward for making progress towards the overall goal (e.g., percentage of services successfully migrated from the old OpenSSL version).

  – (Reasoning Quality): Reward for generating clear, logical intermediate reasoning steps or plans (potentially evaluated by another LLM or based on plan structure).

  – (Tool Use Efficiency): Reward for selecting appropriate tools (compiler, test runner, refactoring engine) and using them effectively.

  – (TKG Enrichment): **Crucially, reward for enriching the TKG with valuable provenance.** *This incentivizes agents to "document their work" by linking their actions to reasons within the shared memory, making the system's history understandable.Example:* Agent B gets a reward for adding an edge `updates_dependency_introduced_by` linking its API change node to Agent A's earlier commit node in the TKG. This explicitly captures the rationale. *Note: Simply counting new edges (R_{kg}) could potentially be gamed by agents adding trivial links. Future refinements will explore quality-based provenance rewards, perhaps using link prediction models trained on human-validated traces or rewarding edges that resolve specific TKG consistency queries.*

  – (Causal Coordination): Reward reflecting an action's interaction with the Causal CRDT layer's dependency constraints. Positive reward is granted when an action satisfies dependencies that unblock *other agents'* buffered operations (promoting flow); negative reward (or cost) is incurred for actions that become buffered themselves due to unmet prerequisites or contribute to potential causal cycles (discouraging deadlock).

    ∗ *Example:* Agent A receives a positive causal boost upon committing its refactor, as this unblocks Agent B's dependent API update operation.

- **Learning Algorithm:** Our primary learning strategy leverages offline RL algorithms (e.g., CQL [?], IQL [?], OREO [?]) for their sample efficiency, training policies ($\pi(a \mid s)$) on accumulated interaction traces. On-policy fine-tuning (e.g., using PPO) based on live interactions within the Synapse environment is a potential subsequent step for further optimization, considered future work for the initial validation phase. Initial traces for bootstrapping the learning process may be sourced from existing human commit histories, curated datasets of code evolution, or through a human-in-the-loop phase where expert developers guide initial agent actions.

- **Flywheel Effect:** Enhanced policies enable agents to produce more effective actions and richer provenance traces. These traces yield higher-quality data for subsequent offline RL training, fostering a self-improving cycle where agents excel at collaborative, long-term software evolution.

6 Practical Considerations & Industry Impact

While Arc presents a powerful vision, practical adoption requires addressing several considerations:

- **Integration:** Integration with existing developer ecosystems is key. This requires developing adapters for common Git workflows (e.g., interpreting Arc operations as Git commits with rich metadata), hooks for CI/CD pipelines (e.g., validating TKG consistency before deployment), and potentially IDE plugins for visualizing TKG context and agent activity.

- **Scalability:** Real-world repositories can be massive. Efficient TKG storage (e.g., using specialized graph databases [**?** **?**]), indexing, and query optimization are paramount. Vector clock and dependency metadata compaction strategies are needed.
- **Human Oversight:** Mechanisms for human developers to inspect the TKG, understand agent decisions via provenance trails, override actions, and guide the RL process are essential for trust and safety [**?** **?** **?**]. Visualization tools are key.

Addressing these points enables substantial industry impact:

- **Accelerated Large-Scale Changes:** Tasks like the OpenSSL upgrade example, major refactorings, or framework migrations, which are currently slow and error-prone, could be significantly accelerated by coordinated AI agents.
- **Reduced Integration Conflicts:**
  Causal CRDTs and shared TKG awareness minimize broken builds and integration issues common in parallel development.
- **Enhanced Maintainability:** The provenance-rich TKG acts as living documentation, making it easier for both humans and AI agents to understand legacy code and the rationale behind past changes.
- **Autonomous Systems:** Arc provides a foundation for more autonomous AI agents capable of long-term maintenance, proactive bug fixing, and continuous modernization of software systems.

7 Research Agenda & Open Challenges

Arc opens research frontiers at the intersection of distributed systems, program provenance, and reinforcement learning. Its success hinges on collaborative innovation to address the following open challenges, presented here as explicit invitations for collaboration:

- **Formal Foundations:** Key questions include establishing safety and liveness guarantees for Causal CRDTs under real-world branching patterns and defining how semantic invariants (e.g., build determinism, API compatibility) can be expressed in merge-able logic. *Hypothesis:* Causal CRDTs will reduce semantic merge conflicts by $\geq 30\,\%$ on the Synapse benchmark versus standard CRDTs. *Call for collaboration:* Formal-methods researchers for distributed protocol proofs.
- **Scalability & Performance:** Critical challenges involve mitigating TKG query latency in interactive RL loops (via caching, state abstraction, or predictive prefetching), adapting TKG schemas to evolving artifact types while maintaining query consistency, sustaining sub-second queries on graphs exceeding 10^8 edges, and compressing vector-clock metadata without violating causality. *Call for collaboration:* Systems engineers experienced with high-throughput graph databases or CRDT optimization, particularly for temporal graph storage and delta-propagation techniques.
- **Advanced Reinforcement Learning:** We aim to design tractable graph-centric state representations for offline RL and explore hierarchical RL to bridge fine-grained edits with macro-architectural tasks. *Research focus:* Curriculum learning on progressively larger TKG subgraphs and reward decomposition ($\{R_{\text{corr}}, R_{\text{comp}}, R_{\text{prov}}, R_{\text{causal}}\}$) tuned via population-based training. *Call for collaboration:* RL scholars interested in latent-graph policies and combinatorial action spaces for provenance-driven agents.
- **Human–Agent Interaction:** Core questions are how to best design visual abstractions for

million-edge causal chains within the TKG and develop just-in-time human oversight mechanisms for resolving highly ambiguous merges. *Opportunity:* Adapt existing provenance-graph UIs [**? ?**] for deployment as VS Code extensions. *Call for collaboration:* HCI and visualization researchers, and maintainers of developer-tooling plugins.

- **Generalisation & Transfer:** We seek to determine if agent embeddings of causal code traces can transfer across programming languages and whether meta-learning can accelerate adaptation for tasks like migrating a security patch from Java to Python. *Call for collaboration:* Experts in few-shot learning and multilingual code-analysis.
- **Cross-Modal Integration:** Key challenges include reconciling heterogeneous signals—such as issue trackers, design documents (ADRs), and log traces—into a unified TKG schema, and developing retrieval strategies that surface the optimal context slice to an agent within latency constraints. *Call for collaboration:* Information-integration researchers and practitioners of large-scale RAG systems.
- **Limitations & Failure Modes:** Known risks include TKG query latency bottlenecking agent planning, offline RL sensitivity to historical data bias, and potential Causal CRDT deadlocks under complex circular semantic dependencies. *Mitigation focus:* Investigating timeouts, dependency height limits, and designated deadlock "breakers" (human or AI) informed by TKG cycle detection, alongside runtime monitoring and adversarial stress testing. *Call for collaboration:* Reliability engineers and researchers in formal verification of RL agents.

The development of Arc is envisioned as an open, collaborative effort. We actively seek partners from academia and industry with expertise in the aforementioned areas. By engaging with the SYNAPSE benchmark (Section 8) and contributing to these core ideas, we can collectively transform software evolution from a fragile, sequential process into a consistent, multi-agent endeavor. We believe building this foundational memory layer is critical for the future of AI in software engineering.

8 Synapse Benchmark

8.1 V1 Task Definition

Scenario. Two RL-driven coding agents, each powered by Arc, must upgrade `OpenSSL` from v1.1 to v3.0 across a *synthetic microservice monorepo* containing 20 heterogeneously-typed services (14 Python + 6 Java). The synthetic repository comprises approximately 8-12 KLOC across services. The full test suite (`pytest` + `mvn verify`) typically completes in under 15 minutes on standard CI hardware (e.g., 4-core GitHub Actions runner). The repository is generated from a parameterised template inspired by industry testbeds such as SOCK SHOP and HIPSTER SHOP; each service exposes either gRPC or REST endpoints and has transitive dependencies on `libssl`. Dependency graphs and CI pipelines are committed into the Temporal Knowledge Graph (TKG) at `t_0`.

Goal. Within an 8-hour wall-clock budget and a 1 000-action cap per agent, perform the upgrade while preserving all unit & integration tests (`pytest -c -q` for Python, `mvn verify` for Java) and maintaining a clean CI run.

8.2 Environment & Infrastructure

- **Execution Harness.** Docker Compose + Bazel build farm (local or K8s) orchestrates parallel builds and test runs; a Git server with pre-receive hooks provides ground-truth CI verdicts.

- **Arc Instance.** A single logical replica hosts the TKG and Causal CRDT engine. Agents interact solely via Arc's gRPC API—no hidden channel exists.
- **Observation/Action Space.** Agents receive `(diff, TKG_snapshot, build_status)` tuples and may issue atomic CRDT operations: `edit(file)`, `commit(msg)`, `merge(branch)`, `query(TKG)`, etc.
- **RL Algorithm.** Agents are trained primarily using offline RL (e.g., CQL/IQL) on previously collected interaction data within the Synapse environment. A curriculum schedule is employed: warm-start training on single-service upgrades, then advance to the full 20-service monorepo dataset once $\geq 80\,\%$ success is achieved on simpler tasks. (On-policy fine-tuning with algorithms like PPO is reserved for future investigation, potentially in V3).

8.3 Metrics

1. **Task Completion Rate & Latency**—fraction of services upgraded + median completion time.
2. **Integration Errors**—CI failures attributable to semantic regressions (build or runtime).
3. **Concurrent Throughput**—95th-percentile number of non-conflicting CRDT commits per minute.
4. **Provenance Quality**—TKG density of causal edges (`depends_on`, `triggered_by`, `resolved_by`) *validated* against ground-truth commit history; manual spot-checks for interpretability.
5. **Causal Conflict Resolution**—ratio of semantic conflicts auto-resolved by the Causal CRDT vs. rolled back or surfaced to humans.

8.4 Baselines

Single-Agent (Long Context) State-of-the-art model long context model with a 200k+ token window (Claude 3.7 Sonnet, Gemini 2.5 Pro, GPT 4.1); mirrors SWE-bench evaluation style.

Multi-Agent (File Locking) Two agents use coarse file-level locks; high contention expected.

Multi-Agent (Standard CRDT) Automerge-style CRDT without semantic guards; syntactic merge success yet high logical error rate.

Multi-Agent (RAG + Scratchpad) Each agent queries a vector store of code chunks plus a private scratchpad, similar to Graph-of-Thoughts orchestration.

Ablations (i) Arc with Causal CRDT disabled (falls back to locking); (ii) Arc with TKG replaced by naive RAG.

8.5 Reward Function

The reward function for the SYNAPSE benchmark combines dense and sparse signals, directly reflecting the multi-component reward R detailed in Section 5.3. It is formulated as: $R = w_{\text{corr}}R_{\text{corr}} + w_{\text{comp}}R_{\text{comp}} + w_{\text{reas}}R_{\text{reas}} + w_{\text{tool}}R_{\text{tool}} + w_{\text{kg}}R_{\text{kg}} + w_{\text{causal}}R_{\text{causal}}$ where:

- R_{corr} (Correctness): Based on tests passing, static analysis, type checks.
- R_{comp} (Completion): Progress towards the OpenSSL upgrade goal.
- R_{reas} (Reasoning Quality): Quality of intermediate reasoning (if applicable).
- R_{tool} (Tool Use Efficiency): Efficient use of build/test tools.
- R_{kg} (TKG Enrichment): Number and quality of new causal edges added to TKG.

- R_{causal} (Causal Coordination): Positive for unblocking operations, negative for causing buffered operations or deadlocks.

The weights (w_i) will be optimized through techniques like population-based training to balance task velocity with the richness of provenance and collaborative behavior. Initial settings will prioritize correctness and completion, with increasing emphasis on TKG enrichment and causal coordination as agent capabilities mature.

8.6 Success Criteria

Arc aims to achieve **(i)** $\geq 95\,\%$ service upgrade rate, **(ii)** $>50\,\%$ fewer integration errors, and **(iii)** $\geq 2\times$ higher concurrent throughput than locking *and* standard-CRDT baselines, while producing a TKG whose causal-edge density is within $10\,\%$ of ground-truth links.

8.7 Proposed Roadmap

V2 will replace the synthetic mono repo with SWE-BENCH subset issues that require multi-file patches, extending from single-agent to cooperative multi-agent repairs. **V3** plans to introduce cross-language refactors (Python $\leftrightarrow$ Go) and partial offline-RL fine-tuning, aligning with emerging multi-agent toolchains.

9 Conclusion

AI-generated code presents both opportunity and risk: accelerated development alongside ”context collapse” where rationale becomes opaque. Current LLM agents hit a ”transient-context ceiling,” struggling with causal understanding across multiple commits. They may generate code efficiently but fail to preserve architectural intent and the crucial ”why” behind changes. Arc addresses this by integrating Temporal Knowledge Graphs, Causal CRDTs, and Provenance-Driven RL into a cohesive memory plane.

We anticipate significant impact: accelerating large-scale refactorings, enabling self-healing systems, and reducing integration conflicts. The SYNAPSE benchmark provides a concrete validation path, but realizing this vision requires collaborative effort. We invite the research community to engage with these ideas and help build the future of structured, concurrent, and provenance-aware collective AI memory for software engineering—moving beyond transient context toward a new generation of AI-assisted development.

Acknowledgments

We thank the research community for valuable discussions that have shaped this position paper. We welcome collaboration on the open challenges outlined in our research agenda. For inquiries about Arc or potential collaborations, please contact `Jarrod@arc.computer`.

Glossary

Temporal Knowledge Graph (TKG) A dynamic graph structure that captures the evolution of code, design decisions, requirements, agent actions, and their causal interdependencies over time. Unlike static code graphs, TKGs represent facts as quadruples (h, r, t, τ) with timestamps or intervals τ.

Causal CRDT An extension of Conflict-free Replicated Data Types (CRDTs) that enforces semantic invariants derived from the TKG during concurrent operations. These invariants preserve crucial dependencies like build order and API contracts, preventing logical conflicts while allowing parallel edits.

Provenance-Driven Reinforcement Learning A learning approach where agents are rewarded for generating and consuming causal context within the TKG—linking changes to requirements, documenting decisions, and unblocking other agents—fostering truly collaborative and auditable AI systems.

SYNAPSE Benchmark A real-world benchmark for evaluating multi-agent engineering systems, where two AI agents must upgrade OpenSSL across a microservice monorepo while maintaining build integrity and test coverage.

Vector Clock A data structure used in distributed systems to establish a partial ordering of events. In Arc, vector clocks help track causal relationships between operations performed by different agents.

Architectural Decision Records (ADRs) Structured documents that capture important architectural decisions made during the development process, including context, consequences, and alternatives considered.

Context Collapse The phenomenon where critical "why" context is scattered across different tools and systems, making it difficult to understand the rationale behind code changes, especially as AI accelerates code generation.

Provenance Gap The challenge of tracking the origin, requirements, and verification status of AI-generated code, which becomes increasingly important as AI generates more code with less human oversight.

Coordination Chaos The problem that arises when multiple AI agents (and humans) work on the same codebase without a common world model, introducing conflicts and inconsistencies.

World Model A comprehensive representation of a system that captures not just its current state but also the causal relationships and dynamics that govern how it evolves over time.

Learn more about Arc at arc.computer