The Repository as Coordination Layer: Federated AI Agent Governance via Version-Controlled Shared Context

Asif Waliuddin
NXTG.AI
axw@nxtg.ai

July 2026

Abstract

Multi-agent AI systems typically require runtime coordination infrastructure: message brokers, shared memory objects, API surfaces, or orchestration frameworks. Recent work explores version control as an alternative substrate: git worktrees/commits for asynchronous software-engineering agents (Geng et al., 2026), the Linux Foundation–governed AGENTS.md standard, and blackboard-style shared-context coordination (Han & Zhang, 2025). We contribute the first documented production-scale evaluation of git-as-multi-agent-coordination at portfolio governance scope: the ASIF system, in which two autonomous AI agents on physically separate machines have governed 21 software projects for twenty-one weeks (152 days), producing 264 structured cross-machine handoff notes, 62 architecture decision records, and more than 180 governance standards through a single shared repository. Coordination is brokerless and asynchronous: no message broker, shared database, or orchestration service — a 5-minute cron performs git synchronization, and rare urgent messages bypass git via terminal injection (disclosed in the limitations). On top of the standard repository-as-state primitive we describe three architectural patterns — a federated NEXUS hierarchy that aggregates per-project governance into a portfolio view, an identity-via-loading-protocol mechanism that produces role isolation without filesystem permissions, and an observed eventual file-level resynchronization within the sync window (which we term self-healing) — and we report what becomes observable only at portfolio scope and multi-week duration: behavioral divergence under shared rules, governance recovery from cold start, and cross-machine governance independence. Our claim is empirical and synthetic, not infrastructural: the repository-as-coordination-layer pattern suffices as the sole governance substrate for a non-trivial multi-agent deployment.

1 Introduction

The dominant paradigm for multi-agent AI coordination relies on runtime infrastructure. LangGraph maintains shared state objects passed between agents in a single process [1]. AutoGen uses message-passing protocols between agent instances [2]. CrewAI assigns roles through explicit task delegation at runtime [3]. These approaches share a structural assumption: coordination requires a living process — something running, listening, dispatching.

A counter-current has emerged in 2025–2026. Geng et al. [4] use git worktrees, commits, and branches as the explicit coordination substrate for asynchronous software-engineering agents, reporting a 26.7% absolute improvement on PaperBench. Wu et al. [5] structure a single agent's context as a versioned filesystem with COMMIT/BRANCH/MERGE operations, achieving 80%+ on SWE-Bench Verified. Han & Zhang [6] demonstrate that blackboard-style shared-knowledge coordination is competitive with SOTA multi-agent systems at lower token cost. Concurrently, the

AGENTS.md standard — a repository-rooted markdown file that agent runtimes load at session start as a persistent instruction set — has been adopted across Sourcegraph, OpenAI, Google, Cursor, and Factory and placed under Linux Foundation governance [7]. The repository, in 2026, is no longer an unconventional substrate for agent coordination; it is becoming a standard one.

We are not the first to propose any of the building blocks. The repository-as-state primitive is AGENTS.md and GitOps lineage [8]. Git-as-coordination for multi-agent work is Geng et al. The shared-knowledge-surface primitive is the blackboard tradition (HEARSAY-II [9]) extended to LLMs by Han & Zhang. Tuple-space temporal/spatial decoupling is Linda [10]. Hierarchical agent organization is OrgAgent [11]. What we contribute is **synthesis** and **empirical evidence**.

The empirical regime is the load-bearing contribution. Two autonomous AI agents — one on a WSL2 development machine, one on an RTX 4090 compute machine — have governed a portfolio of 21 software projects for twenty-one weeks (152 days), coordinating through a git-synchronized shared repository. The deployment has produced 264 structured cross-machine handoff notes, 62 architecture decision records, and more than 180 governance standards (internal git log, July 2026). Coordination is brokerless and asynchronous: there is no message broker, shared database, or orchestration service. A 5-minute cron performs git synchronization, and rare urgent messages bypass git via terminal injection (disclosed in §5). Every published prior-art system in this lineage — Geng et al., GCC, Han & Zhang, OrgAgent — is benchmark-only or single-machine. None reports a multi-week, portfolio-scope, cross-machine production deployment. This is the regime we open.

On top of the standard primitive, three architectural patterns emerge from the deployment that have not, to our knowledge, been previously isolated and named:

1. A **federated NEXUS hierarchy** in which each project owns its governance file (`.asif/NEXUS.md`) and a portfolio layer (`PORTFOLIO.md`) aggregates upward — separating per-project ownership from cross-cutting view (§3).

2. An **identity-via-loading-protocol** mechanism: agents do not read each other's identity files not because of filesystem permissions but because the boot protocol never instructs them to. Identity isolation is a property of *what the protocol loads*, not *what the filesystem permits* (§4.1).

3. An observed **eventual file-level resynchronization within the sync window**, which we term *self-healing*: in the single documented instance — a 28-minute window during which no work occurred on the second machine — a governance inconsistency introduced by an unsynchronized commit was resolved automatically on the next scheduled pull. We report file-level resynchronization after the desync window, not a tested behavioral-recovery mechanism (§5.1).

Each is a layered pattern on top of the repository-as-state primitive, not a replacement for it. Each is observable only because the deployment is long-running and multi-machine.

Our central claim is therefore narrower than an earlier draft of this paper made it: not that we invented git-as-coordination, but that we provide the first documented production-scale evaluation of the pattern at portfolio governance scope, and that doing so surfaces architectural patterns and operational behaviors that benchmark-only studies cannot.

2 Related Work

We treat related work in four groups: closest prior art on git-as-coordination (§2.1), the industry-standard repository-rooted-instruction primitive (§2.2), the broader theoretical lineage (§2.3), and the runtime-coordination frameworks we complement (§2.4).

2.1 Closest Prior Art: Git as Multi-Agent Coordination Substrate

Geng et al., *Effective Strategies for Asynchronous Software Engineering Agents* (CAID) [4] is the closest prior art to this work. Geng et al. introduce Centralized Asynchronous Isolated Delegation: software-engineering agents work in isolated git worktrees, coordinate via commits and branches, and merge through a central manager, achieving a 26.7% absolute improvement on PaperBench over single-agent baselines. CAID and ASIF share the central thesis that *git is a sufficient coordination substrate for autonomous LLM agents*. The differences are architectural and scope-defining, not foundational:

- **Coordination scope.** CAID coordinates software-engineering subtasks within a single project; ASIF coordinates governance across a heterogeneous portfolio of 21 projects.
- **Central manager.** CAID requires a centralized manager process to dispatch and merge agent work; ASIF has no central runtime process — the cron job and git itself are the entire coordination layer.
- **Cross-machine.** CAID is single-machine; ASIF spans physical machines and survives extended periods when one machine is unavailable.
- **Synchrony.** CAID's worktrees coordinate concurrent execution within an active session; ASIF coordinates across session boundaries and machine reboots, with sync latency measured in minutes rather than milliseconds.
- **Empirical regime.** CAID is evaluated on PaperBench; ASIF is evaluated on a twenty-one-week production deployment.

These are real but narrower differentiators than we initially framed. A reviewer may reasonably read ASIF as "CAID without a central manager, applied to governance instead of SWE, run on real workloads instead of a benchmark." We accept that framing.

Wu et al., *Git Context Controller* (GCC) [5] structures a single agent's context as a versioned filesystem with COMMIT/BRANCH/MERGE/CONTEXT operations, reporting 80%+ on SWE-Bench Verified. GCC addresses *intra-agent* memory management; ASIF addresses *inter-agent* behavioral consistency. The metaphor — "manage agent state with git" — is shared, and GCC was published first.

2.2 The Repository-Rooted Instruction Primitive: AGENTS.md

AGENTS.md [7] is, as of 2026, a Linux Foundation–governed cross-tool standard. It specifies a markdown file at the repository root that agent runtimes load at session start and use as a persistent instruction set, documenting commands, boundaries, project structure, and business rules. The standard is implemented by Sourcegraph, OpenAI, Google, Cursor, and Factory, with a documented layered policy hierarchy (managed → user → project → local → path-scoped).

The relationship to ASIF is direct and important. ASIF's `CLAUDE.md` is, structurally, an AGENTS.md instance — a repository-rooted markdown file that an agent runtime (Claude Code)

loads at session start as a persistent instruction set. **The primitive is industry-standard. Our contribution is not the primitive.** Our contribution is the application of this primitive across multiple independent agents on different physical machines, drawing behavioral consistency from a single shared file that is propagated by git rather than by per-agent configuration. We make the case that this application is non-trivial and produces emergent properties (§4) not previously isolated, but the primitive itself is widely deployed and well-understood.

2.3 Theoretical Lineage and Adjacent Architectures

Blackboard architectures. Han & Zhang, *Exploring Advanced LLM Multi-Agent Systems Based on Blackboard Architecture* [6], demonstrate that LLM agents coordinating through a shared knowledge surface — read and write to a common blackboard — are "competitive with SOTA static and dynamic MAS while spending fewer tokens." This is the conceptual predecessor to ASIF in the strongest sense: ASIF is, in essence, a **blackboard architecture extended with version control, cross-machine deployment, and governance scope**. Han & Zhang's empirical token-efficiency result is mechanistically consistent with ASIF's observation that file-based coordination scales without runtime overhead. The blackboard tradition itself derives from HEARSAY-II [9] and its descendants.

Tuple spaces. Linda (Carriero & Gelernter, 1989) [10] formalizes the temporal- and spatial-decoupling properties that make file-based coordination viable across sessions and machines. ASIF's HANDOFF protocol maps directly to Linda primitives: `out(t)` (Agent A writes a note), `rd(template)` (Agent B reads non-destructively), `in(template)` (task claims).

Stigmergy. Grassé's biological model of indirect coordination via environmental traces [12] supplies the formal vocabulary for governance propagation: a directive committed to the repository is a trace; agent responses are stimulated actions, requiring no peer-to-peer invocation.

Hierarchical agent organization. OrgAgent (Wang et al.) [11] organizes agents as a company with CEO/CTO/COO governance, execution, and compliance layers. OrgAgent is the closest published architectural analog to ASIF's coordinator / project-agent hierarchy, though OrgAgent is single-machine, synchronous, and benchmark-evaluated. ASIF's federated NEXUS provides the cross-machine async analog.

GitOps and Infrastructure as Code. The discipline of managing system state in version-controlled files — Argo CD, Flux, and the broader GitOps lineage from 2017 forward [8] — is the foundational design principle ASIF applies to agent behavior rather than infrastructure. ASIF can fairly be described as "GitOps for agent governance."

Adjacent. "Everything is Context: Agentic File System Abstraction" (arXiv:2512.05470) treats the filesystem as a context substrate for human-AI co-work; it addresses single-user workflow, not multi-agent governance. Microsoft's Agent Governance Toolkit enforces policy at runtime; ASIF places policy in the repository as static, version-controlled artifacts.

2.4 Runtime Coordination Frameworks We Complement

We do not compete with runtime coordination frameworks; we complement them at a different time scale. **LangGraph** [1] maintains shared state in graph-traversal processes. **AutoGen** [2] uses conversational message-passing between co-resident or network-connected agents. **CrewAI** [3] dispatches work through an active crew orchestrator. **CAMEL** [13] coordinates via role-playing conversational loops. **MetaGPT** [14] organizes agents in a software-company workflow with explicit role specialization. **Google ADK** [15] and **Anthropic MCP** [16] / **Google A2A** [17] standardize

agent-to-tool and agent-to-agent runtime protocols, both donated to neutral foundations in 2025. **Temporal** [18] provides durable execution for long-running agent tasks.

All of these assume coordination is a *process*. The repository-as-coordination pattern is appropriate at a different operating point — minutes-granularity, cross-machine, governance-scope — and the two paradigms are complementary. Tight-loop task decomposition (Geng et al.; LangGraph) and asynchronous governance propagation (ASIF) are not in competition.

3 The ASIF Architecture

3.1 System Overview

ASIF is a portfolio intelligence layer governing 21 software projects across two physical machines:

- **Machine A**: WSL2 development machine. Hosts Agent A (the senior coordination agent) and 8 project teams.
- **Machine B**: RTX 4090 compute machine. Hosts Agent B (coordination agent) and 13 project teams.

Both machines maintain a clone of the ASIF repository. A cron job on each machine runs `sync.sh` every 5 minutes, executing `git pull && git push` to synchronize state.

```
Machine A (Agent A)            Machine B (Agent B)
    |                               |
    +- <repo>/ <---- git sync ----> <repo>/
    |   +- .claude/CLAUDE.md        |   (same files, same rules)
    |   +- machines/HANDOFF.md      |
    |   +- standards/               |
    |   +- PORTFOLIO.md             |
    |                               |
    +--- Claude Code session -------+
         reads CLAUDE.md on start
```

Figure 1: The ASIF two-machine architecture. Both machines hold a clone of the shared governance repository, synchronized by a 5-minute cron; each coordination agent instantiates its behavioral context from the same version-controlled files at session start.

3.2 The Context Loading Mechanism

Claude Code (Anthropic's CLI) automatically loads `.claude/CLAUDE.md` from the working directory at session start — an instance of the AGENTS.md primitive (§2.2). This file contains the governance protocol, standing orders, decision standards, communication formats, and behavioral constraints for any agent operating in the ASIF repository.

When Agent A opens a session in the governance repository root, it reads `CLAUDE.md`. When Agent B opens a session in the same repository root on Machine B, it reads the same `CLAUDE.md` — pulled from the same git origin. Both agents instantiate with identical behavioral context without any runtime message between them.

This is the core mechanism: **behavioral consistency flows from context consistency, not runtime coordination.** The mechanism itself is the AGENTS.md application pattern; the contribution of this work is observing how it scales across multiple independent agents and machines.

3.3 The Communication Protocol

Inter-agent communication uses `machines/HANDOFF.md` — a structured document committed to the shared repository:

```
### Note N -- [Sender] -> [Receiver]: [Title]
From: [Agent] | Date: [ISO timestamp] | Priority: [P0/P1/P2]

[Body -- structured directives, decisions, acknowledgments]
```

The "message bus" is git history. A note written by Agent A on Machine A is committed, pushed, pulled by Machine B's sync cron within 5 minutes, and read by Agent B at next session start. No broker receives the message; a scheduled git pull retrieves it. The commit *is* the message delivery.

3.4 The Federated NEXUS Hierarchy

A pattern that, to our knowledge, has not been previously isolated in the literature: each project in the portfolio owns its own governance file at `.asif/NEXUS.md`, while a portfolio-level layer (`PORTFOLIO.md` in the ASIF repository) aggregates upward. Project agents read only their project's NEXUS; the portfolio layer aggregates status, surfaces cross-project intelligence, and re-injects relevant insights downward.

This **federation of governance scope** — distinct from federated learning — separates ownership (each project owns its NEXUS), aggregation (portfolio layer reads all NEXUSes), and reinjection (cross-project insights flow back down). OrgAgent [11] provides hierarchy in a single process; the federated NEXUS provides hierarchy across heterogeneous repositories and physical machines.

3.5 Governance Propagation

Standards, architectural decisions, and governance rules live in `standards/` and `decisions/` as plain text files. When a new standard is established on Machine A — for example, the Governance Tooling Observability Standard (ADR-032) — it is committed to the ASIF repo. Within 5 minutes, Machine B has the file. Agent B's next session reads it. The standard propagates to the Machine B agent without any runtime notification.

In twenty-one weeks of operation, this mechanism has propagated:

- 62 Architecture Decision Records
- 180+ governance standards
- 264 cross-machine HANDOFF notes
- Portfolio state for 21 projects

(Counts from internal git log, July 2026.) The synchronization machinery has additionally produced more than 10,000 merge commits across both machines; approximately 99% of these are automated sync merges — including the heartbeat-log churn documented in §5.2 — and we therefore report them as operational churn, not as governance volume.

This volume and duration is the empirical regime we open. To our knowledge, no other published system has reported multi-week, portfolio-scope, cross-machine governance propagation through git as the sole substrate.

4 Emergent Properties

The properties below are observable only at the deployment scale and duration described in §3. Benchmark-only systems (Geng et al., GCC, Han & Zhang, OrgAgent) cannot surface them by construction.

4.1 Identity Isolation via Loading Protocol

ASIF maintains a separate identity document for each coordination agent. These files are committed to the shared repository and therefore present on both machines.

An unexpected property emerged: despite having filesystem access to Agent A's identity document on Machine B, Agent B never reads it. The mechanism: Claude Code does not automatically load arbitrary root-level `.md` files — only `.claude/CLAUDE.md`. Agent B's boot protocol (encoded in its machine-local state file, `machines/cos-state-<machine>.md`) references its own identity context, not Agent A's. The loading chain never points Agent B at Agent A's identity document.

This means **identity isolation is a property of the context loading protocol, not access control.** No permissions prevent Agent B from reading Agent A's identity document. Agent B simply never receives an instruction to read it.

This is the most technically distinctive of the three patterns. The mechanism is concrete: multiple agents are instantiated from a shared repository, with each agent's identity determined by a context-loading sequence specified in a per-agent state file, such that filesystem accessibility is decoupled from behavioral influence. Isolation is a property of what the protocol loads, not of what the filesystem permits.

4.2 Behavioral Divergence Under Shared Rules

Both agents read identical governance rules from `CLAUDE.md`. Despite this, observable behavioral differences have emerged over twenty-one weeks:

- Agent A tends toward editorial caution and cross-project correlation; Agent B tends toward execution speed and parallel agent deployment.
- Agent A's HANDOFF notes are longer and more analytical; Agent B's are more directive and action-focused.
- Agent A enforces governance more conservatively; Agent B ships faster with more post-hoc review.

We report these as qualitative patterns observed by the operator and in review of the artifact record, not as quantified measurements.

This divergence arises from the stochastic nature of language model inference — identical prompts produce different outputs at each call, and accumulated divergence across thousands of interactions produces distinct behavioral profiles. The shared CLAUDE.md provides a common rulebook; it does not produce identical agents.

This is desirable. Identical agents provide no diversity benefit. The shared context enforces consistency where consistency matters (communication formats, governance standards, decision protocols) while permitting divergence where divergence is valuable (operational style, priority weighting, execution approach). The pattern is observable only over multi-week duration; short benchmark runs cannot exhibit accumulated divergence.

4.3 Cross-Machine Governance Independence

A property unique to the multi-machine deployment regime: when one machine is unavailable for extended periods (planned maintenance, network partitions, or simply session inactivity), the other machine continues to operate under the most recently synchronized governance state. New decisions made during the partition are queued in commits and propagate on the next successful sync. There is no degradation of governance behavior on either side during the partition; there is only latency in the propagation of new standards.

We have observed this in multiple ad-hoc partitions during the twenty-one-week deployment. A reviewer might object that this property is trivially inherited from git's distributed model. We agree it is inherited; we observe that it is also empirically demonstrated in a multi-agent governance setting, which has not previously been reported.

4.4 Governance as Repository Property — Cold-Start Recovery

A new agent session — with no conversation history, no prior context, no knowledge of ASIF — can open the governance repository root and immediately behave as a governed agent. The governance is in the files. The agent reads the files. Governance instantiates.

This was validated empirically: on three occasions when context compaction eliminated all session history, fresh Agent A sessions recovered full governance behavior within 3–5 tool calls following a documented cold-start recovery protocol — reading `CLAUDE.md`, the machine-local state file (`machines/cos-state-<machine>.md`), the agent memory index (`MEMORY.md`), and `machines/HANDOFF.md` in sequence. No human intervention required. No re-briefing.

5 Limitations and Failure Modes

5.1 Synchronization Latency and Self-Healing

The 5-minute sync interval means coordination has a nominal maximum latency of 10 minutes — one push interval plus one pull interval — when both machines' sync jobs run on schedule; a delayed or missed sync cycle extends the window in practice, as the 28-minute divergence documented below illustrates. This is acceptable for governance-layer communication but unsuitable for tight-loop coordination. The system cannot support real-time collaboration between agents.

Observed impact: One documented case where Agent B issued a directive before Agent A's counter-directive had synced, resulting in conflicting actions. Resolved by establishing a "wait for ACK" convention in the HANDOFF protocol.

Ablation observation — CLAUDE.md consistency window: On 2026-04-15, when the ADR-033 governance rule was added to `CLAUDE.md` on Machine A (commit `9063b5b51`, 18:02 UTC),

the change was unavailable to Machine B for 28 minutes — until Machine B's scheduled pull at 18:30 UTC (verified from git log: Machine B auto-sync commits at 18:30 UTC and 18:35 UTC bracket the window). During this window, any Agent B session starting would have operated under the prior escalation pattern. No Agent B commits were detected in this window (zero Machine B non-sync commits between 18:02 and 18:30 UTC), and the inconsistency resolved automatically on the next scheduled sync without human intervention. This makes the consistency boundary empirically concrete: CLAUDE.md diverges in direct proportion to the sync lag, and the 5-minute polling cadence bounds but does not eliminate the desync window. What this demonstrates is narrower than behavioral self-healing: because no Agent B work occurred in the 28-minute window (zero Machine B non-sync commits), it shows **eventual file-level resynchronization within the sync window** on the next scheduled pull, not that any divergent agent behavior was detected and corrected. We use *self-healing* as shorthand for this within-window resynchronization.

5.2 Merge Conflicts

Concurrent writes to shared files create merge conflicts. The system addresses this through:

- Machine-partitioned write ownership (Machine B writes Agent B notes; Machine A writes Agent A notes)
- Append-only JSONL formats for high-frequency files (promises, observations)
- Explicit gitignore entries for machine-local state files

Observed impact: The heartbeat log file (`heartbeat/heartbeat.log`) produced 38 conflict commits and 5,800 merge commits before being added to `.gitignore`. Partitioned ownership was the correct fix.

5.3 No Real-Time Signals

The system cannot notify a running agent of new messages. Agent B's session may run for hours before ending; Agent A's HANDOFF note written during that time is not delivered until Agent B's next session start. For time-sensitive coordination, this latency is a real limitation.

Partial mitigation: A terminal-multiplexer key-injection facility allows direct injection into a running session's terminal, bypassing git entirely for urgent messages. This is used sparingly for P0 escalations.

5.4 Trust Is Implicit

Any process with write access to the ASIF repository can modify governance rules. There is no cryptographic signing of governance documents. An adversarial commit to `CLAUDE.md` could alter agent behavior without agent awareness.

Mitigation: Git history provides full auditability. Pre-commit hooks and CI gates validate structural integrity. The attack surface is equivalent to any shared configuration repository — analogous to the supply-chain risk of any AGENTS.md file under shared write access.

6 Discussion

6.1 When to Use This Pattern

Repository-based coordination is appropriate when:

- Coordination granularity is measured in minutes, not milliseconds
- Agents operate asynchronously (different sessions, different machines)
- Governance consistency matters more than execution speed
- Teams want auditable, version-controlled coordination history
- Infrastructure simplicity is a priority

It is inappropriate when:

- Agents must coordinate in real-time within a single task (use Geng et al. CAID, LangGraph, or AutoGen)
- Low-latency message passing is required (use MCP / A2A runtime protocols)
- Agents share state that mutates faster than the sync interval

6.2 Precision on "Federated"

This work uses "federated" in a sense that differs from federated learning. In classical FL, federation means multiple participants collaborating under a shared training protocol while data remains local — there is typically still a coordinating server, and the unit of federation is datasets and model updates. ASIF federates differently: autonomous agents retain local execution authority while sharing a governance protocol. Each agent executes locally, maintains local state, and reports compliance through local artifact writes. The federation is of **scope authority and governance context**, not of gradient exchange.

A more precise description: **centrally governed, locally autonomous, and operationally asynchronous**. Central governance (the shared CLAUDE.md and directive protocol) provides behavioral consistency. Local autonomy provides operational independence. Asynchronous operation (git-synced at 5-minute intervals) provides resilience to connectivity loss.

6.3 What the Empirical Regime Adds

The closest prior-art systems (§2.1) are evaluated on benchmarks: PaperBench (Geng et al.), SWE-Bench Verified (GCC), reasoning datasets (OrgAgent), token-efficiency comparisons (Han & Zhang). Benchmarks cannot exhibit:

- Behavioral divergence across thousands of inferences over weeks
- Cross-machine governance independence under partition
- Governance-recovery from cold start across compaction events
- Self-healing within a sync window
- The operational cost profile of running governance through git for twenty-one weeks (more than 10,000 merge commits, approximately 99% of them automated sync merges; 38 conflict commits in one file before mitigation; etc.)

These observations are the contribution of long-running production deployment. They do not replace benchmark evaluation; they complement it. We argue that the multi-agent coordination literature would benefit from more deployments of this kind, and we describe ASIF in sufficient detail that other groups can reproduce the pattern.

6.4 Implications for the Broader Field

A recent large-scale survey of agents in production finds that 68% execute at most 10 steps before human intervention and that reliability — not capability — is the top development challenge [19]. The ASIF deployment is consistent with this finding: governance-first, auditable architectures match how stable production systems are actually built. The deeper implication is that the *repository* — not the *agent* — is the unit of behavioral consistency. Organizations deploying multiple AI agents should invest as much care in their shared context files as they invest in prompt engineering. The governance files are the architecture. The agents are the execution layer. AGENTS.md is the substrate; what is built on top of it is open.

7 Conclusion

We have presented the ASIF architecture — a production multi-agent system achieving behavioral consistency across two agents on separate machines for twenty-one weeks (152 days) with brokerless, asynchronous coordination: no message broker, shared database, or orchestration service, only a 5-minute git-synchronization cron (with rare urgent messages bypassing git via terminal injection, per §5). The mechanism is not novel: shared context files in a version-controlled repository, in the lineage of AGENTS.md (industry standard), Geng et al. CAID (closest prior art), and Han & Zhang Blackboard (conceptual predecessor). What we contribute is the **first documented production-scale evaluation** of this lineage at portfolio governance scope, and three architectural patterns that emerge from that evaluation: federated NEXUS hierarchy, identity-via-loading-protocol, and self-healing within the sync window.

Our central claim is empirical: an industry-standard primitive — repository-rooted markdown loaded at agent session start — suffices as the sole governance substrate for a non-trivial multi-agent deployment, and the patterns visible at this scale and duration are not visible in benchmark-only studies.

The repository is the coordination layer. We are not the first to say so. We are, to our knowledge, the first to document it in production at this scale.

8 Broader Impacts

This work has several positive societal implications. First, it reduces the infrastructure barrier to multi-agent governance: teams without dedicated DevOps capacity can deploy consistent multi-agent behavioral frameworks using tools they already have, democratizing AI governance beyond well-resourced organizations. Second, the architecture provides an audit trail by default — git history is a forensic record of every governance decision, enabling accountability and post-incident review without additional logging infrastructure. Third, the observational data from twenty-one weeks of production operation contributes to empirical understanding of how LLM agents behave under shared behavioral constraints over extended periods, a data regime that synthetic benchmarks cannot replicate.

The democratization implication deserves direct emphasis. Current multi-agent governance frameworks — LangGraph, AutoGen, CrewAI — require infrastructure teams to deploy and maintain runtime services: container orchestration, message broker management, state persistence layers, monitoring stacks. This infrastructure requirement creates a structural asymmetry. Repository-based coordination eliminates this asymmetry. A team with git access and a cron job has all the infrastructure the ASIF pattern requires.

The primary risk is adversarial: the same simplicity that enables legitimate governance enables adversarial behavioral manipulation if repository write access is compromised. A malicious actor with push access to the shared CLAUDE.md could alter agent behavioral rules without immediate agent awareness. At scale — if repository-based governance is widely adopted — this attack surface expands: a compromised canonical AGENTS.md or shared CLAUDE.md template could propagate adversarial behavioral modifications to many independent deployments simultaneously, a supply-chain risk analogous to malicious npm packages or compromised CI pipeline actions. We note this risk is equivalent to the profile of any shared infrastructure-as-code repository, and the mitigations (branch protection, commit signing, CI validation gates, organizational source verification) are well-established. The paper does not introduce new attack surface; it makes the existing shared-configuration attack surface visible and therefore addressable.

Data Availability

The deployment repository underlying the metrics in this paper is a proprietary internal repository and is not publicly available. All reported counts — handoff notes, architecture decision records, governance standards, merge-commit totals, and the §5.1 timing observations — are drawn from its internal git logs as of July 2026 and are therefore not externally verifiable. The architecture, file formats, and synchronization protocol are described in sufficient detail (§3, Appendices A–B) for independent reproduction of the pattern.

Acknowledgments and AI-Assistance Disclosure

The multi-agent system described in this paper is designed, operated, and supervised by the author. The coordination agents studied ("Agent A" and "Agent B") are large-language-model agents (Anthropic Claude models operated via the Claude Code runtime); they are the object of study, not authors. In accordance with arXiv's policy on generative-AI authorship, no AI system is listed as an author. Portions of the manuscript text were drafted and revised with the assistance of large-language-model tools under the author's direction; the author has reviewed all content and takes sole responsibility for it.

References

[1] Harrison Chase et al. Langgraph: Building stateful, multi-actor applications with LLMs. LangChain, 2024. URL https://langchain-ai.github.io/langgraph/.

[2] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation, 2023.

[3] João Moura. Crewai: Framework for orchestrating role-playing, autonomous AI agents. GitHub, 2024. URL https://github.com/joaomdmoura/crewAI.

[4] Jiayi Geng and Graham Neubig. Effective strategies for asynchronous software engineering agents, 2026. CAID.

[5] Junde Wu et al. Git context controller: Manage the context of LLM-based agents like git, 2025.

[6] Bochen Han and Songmao Zhang. Exploring advanced LLM multi-agent systems based on blackboard architecture, 2025.

[7] AGENTS.md contributors. AGENTS.md: A cross-tool standard for repository-rooted agent instructions. Linux Foundation cross-tool standard; adopted by Sourcegraph, OpenAI, Google, Cursor, Factory, 2026. URL https://agents.md/.

[8] Weaveworks and others. Gitops: Operations by pull request. 2017–2026 lineage; Argo CD, Flux, 2017. URL https://www.firefly.ai/academy/beyond-provisioning-a-2025-guide-to-infrastructure-orchestration-with-iac-and-gitops.

[9] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys*, 12(2):213–253, 1980.

[10] Nicholas Carriero and David Gelernter. Linda in context. *Communications of the ACM*, 32(4): 444–458, 1989.

[11] Yizhou Wang et al. Orgagent: Organize your multi-agent system like a company, 2026.

[12] Pierre-Paul Grassé. La reconstruction du nid et les coordinations inter-individuelles chez *Bellicositermes natalensis* et *Cubitermes* sp. la théorie de la stigmergie. *Insectes Sociaux*, 6(1): 41–80, 1959.

[13] Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society, 2023.

[14] Sirui Hong et al. MetaGPT: Meta programming for a multi-agent collaborative framework, 2023.

[15] Google. Agent development kit (ADK), 2025. URL https://google.github.io/adk-docs/.

[16] Anthropic. Model context protocol (MCP). Donated to Agentic AI Foundation, December 2025, 2024. URL https://modelcontextprotocol.io.

[17] Google. Agent2agent protocol (A2A). Donated to Linux Foundation, June 2025, 2025. URL https://a2a-protocol.org/.

[18] Temporal Technologies. Temporal: Open source durable execution system, 2024. URL https://temporal.io.

[19] Melissa Z. Pan et al. Measuring agents in production, 2025.

A ASIF Governance Files Reference

B Sync Protocol

```
# sync.sh -- runs every 5 minutes via cron on both machines
git pull origin main --no-rebase 2>/dev/null
git add -A
git commit -m "auto-sync $(hostname) $(date '+%Y-%m-%d_%H:%M')" 2>/dev/null || true
git push origin main 2>/dev/null
```

File	Purpose	Update frequency
`.claude/CLAUDE.md`	Agent behavioral rules (AGENTS.md instance)	Per-session, curated
`machines/HANDOFF.md`	Cross-machine communication log	Per significant event
`PORTFOLIO.md`	Portfolio state dashboard	Per enrichment cycle
`.asif/NEXUS.md` (per project)	Per-project federated governance file	Per project event
`standards/*.md`	Governance standards and protocols	Per ADR
`decisions/*.md`	Architecture Decision Records	Per architectural decision
Agent identity documents	Per-agent identity (one per agent)	Self-directed, periodic
`heartbeat/heartbeat.conf`	Operational configuration	Per tuning session

Table 1: Governance files in the ASIF repository.

The coordination layer is 6 lines of shell script on top of an industry-standard primitive.