Intent, Custody, Trajectory, and Proof:
Toward Accountable Execution in Agentic Software Engineering

Alex H. Raber

Decapod Labs

alexhraber@gmail.com

July 13, 2026

Abstract

Large language model (LLM) agents are transitioning from conversational programming assistants to autonomous executors capable of editing codebases, running build tools, executing tests, and managing multi-file software deployments. Despite this shift, many coding-agent workflows still rely heavily on prompt transcripts and code diffs as durable records. As runs become concurrent, long-running, and tool-intensive, this chat-centered baseline exposes repositories to lost intent, cross-runtime discontinuity, ambiguous ownership, duplicated inference, workspace and integration failure, and unverifiable completion claims.

This paper frames agentic software engineering as a distributed execution problem and asks whether one complex natural outcome-oriented request can support "type the outcome and walk away." We introduce four primitives: *Intent* (human/team desired outcome and its governed resolutions), *Custody* (task ownership and bounded workspace), *Trajectory* (event-backed governance records), and *Proof* (machine-checkable validation and provenance evidence). We present Decapod, a daemonless, local-first governance kernel, as a reference implementation and define a primary paired comparison under the identical natural prompt. We further define *fleet coherence*: the bounded ability of independently launched, heterogeneous agents to use one durable project authority, custody model, governance trajectory, and proof boundary across sessions, handoffs, and similar-but-distinct concurrent work. Decapod is also offered as a candidate agent-governance interoperability profile that coding-agent harnesses could bundle or discover; current MCP, A2A, HTTP, and organization-policy integration remains prospective. Secondary studies test handoff continuity, duplicate-work prevention, tool switching, and integration; a separate longitudinal snapshot establishes ecological use without a causal claim. The artifact remains in harness validation: checked-in controlled records are synthetic. The hypotheses concern system-level operational alignment and coordination at a fixed model, not changed model intelligence, complete sandboxing, conflict-free merging, or global consensus.

1 Introduction

The integration of Large Language Models (LLMs) into software engineering has progressed rapidly through three distinct phases: suggestion (e.g., inline autocompletion), instruction (e.g., conversational chat interfaces), and delegation (e.g., autonomous agents).

In this current phase of delegated execution, an agent is not merely a generator of code snippets; it is an active developer. Equipped with a tool belt of shell executors, file readers, linters, compilers, and test suites, the agent operates directly on target repositories, running iterative loops to edit files, compile binaries, parse build logs, and debug errors.

However, as agentic workflows evolve from simple scripts to long-running, concurrent, multi-agent systems, their durable coordination mechanisms remain underdeveloped. Many contemporary workflows still depend heavily on the natural-language prompt, one workbench conversation, and the final git diff. A session or tool switch therefore changes not only the executing process but often the available project memory, ownership state, and evidence.

We argue that this transcript-only paradigm is fundamentally inadequate for serious software engineering environments. Chat transcripts are verbose, non-deterministic, and prone to context window drift. More critically, they lack clear boundaries of authority, fail to capture environment validation results in a machine-checkable format, and provide no guarantees that code changes satisfy the intended constraints. Once an agent is granted write access to a repository and the authority to run arbitrary CLI tools, software engineering transforms from a language-modeling problem into a **distributed execution and security problem**.

When viewed through a systems-oriented lens, agentic software engineering shares core challenges with classic distributed systems:

- **Authority & Isolation**: How do we ensure that an agent only mutates the directories it is authorized to touch, without leaking credentials or accessing sensitive environment variables?
- **Concurrency**: How do we reduce duplicate task starts, isolate mutations, respect dependencies, and detect integration failure without pretending worktrees guarantee compatible merges?
- **Failure, Handoff & Recovery**: If an agent runs out of tokens, changes workbenches, or crashes mid-task, how can a subsequent process recover intent, unresolved state, ownership, and proof without replaying one vendor conversation?
- **Auditability**: How can a human developer efficiently verify that an agent's changes are correct without manually reviewing thousands of lines of conversational LLM reasoning?

To address these challenges, this paper formalizes four core primitives for accountable execution:

1. **Intent**: The human or team's desired outcome, motivation, constraints, priorities, unresolved questions, and completion standard; governed artifacts are durable projections rather than the source of authority.
2. **Custody**: Exclusive task ownership and isolated git-worktree or container boundaries within which the agent operates.
3. **Trajectory**: Event-backed records of task, broker, proof, and governance activity that can be rendered as a flight-recorder timeline.
4. **Proof**: Programmatically checkable validation, proof-run, workunit, and provenance evidence; cryptographic signing is a future extension, not an implemented claim here.

We present **Decapod**, a daemonless, local-first governance kernel, as a reference implementation of this model. The primary experiment remains one-shot walk-away delegation: whether, under the same complex natural request, pre-inference resolution of authoritative context lets an unchanged model converge toward the intended outcome with fewer failed cycles and less human intervention. We also develop a second systems thesis: Intent, Custody, Trajectory, and Proof may produce *fleet coherence*, allowing independently launched agent processes to share durable project authority, transfer work, avoid duplicated cognition, and publish against common evidence without sharing one conversation or long-running orchestrator. Agents and conversations are ephemeral execution processes; the governed repository is the durable coordination subject.

The central fleet question is whether multiple agents can concurrently solve similar but distinct problems in one codebase while preserving common authority, distinct task custody, isolated mutation, dependency awareness, and integrated proof. The human-interface goal is complementary: a person speaks naturally to the available agent, while the agent invokes a stable governance contract. We offer that contract as a candidate agent-governance interoperability profile that coding-agent harnesses could pre-bundle or discover. The current system is not an adopted standard or a native MCP, A2A, or HTTP implementation; these protocols provide concrete comparison points for a future thin waist across Claude, Codex, Antigravity, Grok, and other conforming workbenches.

The implementation mapping is pinned to Decapod `207e9a3` (`v0.66.3`). Prospective Studies B–D test handoff, concurrent work, and tool switching; Study E is a non-causal longitudinal case study. The contribution remains pre-results: mechanisms, boundaries, schemas, and falsifiable protocols rather than measured treatment effects.

2 Background and Lineage

The transition from static algorithms to autonomous software engineering agents is underpinned by a century-long chain of intellectual developments in computation, information, learning, scale, and distributed systems. Understanding this lineage is essential to framing agentic software engineering not as an ad-hoc scripting trend, but as a formal systems execution domain.

2.1 Formalizing the Machine and the Message

The mathematical foundation of computing begins with Turing's formalization of computation in 1936 [22]. By introducing the Turing Machine, Alan Turing defined the limits of what can be computed algorithmically and established that a universal machine could simulate any other process given a discrete set of rules and an infinite tape.

Twelve years later, Shannon defined what flows through these machines [20]. In *A Mathematical Theory of Communication*, Claude Shannon stripped meaning from language, formalizing information as a measurable quantity of entropy and establishing the bit as the core unit of channel capacity. This mathematical framing of entropy and prediction under uncertainty directly underpins the loss functions and token-generation architectures used in modern language models.

2.2 The Connectionist Evolution and Distributed Ordering

Early attempts to make machines learn from their environments began with Rosenblatt's Perceptron in 1958 [18], which showed that machines could adjust numerical weights

to classify input patterns. However, Minsky and Papert's critique in 1969 [13] exposed the geometric limitations of single-layer networks, notably their inability to learn non-linearly separable functions like XOR. This critique initiated the first AI winter, which was resolved seventeen years later by Rumelhart, Hinton, and Williams [19] who formalized backpropagation, showing that stacking layers of perceptrons and using the chain rule from calculus allowed multi-layered networks to learn complex internal representations.

Concurrently, the systems substrate of computing was expanding from single processors to distributed networks. Lamport's seminal 1978 paper [12] established that without a shared physical clock, events in a distributed system must be ordered based on causality (the "happens-before" relation) rather than wall-clock time. This conceptual breakthrough made large-scale distributed databases, cloud storage, and eventually parallel GPU training clusters possible, preventing concurrent machines from dissolving into state inconsistency.

2.3 Scale, Attention, and Emergent Intelligence

In 1998, Brin and Page demonstrated that relevance could emerge from planetary-scale graph structure through the PageRank algorithm [2]. This proved that indexing and ranking massive piles of human text and links could yield high-quality search interfaces. By 2012, the ImageNet breakthrough (AlexNet) by Krizhevsky et al. [11] proved that connectionist backpropagation, when combined with massive datasets and consumer-grade graphics processing units (GPUs), dramatically outperformed hand-engineered features in computer vision.

The modern era of LLMs was ignited by Vaswani et al. in 2017 [23] with the Transformer architecture. By discarding sequential recurrent networks in favor of parallelizable attention mechanisms, the Transformer allowed models to compute associations between all tokens in a context window simultaneously. Finally, in 2020, Brown et al. showed in GPT-3 [3] that scaling these transformers to hundreds of billions of parameters transforms language predictors into general-purpose few-shot learners. GPT-3 proved that scale turns predictions into programmable interfaces.

2.4 The Agentic Transition and the Accountability Gap

While GPT-3 and subsequent models excel at static text completion, software engineering requires active intervention. This transition from static representation to dynamic action was foreshadowed by Mnih et al. in Deep Q-Networks (DQN) [14], where neural networks were integrated into closed-loop reinforcement learning environments—mapping perception directly to actions and environment feedback.

Today, LLM agents combine the semantic reasoning of transformers with the action loops of reinforcement learning, executing commands via shell tools over mutable code repositories. Yet, as computation has transitioned from Turing's abstract tape, through Shannon's bits, Lamport's clocks, and Transformer attention, we have reached a critical systems gap: **we have executable intent, but we lack custody and proof**. This paper bridges this gap, developing the control plane primitives needed to make autonomous agent runs safe and accountable.

3 Problem Formulation

The standard model of agentic software engineering often relies on a chat-centered workflow. The agent is initialized with a system prompt and a user command, interacts with files and commands through tool wrappers, and returns a conversational transcript and repository diff. We first retain five single-run failure modes, then distinguish failures that arise when independently launched agents share a project without durable coordination.

3.1 Intent Loss (State Loss Under Interruption)

When an agent's process is interrupted due to network timeouts, model rate limits, or crash events, the current state of its work is lost. In a transcript-only workflow, there is no separate, structured record of progress. Resuming the task requires either:

1. Replaying the entire transcript from the beginning, which is slow and expensive.
2. Launching a new agent in the modified workspace. Because the new agent has no access to the previous agent's reasoning, it must guess the intent from the mutated files. This often leads to the agent reverting partially finished edits or introducing redundant code.

3.2 Context Drift (Loss of Task Focus)

LLMs operate within a finite context window. As an agent runs multi-step tasks (e.g., executing commands, reading compiler stack traces, parsing log files), the context window becomes saturated with terminal stdout/stderr. As the initial prompt and instructions are pushed out of the active context, the agent suffers from context drift: it forgets original constraints, ignores boundary rules, and enters infinite loops of repeating the same failing tool commands.

3.3 Authority Leakage (Unbounded Privileges)

Chat-centered workflows do not themselves partition credentials or system permissions. An agent commonly inherits the parent runtime's authority unless the workbench supplies a stronger boundary. Repository worktrees can isolate mutation surfaces, but they do not establish credential isolation or a complete operating-system sandbox. A governance layer must therefore state exactly which mutations it gates and which runtime authority remains outside its control.

3.4 Concurrent Workspace Collision (State Inconsistency)

In active development environments, multiple agents or human engineers work concurrently on the same codebase. In a transcript-only workflow operating directly on a single shared repository directory, concurrent edits cause state corruption. If Agent A compiles the code while Agent B is modifying a dependency, the build output is non-deterministic. Furthermore, without logical branch isolation, agents overwrite each other's files, leading to silent code regressions and complex merge conflicts.

3.5 Unverifiable Completion (False Success Claims)

Agents can declare completion while the repository remains broken or incomplete. Without an independent, machine-checkable verification gate tied to the governed state, these false-positive assertions require humans to compile, test, and reconstruct the evidence themselves.

3.6 Cross-Runtime Context Discontinuity

A new session or workbench does not inherently inherit another provider's conversation. Relevant decisions may exist, yet remain trapped in a prior chat. This *cross-runtime discontinuity* differs from context-window drift: the lost state was available to another process, not merely displaced inside the current one. The receiver reconstructs intent, progress, and uncertainty from diffs, summaries, or guesses.

3.7 Ownership Ambiguity and Duplicated Inference

Without durable claims, multiple agents may investigate the same failure, reread the same architecture, run the same tests, or produce competing implementations. This is waste before a textual conflict occurs. Branch isolation alone does not tell an agent whether another process already owns the task or a prerequisite.

3.8 Handoff Loss and Project-Memory Fragmentation

A commit or diff does not necessarily preserve why a task exists, rejected alternatives, unresolved questions, dependency state, validation already performed, missing proof, or the reason execution stopped. Vendor conversations and personal session memory further fragment knowledge that should belong to the project. A handoff protocol must preserve durable references while acknowledging that hidden reasoning and complete transcripts may remain unavailable.

3.9 Workbench Dialect Fragmentation

Agent platforms expose different slash commands, memory behavior, lifecycle controls, and orchestration conventions. Humans can be forced to translate the same desired outcome into provider-specific operational rituals. We call the separation of natural human delegation from those machine-facing governance calls *operator decoupling*: the agent uses a stable control-plane contract at governance boundaries. This does not imply that vendor commands cease to exist.

3.10 Integration, Publication, and Governance-State Races

Separate worktrees prevent direct file trampling but do not guarantee compatible designs. Multi-agent analysis must distinguish mutation collision, overlapping ownership, dependency-order violation, textual merge conflict, semantic integration failure, task-local proof, post-integration proof, and publish authorization. Related agents may also resolve inconsistent context or stale projections. A local validation pass is insufficient when the integrated state violates a dependency or publication gate.

4 Formal Execution Model

To resolve the five failure modes identified in Section 3, we formalize an execution model built around Intent, Custody, Trajectory, and Proof. This model abstracts agentic software engineering as a state transition system operating over a repository space.

Let R represent the state of the repository, including the files, directory structure, git history, and runtime environment. An agent run is a sequence of state transitions $R_0 \xrightarrow{a_1} R_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} R_n$, where each action a_i is executed by the agent via tools.

4.1 Intent: Desired Outcome and Resolution

Intent (I) is the human or team's desired outcome, why it matters, the constraints and priorities that must remain true, unresolved questions, stop conditions, and standard of completion:

$$I = \langle O, M, C, Q, S, A \rangle$$

where O is the desired outcome, M its motivation, C constraints and priorities, Q unresolved questions, S stop conditions, and A acceptable completion. A prompt is an initial and often incomplete expression of I. Plans, todos, specifications, context capsules, proof hooks, and rubrics are governed projections or resolutions of intent, not its source of authority. Decapod distributes these representations across repository artifacts so intent can be re-resolved against repository state and human escalation points without requiring the initial prompt to contain a complete procedural specification.

Before implementation inference, the governed system resolves a context set $X = \langle U, A_e, E, D, Q_e, V \rangle$, where U contains selected authority and constitution nodes, A_e contains architecture and project specifications, E contains capabilities and constraints, D contains prior decisions and applicable standards, Q_e contains unresolved questions and escalation rules, and V contains validation and proof expectations. Each supplied item has an authority source, selection rationale, content hash, and relevance record. The primary experiment tests this context-alignment mechanism against a conventional run receiving the identical natural prompt, not a different model.

4.2 Custody: Environment Bounding

Custody (K) defines the execution and security sandbox for the agent. Formally, it is a tuple:

$$K = \langle W, B, A, P \rangle$$

where:

- $W \subset R$ is the isolated workspace slice (a task-scoped git worktree and, when configured, a container).
- B is the task-specific branch locked for the agent.
- A is the set of governance operations and proof capabilities available to the run.
- P represents the repository and runtime boundary enforced by the workspace configuration.

By enforcing custody boundaries, the kernel keeps the agent's repository mutations in W and prevents direct work on protected branches. This reduces cross-task workspace collisions; it is not a complete operating-system sandbox or credential-isolation proof.

4.3 Trajectory: The Governance Record

The Trajectory (T) is an event-backed sequence of governance records:

$$T = [t_1, t_2, \ldots, t_n]$$

where a record may include timestamp, operation, actor, session, correlation identifier, status, and structured details. Decapod journals todo, broker, proof, and related governance events and can render them through its flight recorder. These records improve reconstruction and review, but they are not asserted here to be a complete capture of every model token, shell byte, file diff, or tamper-proof execution event.

4.4 Proof: Completion Evidence

A Proof (P_f) is the set of machine-checkable evidence attached to a governed task:

$$P_f = \langle I, R_n, \{\text{Pass}(v_j)\}, E \rangle$$

where E contains relevant proof events, workunit results, validation output, and provenance references. In Decapod, validation and proof gates can block promotion or completion paths when required evidence is absent. A proof record establishes what the configured gates observed; it does not establish semantic correctness beyond those gates, and this paper makes no claim that the current implementation signs a universal certificate.

4.5 Heterogeneous Multi-Agent Layer

The single-run transition sequence is insufficient for independently launched processes. Let $\mathcal{A} = \{\alpha_1, \ldots, \alpha_m\}$ be agent processes, $\mathcal{M}$ the underlying models, $\mathcal{W}$ the workbenches or runtimes, and $G = (V, E_G)$ a governed task/dependency graph. Each agent has a model/workbench pair, but neither is the durable project subject. We define:

$o : V \to \mathcal{A} \cup \{\varnothing\}$	primary task ownership,
$w : V \to \mathcal{S}$	task-to-workspace assignment,
$C(v)$	resolved authoritative context,
$T(v)$	accumulated governance trajectory,
$P(v)$	proof and validation state,
$H(v, \alpha_i, \alpha_j)$	governed ownership transfer,
$\rho(v_i, v_j)$	audited task relationship,
Q	integration and publication state.

The relation ρ distinguishes duplicate, independent, dependent, and similar-but-distinct tasks. The last category shares architecture, modules, tests, or integration boundaries without sharing one desired outcome. It is a prospective experimental annotation, not an implemented Decapod oracle. The repository state includes the durable representations of G, o, w, C, T, P, and Q; a workbench conversation does not. A handoff H changes ownership and records durable transfer information while the receiving process resolves the current repository state. It cannot transfer inaccessible hidden reasoning.

4.6 Qualified Fleet Invariants

The implementation audit distinguishes enforced, conditional, and desired properties:

1. **Exclusive primary custody (enforced for exclusive mode):** an atomic exclusive claim conflicts with another active primary assignee. Trust-gated shared secondary ownership exists, so uniqueness is not unconditional.
2. **Workspace custody (enforced on governed paths):** a todo-scoped worktree is required and protected/root checkout work is blocked. External tools can bypass Decapod, and containers are configuration-dependent.
3. **Dependency awareness (represented and partially gated):** dependencies are durable and replayable; this is not a globally serializable scheduler.
4. **Semantic-overlap awareness (desired, not enforced):** similar-but-distinct tasks should remain concurrent when safe and coordinate when their integration surfaces interact. Current task claims do not infer this relation automatically.
5. **Handoff continuity (implemented but bounded):** approved ownership transfer preserves task assignment, summary/event, durable knowledge, and available repository references. Complete conversation state, unresolved-state completeness, and merge success are not guaranteed.
6. **Provider-independent project state (implemented as an external contract):** generated entrypoints route multiple workbenches to one repository contract. Compliance by every workbench is empirical, not enforced by Decapod.
7. **Proof-gated publication (enforced on the Decapod publish path):** provenance manifests and configured workunit/evaluation gates are required. Task-local evidence remains insufficient if post-integration gates fail or are absent.
8. **Projection consistency (enforced for declared surfaces):** validation detects known derived-state drift. This is not global distributed consensus.

We define *fleet coherence* as the degree to which these bounded properties let heterogeneous processes preserve common authority, avoid conflicting custody, reconstruct relevant state, and share an acceptance boundary. It is an emergent property of Intent, Custody, Trajectory, and Proof, not a fifth primitive.

5 Decapod: A Reference Governance Kernel

Decapod is a daemonless, local-first governance kernel for AI coding agents. It is invoked by agents at governance boundaries; it does not replace the model's tool loop or alter model weights. The implementation mapping is pinned to commit `207e9a3` (tag `v0.66.3`); the companion claim matrix lists command, source, test, status, boundary, and planned measure for each research concept. The changes from the previously inspected `v0.66.1` revision through this release are release, CI, and research-link documentation changes; the mapped implementation and test paths are unchanged.

5.1 System Architecture

The CLI and Decapod-specific structured process-RPC surfaces route requests into core subsystems that persist state under `.decapod/`. The principal surfaces relevant to this paper are:

1. **Intent and coordination**: governed plans, todos, specifications, context capsules, knowledge, and proof hooks progressively resolve human intent. The todo subsystem records claims, tags/categories, dependencies, presence, owners, and approved handoffs; none of these projections replaces human or team authority.
2. **Custody**: workspace management creates todo-scoped git worktrees and, when configured, container workspaces. Protected branches and workspace checks prevent an agent from treating the human root checkout as its execution surface.
3. **Trajectory**: todo, broker, proof, federation-memory, and related event journals are rendered by the flight recorder as a timeline or transcript. This is a selected governance record, not a complete capture of all model or shell activity.
4. **Proof and validation**: configurable proof commands produce structured results and proof events; `decapod validate`, workunit gates, provenance artifacts, and verification checks supply promotion evidence.

The generated specs describe the same corridor as intent $\rightarrow$ claim $\rightarrow$ workspace $\rightarrow$ proofs $\rightarrow$ publish. The public crate's cloud backend explicitly reports unavailable; cloud-mode initialization writes only an experimental registration payload while local state remains usable. Remote leases, synchronization, and cloud handoff are therefore future work, not evaluated capabilities.

5.2 Intent and Governed Workflow

Decapod does not currently require a universal `.decapod/intent.json` manifest or file-marker completion protocol. Instead, the workflow is distributed across explicit interfaces such as:

```
decapod session acquire
decapod todo add "task description"
decapod todo claim --id <task-id>
decapod todo handoff --id <task-id> --to <agent> --summary "..."
decapod infer orientation --intent "..." --task-id <task-id>
decapod workspace ensure --container
decapod validate
decapod todo done --id <task-id> --validated
```

Governed plans add a stronger pre-execution contract: execution is blocked when the plan is not approved, when intent or unknowns remain unresolved, or when no todo is selected. These mechanisms operationalize durable intent without implying that every natural-language requirement is machine-verifiable.

5.3 Constitutional Authority and Context-Poisoning Boundary

The baseline constitution is compiled into the Decapod binary and exposed as structured, queryable nodes. A binding project-local override is merged with tested project precedence. Context resolution then produces task-scoped, provenance-bearing projections of

this authority. Thus the constitution is not merely another repository instruction file: the intended chain distinguishes embedded baseline authority, explicit override, and selected context supplied before implementation inference. Exact asset, command, and test paths appear in the companion implementation matrix.

This is Decapod's proposed answer to repository-instruction poisoning: authority, scope, negative requirements, and proof expectations should be queryable and auditable rather than treating every available instruction as equally trustworthy. It does not prove that supplied context is relevant or benign, and project-controlled surfaces can still be hostile. The current code has no organization-level overlay; organization policy, three-level precedence, revocation, and conflict semantics remain roadmap work.

5.4 Operator Decoupling

Generated provider/workbench entrypoint shims delegate to one universal `AGENTS.md` contract. The intended human interface remains natural language: the operator states the outcome to the available agent, and the agent invokes Decapod's CLI or structured RPC at governance boundaries. This is interaction-contract portability, not a claim that vendor-specific commands disappear or that every workbench obeys the contract.

5.5 Candidate Agent-Governance Interoperability Profile

We offer the governance boundary as a candidate thin-waist profile: versioned semantics for capability discovery, constitutional authority, task custody and handoff, workspace status, selected trajectory, proof, and publication. Coding-agent harnesses could pre-bundle or discover a conforming local component so a user and project can move among Claude, Codex, Antigravity, Grok, and other workbenches without making one vendor conversation authoritative.

The implemented base is narrower. `decapod rpc` accepts a Decapod-specific structured request/response document over process standard input/output. It is not JSON-RPC 2.0, an MCP server, an A2A endpoint, or an HTTP service; no vendor-bundling agreement is claimed. MCP supplies a host–tool lifecycle and transport model [15]. A2A supplies agent discovery and task/artifact exchange [1]. HTTP and IP illustrate narrow layered interfaces that permit independent evolution [6, 16], but the analogy does not imply comparable maturity, governance, or adoption. Native adapters, negotiation, conformance vectors, identity, and version-skew behavior are prospective.

As of a July 13, 2026 API snapshot, Decapod had 222 GitHub stars, 21 forks, and 6,983 cumulative crates.io downloads [4, 8]. These counts establish public distribution and interest only, not correct use, interoperability, or empirical efficacy.

Decapod is used across 18 repositories in three owner namespaces (15 public and three private):

- **alexhraber:** `alexhraber`, `pi-cluster`, `builddeck`, `agentic-sdlc-intake-attack`, `metaplay-fake-interview-supply-chain-incident`, `flowhawk`, `tensors-to-consciousness`, `go-scaling`, `nixos`, private `cycle`, and private `mentora-automa`;
- **worldofgeese:** `den`, `tarot-api`, and `mythras-chargen`;
- **DecapodLabs:** `decapod`, private `gensor`, `accountable-agentic-execution`, and `pincher`.

The alexhraber/alexhraber repository is the author's public researcher and engineer profile. The worldofgeese maintainer discovered and adopted Decapod independently; the other namespaces are author- or organization-affiliated. Because repositories within each namespace share maintainers, this footprint is not evidence of 18 independent adopters, active protocol compliance, or a treatment effect. The companion inventory records the repository list and its methodological boundary.

The author reports a consistent lifecycle across new projects and engineering efforts: run `decapod init` once when establishing a repository, before the first agent interaction or other engineering operation. Thereafter the human interfaces naturally with one or more agents, while the generated `AGENTS.md` contract requires those agents to invoke Decapod throughout governed orientation, task custody, workspace use, validation, proof, and completion. Decapod is therefore a continuing control-plane dependency after one-time initialization, not a command the human repeatedly enters. This contract does not imply interception of every runtime action, and repository markers alone do not prove compliance or outcome improvement.

Decapod has also aided the development of this paper and its executable research artifact since their inception. The repository's governed specifications, tasks, workspaces, validation surfaces, and proof hooks make that use inspectable. This is reflexive dogfooding of the system under study, useful for implementation feedback but not independent validation or a controlled result.

5.6 Proof and Evidence Boundaries

Proof definitions contain commands, arguments, descriptions, and required status. A proof run records exit status, duration, pass/fail state, bounded output, run identifiers, actor, and related requirements in the proof event journal. Workunit and workspace-publish gates can require proof results and validation evidence before promotion.

The current implementation therefore supports auditable, machine-checkable evidence, hashes and attestations in selected governance surfaces, and deterministic replay or drift checks where specified. It does not support the stronger certificate described in the earlier draft: this paper does not claim per-command cryptographic diffs, a universal signed `proof.json`, GPG/SSH signing, arbitrary credential filtering, or complete interception of an agent's PTY. Those are possible future extensions and are excluded from the present empirical claims.

6 Cross-Agent Continuity and Heterogeneous Fleet Coordination

Decapod's second systems contribution is not a role-playing multi-agent team inside one orchestrator. It externalizes selected governance state into the project so independently launched processes can participate through a common contract. Figure 1 summarizes this local-first topology. We call the resulting bounded property *fleet coherence*.

6.1 Provider-Independent Project State

Generated workbench entrypoints route agents to one universal contract, while todos, context capsules, specifications, memory records, event journals, and proof artifacts live outside any one provider conversation. This makes a workbench replaceable in principle: a later process can query current project state rather than inherit only a chat summary.

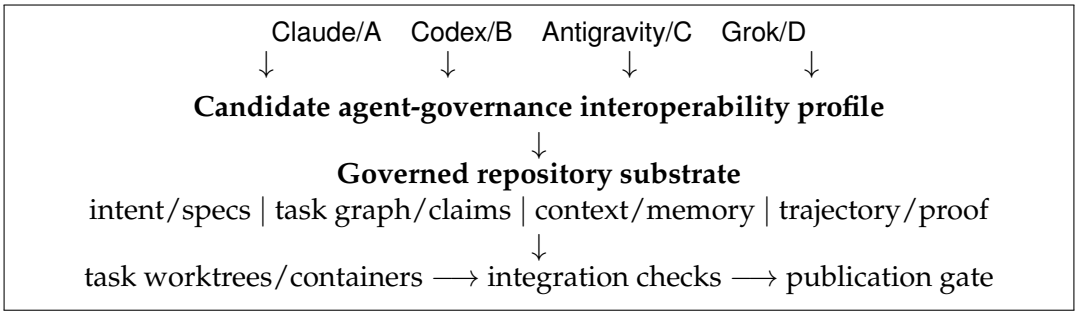

Figure 1: Ephemeral agent/workbench processes use one durable local project substrate. Arrows denote governed interfaces, not interception of every runtime action or global consensus.

The current implementation does not prove that every workbench follows the contract, and machine-local session credentials are not themselves project memory.

6.2 Project-Owned Memory and Context Continuity

Decapod exposes provenance-bearing knowledge, aptitude observations/preferences, and a typed local memory graph with lifecycle state. Context bundles and capsules bind canonical contents to task, scope, policy, and repository revision. These mechanisms permit later agents to retrieve selected decisions and constraints, but they do not ingest complete conversations automatically. More memory is not necessarily better: stale, contradictory, irrelevant, or harmful context is a first-class failure condition and is independently audited in the proposed studies.

6.3 Ownership, Dependencies, and Handoff

Exclusive claims use an atomic conditional update and conflict when another primary assignee is active. Trust-gated shared secondary ownership, category routing, tags, dependencies, heartbeat/presence, and agent expertise also exist. Because shared mode exists, this paper does not assert unconditional single ownership; the controlled fleet protocol selects exclusive claims.

The `todo handoff` path requires verified trust and explicit approval. It transfers assignment, appends a task-handoff event, stores a persistent knowledge record and observation, and attempts branch reconciliation. This is a real handoff protocol, but a bounded one: the summary may omit relevant reasoning, hidden chat state is unavailable, and reconciliation may fail. Study B measures exactly what the receiver obtained and what was preserved or lost.

6.4 Workspace Isolation and Integration Boundaries

Todo-scoped worktrees isolate direct mutations and protected-branch interlocks keep governed implementation out of the root checkout. Optional containers add a configured runtime boundary. Neither mechanism guarantees credential isolation, prevents an ungoverned external process from bypassing Decapod, or ensures independently valid changes are semantically compatible.

Publication therefore requires a separate state Q: task-local proof, integrated validation, provenance, evaluation/workunit state, and authorization may differ. Decapod's

publish path checks an unprotected worktree, provenance manifests, and configured workunit/evaluation gates. It does not manufacture post-merge correctness when no integration gate is configured.

6.5 Trajectory Reconstruction and Proof Portability

The flight recorder combines known todo, broker, proof, federation, and related events into timelines or transcripts. A later agent can use those records and state-bound proof references to avoid some reconstruction and repeated verification. Coverage is selective: model tokens, arbitrary shell output, every diff, and every tool event are not universally captured. Proof portability means evidence can be inspected and revalidated against its state; it does not authorize reuse after relevant state changes.

6.6 Natural-Language Operator Interface

The human-facing contribution is operator decoupling. The human continues to express the outcome and constraints in natural language to whichever workbench is available; the agent translates governance boundaries into stable machine calls. A candidate agent-governance profile would let harnesses pre-bundle or discover this component without requiring people to memorize a new vendor dialect. The proposed Study D measures whether this reduces ad hoc briefing and workbench-dialect translation rather than assuming it does.

6.7 Similar-but-Distinct Concurrent Work

The central fleet question is not merely whether two agents can edit separate worktrees. It is whether independently launched agents can solve similar but distinct problems in one codebase while preserving common authority, distinct custody, dependency awareness, and integrated proof. Such tasks can share architecture, modules, tests, or publication boundaries without being duplicates. Current Decapod claims identify declared task ownership but do not infer semantic overlap automatically. Study C therefore audits pairwise task relationships and measures duplicate work, false serialization, missed overlap, deferred merge/integration failure, and post-integration proof.

6.8 Local Coordination Boundary

The inspected implementation supports local repository coordination. Its subsystem named *federation* is a local typed memory graph, not a cloud consensus service, and the public cloud backend is explicitly unavailable. Cross-machine leases, remote ownership transfer, identity federation, and synchronization are future systems questions.

7 Evaluation Protocol and Research Program

The artifact is in harness validation, not empirical data collection. Checked-in records are explicitly `synthetic_fixture` data and cannot support a treatment-effect claim. The study's primary test is one-shot walk-away delegation: whether one sufficiently complex natural outcome-oriented request can produce an independently acceptable result without ongoing human coaching because Decapod resolves authoritative project context before implementation inference.

7.1 Primary Comparison and Mechanism

The primary comparison uses the same natural delegation prompt, base model and version, decoding and runtime configuration, repository and starting commit, tools, timeout, hidden acceptance rubric, evaluator, and clean-context reset. The conventional cell (CN) receives ordinary repository and tool access without Decapod-mediated constitution resolution, governed orientation, task claim, context capsule, workspace custody, or proof corridor. The Decapod cell (DN) receives the applicable constitution, architecture and specification nodes, capabilities and constraints, standards, prior decisions, task ownership, custody, validation and proof expectations, and escalation rules through the configured corridor. No corrective human coaching is allowed after either run starts; genuine clarification follows only a frozen oracle.

The underlying model is unchanged. Decapod is hypothesized to make the agent system operationally more knowledgeable and contextually aligned by resolving relevant, authoritative project context before implementation inference. The primary interaction is therefore "type the outcome, start the agent, and walk away," not a comparison of prompt length alone.

7.2 Primary Outcome and Convergence Measures

The primary outcome is independent fidelity of the final repository state to the human or team's complete intent under zero or tightly bounded post-prompt intervention. A hidden evaluator scores functional correctness, explicit requirements, implicit project invariants, architectural alignment, project conventions, applicable industry standards, regression avoidance, scope discipline, validation completeness, evidence completeness, and readiness for human acceptance.

Convergence efficiency is measured using defined quantities: model inference requests; input and output tokens; tool calls; repository searches; files opened before the first implementation edit; time to the first relevant implementation action; failed implementation attempts; repeated or redundant calls; compiler, test, lint, and validation failure cycles; reversals or substantial rewrites; clarification requests; unresolved questions; elapsed time to acceptable completion; premature completion claims; and cost where available. The paper does not use "inference hits" as a measured quantity.

7.3 Pre-Inference Context Alignment

For each DN run, the capture path records selected constitutional nodes, project specifications, capability overlays, constraints and negative requirements, standards, prior decisions, unresolved questions, excluded context, total context size, context-resolution latency, and item-level provenance. Each item has an authority source, selection rationale, content hash, size, and selected/excluded status. An independent audit classifies each supplied item as necessary, useful, irrelevant, redundant, contradictory, or harmful. More context is not presumed to be better. CN receives an explicit empty Decapod-context manifest and does not receive the treatment capsule or hidden rubric.

7.4 Hypotheses and Secondary Ablation

H1 predicts higher independent intent fidelity for DN than CN. H2 predicts fewer model requests, failed cycles, redundant operations, and substantial rewrites. H3 predicts more application of relevant architecture, conventions, and standards without prompt

enumeration. H4 predicts more completion without clarification or corrective intervention. H5 predicts more complete validation, provenance, and review evidence. H6 requires all gains to be evaluated against context-resolution, workspace, validation, token, latency, and operational overhead.

Natural-versus-procedural prompting remains a secondary 2×2 ablation crossing conventional and Decapod-governed substrates with natural and semantically matched procedural prompts. The procedural variant additionally specifies anticipated files, steps, commands, tests, sequencing, and defensive instructions without changing the canonical intent or hidden rubric. A checklist-only condition is secondary only.

7.5 Tasks, Stages, and Falsification

Tasks require interacting reasoning: architecture discovery, multi-file or multi-layer changes, compatibility, project conventions, an applicable security, persistence, API, concurrency, or operational standard, validation, and scope discipline. Each package contains canonical intent, motivation, constraints, open questions, escalation policy, a hidden rubric, independent evaluator, paired prompts, clarification oracle, starting commit, available context, interruption/concurrency configuration, and exclusion rules. Runs start from clean repository state and isolated model context.

The program proceeds through harness validation, a labeled pilot, a tagged protocol freeze, confirmatory execution, and independent audit. Pilot records are excluded from confirmatory estimates. Every run identifies its data kind and records model/runtime, repository and Decapod commits, environment, permissions, prompts, events, interventions, oracle responses, context manifest, evaluator result, exclusions, and deviations. Synthetic and pilot records fail closed in confirmatory aggregation.

This is a prospective research program against an actively developed system. As Decapod adds or narrows features, the experiments, hypotheses, conditions, instrumentation, and sample plan may change before protocol freeze so that they continue to test the mechanisms actually implemented. Every pre-freeze revision must remain versioned and disclose what changed and why. Once a study protocol is frozen, its confirmatory hypotheses, conditions, outcomes, exclusions, and analysis rules do not change in response to observed data; a material implementation change requires a new protocol version or a declared deviation rather than silently moving the target.

The delegation thesis is weakened if DN does not improve fidelity, does not reduce failed or redundant cycles, increases cost without improving results, supplies irrelevant or harmful context, still requires regular human correction, is matched by CN at equal or lower effort, benefits from hidden rubric leakage, improves proof while task quality does not improve, or merely relocates supervision into setup. The analysis reports paired task-level contrasts, uncertainty, effect sizes, and task-clustered results; non-inferiority is prohibited until a margin and power analysis are frozen.

7.6 Study B: Cross-Workbench Handoff Continuity

Study B terminates Agent A at a frozen observable checkpoint and starts Agent B with a clean context. The credible conventional condition supplies the repository/branch, git history, public task description, and a standardized structured handoff document. The Decapod condition receives the same artifacts plus the implemented approved ownership transfer and frozen Decapod task, context, trajectory, unresolved-state, workspace, and proof references. The first level holds the model fixed where possible; heterogeneous models are an external-validity extension.

RQ7 asks whether governed handoff reduces reorientation and reconstruction; RQ10 asks whether a receiving workbench can continue without transcript replay or an ad hoc human briefing; RQ11 asks whether project-owned state preserves constraints and unresolved questions. H7 predicts fewer reorientation requests and duplicate reads, searches, commands, tests, and decisions. H9 predicts less constraint loss, and H11 predicts less repeated verification when proof remains state-valid. A conventional handoff that performs equally well falsifies the stronger claim.

7.7 Study C: Concurrent Heterogeneous Fleet

Study C asks whether multiple independently launched agents can concurrently solve similar but distinct problems in the same codebase while preserving shared authority, distinct custody, isolated mutation, dependency awareness, and integrated proof. It supplies a frozen graph of independent, dependent, duplicate, and similar-but-distinct tasks to at least two agent processes. Similar-but-distinct pairs share architecture, modules, tests, standards, or integration boundaries without expressing the same outcome. Both conditions use separate branches or worktrees. The conventional condition receives normal issue/task descriptions; the treatment additionally uses implemented Decapod claims, labels/categories, dependencies, workspaces, context, events, proof, and publication paths.

RQ8 asks whether custody prevents duplicate starts, RQ9 separates direct collision avoidance from deferred integration failure, and RQ12 evaluates net coordination cost. An independent pre-run audit labels pairwise task relationships and expected overlap. Measures include duplicate starts, false serialization, missed semantic overlap, claim conflicts, redundant discovery, overlapping paths, dependency violations, direct collisions, textual merge conflicts, semantic integration failures, abandoned work, repair effort, duplicated model calls/tokens, wall-clock time, integrated fidelity, proof completeness, and publication success. H8 predicts fewer duplicate starts. H10 predicts fewer direct collisions but explicitly permits no reduction in semantic merge conflicts. Serializing every similar task does not support the concurrency thesis.

7.8 Study D: Tool-Switch Continuity

Study D applies Study B's checkpoint and measures to frozen workbench pairs. It records source/destination workbench, model, session, exact human briefing, and the Decapod information delivered to the receiver. Same-model and heterogeneous-model transfers are stratified rather than silently pooled.

7.9 Study E: Longitudinal Experience Report

Study E is observational. A checked-in historical snapshot pinned to Decapod `e7df80d` derives 2,127 commits, 326 tags, 612 merge commits, two explicit revert-subject commits, the February–July 2026 development interval, authorship counts, and changed top-level path frequencies from git. It remains an explicitly older observational dataset than the `207e9a3` implementation audit. These facts establish sustained repository activity, not a treatment effect, correctness, low review burden, or proof that named workbenches produced particular commits. Author-reported workbench use must be labeled separately. The experience report motivates controlled studies but cannot enter their estimates.

7.10 Study-Specific Provenance and Authorization

Run schema version 4 declares one of `walk_away_alignment`, `handoff_continuity`, `concurrent_fleet`, or `tool_switch_continuity`; Study E uses a separate observational schema. Capture records source/destination model, workbench and session; owner transfer; handoff payload; context before/after; preserved/lost questions, dependencies, and proof; duplicate operations; workspace and claim state; integration and publication outcomes; human briefing; cost; and deviations. Aggregation requires one study, one run kind, and one evidence class. Synthetic, pilot, empirical, and observational records fail closed across those boundaries.

No fleet pilot or empirical run is authorized by this paper revision. Execution requires separately approved task graphs, checkpoints, model/runtime access, credentials, and budget.

8 Discussion

The proposed evaluation is designed to test whether a human can delegate one complex outcome and walk away while the agent resolves authoritative context before implementation inference. This is a system-level capability hypothesis, not a claim that Decapod changes the underlying model's parameters or intrinsic intelligence. The primary estimand is independent final-state fidelity and convergence under identical natural prompts; prompt style is secondary.

8.1 Systems Framing vs. Prompt Engineering

Historically, the primary method for improving coding agent performance has been prompt engineering—tuning system instructions, adding few-shot examples, or structuring chat outputs. While these methods improve the probability of a model generating correct syntax, they do not change the fundamental execution model. A model executing directly on a production workspace remains unbounded and unverified.

Enforcing explicit custody and proof shifts the focus from inputs (prompts) to execution boundaries and evidence. Decapod can require a claimed task to enter an isolated workspace and can block promotion paths when validation or proof requirements are missing. This is narrower than complete sandboxing: the current kernel does not guarantee that every incorrect change is detected, that every credential is inaccessible, or that every agent action is intercepted. The evaluation must therefore measure both safety benefits and governance overhead.

The findings of Gloaguen et al. [9] sharpen this argument. Their results show that repository context can be respected while still failing to improve task success and increasing inference cost. Instruction following is not equivalent to good governance: an agent may faithfully follow a poisoned, redundant, or over-broad context file. Decapod's constitution and context-capsule structure is motivated by this distinction. It aims to make authority, scope, and proof-relevant context explicit and queryable before inference, while preserving minimality as a design constraint. The independent context audit therefore classifies supplied items as necessary, useful, irrelevant, redundant, contradictory, or harmful. This is a testable architectural hypothesis, not a result established by the present pre-results artifact.

Decapod's current authority chain is concrete: a baseline constitution is embedded in the binary, a project override is explicitly binding, and task context is a selected

projection with provenance. The author coined *Intent-Driven Design* as the name of this software-engineering approach in public LinkedIn posts beginning in August 2025 and, by September 2025, was using a constitution to scaffold project boilerplate and derived agent guidance. This design lineage preceded and arose independently of the February 12, 2026 posting of the Gloaguen et al. preprint. The historical attribution does not establish mitigation efficacy: whether structured constitutional authority selects better context or improves outcomes remains untested.

8.2 The Control Plane for Machine Work

In traditional software engineering, human developers follow processes: they checkout branches, write tests, request pull request reviews, and run CI/CD pipelines. However, these processes were designed for human timelines. An autonomous agent can run dozens of commands per minute, coordinate across multiple repositories, and execute complex workflows concurrently.

Decapod acts as a local control plane for machine work. It brings intent resolution, task ownership, workspace isolation, validation, proof events, and publication gates into a repository-native workflow. It does not itself coordinate every command issued by an arbitrary model runtime, so the boundary between Decapod evidence and the agent host remains part of the threat model.

For one agent, the central mechanism is pre-inference contextual alignment. For a fleet, the mechanism is reuse: a later process can query durable authority, ownership, selected context, trajectory, and proof instead of treating the previous conversation as the project. This does not require agents to share one orchestrator process. It also creates new risks: stale project memory can spread, claims can become abandoned, and locally valid proof can be reused after its state binding has changed.

8.3 Cost of Fragmentation

The economic hypothesis separates nine costs: *context rehydration* (repeated project discovery), *duplicated inference* (multiple agents solving the same work), *collision* (repair after overlapping mutations or assumptions), *handoff* (reconstructing missing state), *verification* (rerunning unavailable or untrusted checks), *integration* (repairing incompatible results), *dialect* (human translation into workbench procedures), *context bloat* (irrelevant governance input), and *abandonment* (work that cannot be resumed economically).

Decapod seeks to lower the cost floor by making useful intent, context, ownership, progress, and evidence reusable across inference requests, sessions, models, workbenches, and teams. This is not yet a result. Every study must count setup, context resolution, state maintenance, workspace, proof, integration, token, latency, and human configuration costs. A simple issue tracker, structured handoff, and worktrees may capture most of the benefit.

A standard agent-callable component could reduce dialect waste further: harnesses would bundle or discover the same versioned governance semantics while humans continue delegating in natural language. That is a portability hypothesis, not current deployment. A Decapod-specific CLI may itself become lock-in unless the profile, conformance vectors, adapters, and failure behavior can be implemented independently.

8.4 Trust, Auditing, and the Limitations of Proof

A central hypothesis of this study is that structured proof files and trajectories will reduce human review time. In a transcript-only workflow, human reviews are cognitively demanding: the developer must read long chat threads, try to understand why the agent chose a specific path, and then manually check if the code works. A Decapod run, however, separates the conversational noise from the systems evidence.

Programmatic proof has strict limits. Decapod evidence can show that configured commands ran and what they returned, but it cannot verify soft requirements such as code elegance, architectural maintainability, or completeness of an unstated user goal. The goal is therefore to make objective checks and governance state easier to inspect, not to replace human review or convert a validation pass into a security guarantee. The thesis is weakened if natural delegation performs no better, if procedural prompts remain necessary, if governance relocates human effort into setup, or if overhead outweighs reliability and autonomy benefits.

Fleet coherence has equally strict limits. Exclusive claims can reduce duplicate starts while doing nothing for task decomposition quality. Worktrees can eliminate direct file trampling while deferring semantic conflicts to integration. Handoff records can improve attribution while omitting the decision the receiver needs. Proof can become more portable while actual task quality stays unchanged. Null and negative results on these dimensions would narrow the systems thesis even if the mechanisms operate as implemented.

9 Related Work

Our research intersects with coding-agent workbenches, multi-agent orchestration, repository context and memory, task/workspace coordination, validation, and software-supply-chain evidence.

9.1 LLM Agents in Software Engineering

Since scalable Transformers [23] and few-shot models such as GPT-3 [3], code generation has moved from autocompletion to read-write-execute agents. OpenHands provides a model-flexible software-agent platform with command-line, browsing, sandbox, benchmark, and multi-agent coordination surfaces [24]. These systems overlap substantially with Decapod on realistic execution and isolation.

The author coined *Intent-Driven Design* as the name of Decapod's software-engineering approach and described it publicly in LinkedIn posts beginning in August 2025. By September 2025, the core implementation pattern used a project constitution to represent durable human authority and scaffold project boilerplate and derived guidance before implementation. The prompt was treated as an initial expression of intent, while the constitution and generated project artifacts progressively operationalized that intent for agents. This terminology and constitution-first lineage originated independently of the February 2026 repository-instruction study discussed below. The historical attribution identifies the provenance of Decapod's design; it is not evidence that the mechanism improves task outcomes, which remains an empirical question for this research program.

Our narrower focus is the project control plane external to one agent loop. The research question is whether repository-native authority, ownership, handoff, context, and proof remain usable when independently launched processes do not share one conversation or

orchestrator lifecycle. We do not claim that all contemporary agents are transcript-only or lack memory, sandboxing, or coordination.

9.2 Multi-Agent Orchestration and Role-Based Teams

MetaGPT encodes software-development roles and standard operating procedures into a multi-agent framework [10]. ChatDev organizes specialized agents into a chat chain across design, coding, testing, and documentation [17]. AutoGen provides conversable-agent abstractions for composing multi-agent applications and human/tool interaction [25]. OpenHands also supports coordination among agents inside its platform [24].

These frameworks show that role specialization, conversation topology, model flexibility, memory, and sandboxed execution are not unique to Decapod. The proposed distinction is architectural rather than categorical: Decapod externalizes governance into durable project state and a CLI/RPC contract so separately launched workbenches can participate without joining one chat chain or long-running orchestrator process. Whether that externalization improves handoff or concurrency is precisely what Studies B–D must test. A structured baseline may match it.

9.3 Repository Context, Context Poisoning, and Memory

Gloaguen et al.'s study of repository-level agent instructions is the closest contemporaneous empirical comparison [9]. Across multiple agents and LLMs, that study reports no improvement in task success, more than 20% higher inference cost, broader exploration such as additional testing and file traversal, and substantial instruction-following. Its result is important for this paper's thesis: repository context is not inert documentation. It changes the agent's execution trajectory and resource consumption, so unnecessary or contradictory requirements can act as context poisoning even when the agent appears to follow them.

Our position is not that a longer constitution is automatically better. Decapod responds to the same risk by treating context as governed control-plane input: the constitution is structured into queryable nodes, `context.resolve` and context capsules scope what is supplied before inference, and plans preserve unknowns, human questions, stop conditions, and proof expectations. This design makes context selection and authority inspectable rather than placing an unbounded instruction file in the prompt. The ETH Zürich findings therefore support the need to evaluate context quality, scope, cost, and behavioral side effects; they do not by themselves establish that Decapod's structure improves task success. Our evaluation should test whether structured, scoped context reduces harmful drift and review burden while avoiding the overhead identified by Gloaguen et al.

9.4 Program Verification and Sandboxing

Our custody and proof primitives build on program verification and systems sandboxing. Traditional verification systems target mathematical correctness; our proof primitive is pragmatic, configured evidence from compilers, test suites, static analysis, workunits, and provenance.

Sandboxing platforms isolate untrusted execution, while Git worktrees support multiple branches in separate working directories [7]. Decapod composes task claims, worktree checks, and optional containers. Worktrees isolate mutations but do not isolate credentials or prove merge compatibility; this paper does not call them a shell sandbox.

9.5 Workflow Durability, Task Coordination, and Provenance

Issue trackers and workflow engines preserve task and execution state independently of one worker. Software supply-chain systems such as in-toto go further by binding authorized steps, actors, materials, and products into verifiable metadata [21]. Decapod's proof and provenance surfaces occupy a narrower development-time role: they record configured governance/validation evidence for agent work. They are not a substitute for in-toto's cryptographic supply-chain guarantees, nor do they sign every action.

The closest conventional controls for the fleet studies are therefore not ungoverned shared directories. They are issue/task descriptions, structured handoff documents, separate worktrees, git history, CI, and provenance tooling. The experiments compare Decapod against these credible combinations.

9.6 Agent and Protocol Interoperability

MCP specifies a host–client–server architecture, JSON-RPC data layer, capability negotiation, tools/resources/prompts, and standard local or HTTP transports [15]. A2A specifies discovery and task/artifact exchange between independent, opaque agent systems [1]. They address adjacent interoperability layers: MCP connects an agent host to callable context/tools, while A2A connects collaborating agents. HTTP and IP demonstrate the broader value of narrow interfaces that permit intermediaries, layered transports, and independent implementations [6, 16].

Decapod does not currently implement these protocols. Its present interface is a local CLI and Decapod-specific structured process RPC. The proposed contribution is a candidate governance profile over constitutional authority, task custody, trajectory, proof, and publication, with MCP and A2A adapters rather than a competing all-purpose agent protocol. The claim must remain conditional until an independent implementation, conformance suite, identity model, and actual harness integrations exist.

9.7 Distributed Systems and Ledger Audits

The framing of agentic coding as a distributed systems execution problem is inspired by classic distributed consensus and audit architectures. Lamport's happenings-before relation [12] laid the groundwork for ordering events across independent computing nodes. MapReduce [5] and distributed databases demonstrated how to partition state and manage concurrency across large scale clusters.

Similarly, event sourcing and cryptographically bound artifacts provide conceptual inspiration for Trajectory and Proof. Decapod's current contribution is narrower: it journals selected governance/proof events and exposes validation, provenance, and attestation surfaces. It does not implement distributed consensus, log every tool call, or sign one universal execution ledger.

10 Threats to Validity and Future Work

10.1 Controlled-Study Validity

The walk-away study may be sensitive to one model/runtime, provider nondeterminism, task selection, prompt-pair equivalence, benchmark contamination, evaluator validity, hidden-rubric leakage, and repeated-run dependence. Instrumentation may alter behavior, while task authors may encode Decapod-compatible conventions that favor the

treatment. Clean context and repository resets, frozen oracles, blinded independent evaluation, task-level clustering, protocol tags, and complete failure accounting reduce but do not remove these threats.

Fleet studies add checkpoint selection, workbench/version drift, model heterogeneity, handoff-document quality, shared-state contamination, and concurrency scheduling. Duplicate operations are difficult to classify semantically: rereading a file may be prudent verification rather than waste. A Decapod condition may also receive more information than the baseline. The protocols therefore record exact supplied context, use credible structured handoffs and worktrees, stratify model changes, and audit relevance. Generalization to production teams, proprietary repositories, distributed organizations, and long-lived maintenance remains unestablished. However, the author has continuously used Decapod since early February 2026 to build Decapod itself and reports a repeated repository lifecycle: one `decapod init` before the first agent interaction, followed by continuing agent use under the generated `AGENTS.md` contract. The 13-repository marker inventory is consistent with this practice. Together these are ecological evidence of sustained and repeated use by one maintainer network, not evidence of generalization to other teams or settings.

The longitudinal case study is especially limited. Git history measures repository activity, not agent autonomy, review effort, defect escape, or causal benefit. Author testimony about workbench use is subject to recall and selection bias and remains separate from repository-derived facts.

10.2 Implementation and Security Boundaries

The implementation audit is pinned to one Decapod commit and may become stale. The kernel does not alter model weights, provide a complete OS sandbox, guarantee credential isolation, capture every token/shell byte/tool event, guarantee conflict-free merges, prove semantics beyond configured gates, or make all shared context useful. It does not currently provide organization-level constitution overlays, native MCP or A2A adapters, an HTTP governance service, external conformance certification, or vendor pre-bundling. Governance may relocate human work into configuration, increase latency/tokens, or perform no better than simpler controls.

10.3 Interoperable Agent-Governance Standardization

A future profile should define a narrow, versioned contract for capability discovery, constitutional authority, task custody and handoff, workspace state, selected trajectory, proof, and publication. MCP adapters could expose those semantics to agent hosts; A2A adapters could exchange governed task and evidence references among independent agents; an optional HTTP transport could support remote clients. Conformance vectors, capability negotiation, identity and authorization, idempotency, version-skew behavior, and fail-closed degraded operation are prerequisites.

The goal is independent implementation and harness pre-bundling, not a new human command dialect or a Decapod-only protocol moat. HTTP and IP are design analogies, not maturity or standards-status claims. Standardization would fail its portability goal if integrations merely translate every vendor command, if one implementation remains authoritative, or if a thin contract cannot preserve the security and provenance boundaries of local execution.

10.4 Federated Agentic Software Execution

The evaluated system is local-first. A future optional cloud coordination plane could expose cross-machine task discovery, distributed leases, remote handoff, evidence publication, and cross-team visibility while retaining local repository authority. That direction introduces identity and actor attribution, lease expiry, synchronization conflict, institution-specific policy, privacy and data minimization, disconnected operation, trust, revocation, and evidence-portability questions.

Such federation should not silently move source-of-truth authority into one vendor's agent platform. Institutions may retain local custody and synchronize only bounded coordination/evidence records. The inspected public crate does not implement this system: its cloud backend is unavailable and its local subsystem named federation is a typed memory graph, not distributed consensus. Future claims require an implemented protocol, threat model, consistency semantics, and separate evaluation.

11 Conclusion

As LLM-based coding agents transition from suggestion to delegated execution, we must treat agentic software engineering as a distributed systems execution problem rather than a simple text-generation task. The traditional transcript-only workflow—relying solely on chat transcripts and code diffs—fails to provide the safety, recovery, and verification guarantees required for enterprise codebases.

This paper formalized four primitives for accountable agentic execution:

- **Intent**: Human/team desired outcomes, motivations, constraints, priorities, unknowns, stop conditions, and completion standards, progressively resolved through governed artifacts.
- **Custody**: Bounded execution sandboxes and permission limits.
- **Trajectory**: Event-backed governance records rendered for reconstruction and review.
- **Proof**: Machine-checkable validation, proof-run, workunit, and provenance evidence.

We presented Decapod, a daemonless, local-first governance kernel, as a reference implementation pinned to commit `207e9a3`. The implemented binary enables pre-inference context alignment, allowing a human to delegate complex desired outcomes and walk away with less procedural cognitive load. The primary research program tests whether that mechanism measurably preserves intent fidelity and reduces correction relative to credible controls. The expanded program tests fleet coherence: whether heterogeneous, independently launched processes can preserve common authority, own separate work, hand tasks across workbenches, solve similar but distinct problems concurrently, and share state-bound evidence without depending on one conversation. The constitution—embedded baseline plus project override in the current implementation—is the authority root from which task context is selected.

We additionally offer these governance semantics as a candidate agent-callable interoperability profile that coding-agent harnesses could pre-bundle or discover. The goal is for humans to delegate naturally while agents invoke a portable machine contract, not

for users to learn another vendor dialect. Native MCP/A2A adapters, HTTP transport, organization policy, conformance, and actual vendor bundling remain future work.

The comparative-effect claims are prospective. Controlled records remain synthetic; the longitudinal git snapshot is observational and non-causal. Decapod demonstrates real local claims, handoff, context, memory, workspace, event, proof, and publication mechanisms, but not complete security, lossless conversation transfer, conflict-free integration, global consensus, or production cloud federation. The research program is designed to reject the thesis if credible structured handoffs and ordinary worktrees perform equally well, shared context harms agents, or coordination overhead outweighs measured benefit. The author has not observed those failure conditions while using Decapod to build Decapod since early February 2026 or across subsequent repositories initialized once and then continuously governed through their agent contracts; that experience is observational and author-reported rather than a controlled comparative result.

References

[1] Agent2Agent Protocol Contributors. Agent2agent protocol specification. A2A Protocol documentation, 2026. Version 1.0; accessed July 2026.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 1877–1901, 2020.

[4] crates.io. Crate metadata for decapod. crates.io API, 2026. Snapshot queried July 13, 2026.

[5] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. First presented at OSDI 2004. ACM journal publication year is 2008.

[6] R. T. Fielding, M. Nottingham, and J. Reschke. HTTP semantics. RFC 9110, Internet Engineering Task Force, 2022.

[7] Git Project. git-worktree documentation. `https://git-scm.com/docs/git-worktree.html`, 2026. Accessed July 2026.

[8] GitHub. Repository metadata for DecapodLabs/decapod. GitHub API, 2026. Snapshot queried July 13, 2026.

[9] T. Gloaguen, N. Mündler, M. N. Müller, V. Raychev, and M. Vechev. Evaluating AGENTS.md: Are repository-level context files helpful for coding agents? In *MemAgents Workshop at ICLR*, 2026. Preprint posted February 12, 2026; oral presentation and runner-up best paper.

[10] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Zhang, C. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 25, pages 1097–1105, 2012.

[12] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[13] M. Minsky and S. A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[15] Model Context Protocol Contributors. Architecture overview and server specification. Model Context Protocol documentation, 2025. Accessed July 2026.

[16] J. Postel. Internet protocol. RFC 791, Internet Engineering Task Force, 1981.

[17] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024.

[18] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[20] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[21] S. Torres-Arias, H. Afzali, T. K. Kuppusamy, R. Curtmola, and J. Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium*, pages 1393–1410, 2019.

[22] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(1):230–265, 1936.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[24] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig. OpenHands: An open platform for AI software developers as generalist agents. In *International Conference on Learning Representations*, 2025.

[25] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.