CONTRACT-GOVERNED MULTI-AGENT GRAPH ORCHESTRATION FOR LONG-HORIZON AUTONOMOUS RESEARCH PIPELINES

Anonymous authors
Paper under review

ABSTRACT

Long-horizon research automation needs stronger semantics than prompt chaining or schema-only artifact passing. We study a restricted contract-governed formulation of Quarks in which a finite active workflow branch is modeled as a typed directed acyclic graph, each node carries explicit assumptions and guarantees over artifact spaces, cross-node transport is mediated by sound adapters and admissible aggregators, and execution state is split into a global append-only provenance ledger and branch-local working memory. Within that regime we prove three results. First, typed local admissibility composes over the branch when the contract premises hold. Second, branch isolation and provenance monotonicity follow from the append-only, non-merge memory discipline. Third, local validator passes do not imply unrestricted graph-level correctness: we construct a two-node counterexample and then prove a narrow positive corollary for a manuscript-defined typed property class under explicit coverage, adapter, and memory assumptions. The proof package is complemented by seeded finite theorem audits and implementation-alignment checks against the Quarks architecture context. The positive audit families close completely in the declared regime, with 40/40 contract-composition cases, 45/45 memory cases, and 15/15 restricted-assurance cases passing; each theorem also retains a nontrivial archive of negative witnesses outside the proved setting. The result is a bounded but useful semantics for research orchestration: it supports typed admissibility, provenance discipline, and limited assurance claims without upgrading branch merges, semantic quality, or unrestricted agent competence into theorem-level guarantees.

1 INTRODUCTION

Multi-agent systems have become a standard organizing pattern for language-model agents, software pipelines, and research assistants because they let practitioners decompose long tasks into role-specialized components with explicit intermediate artifacts (Chowa et al., 2026; Li et al., 2024; Guo et al., 2024; Wang et al., 2023b). Systems such as AutoGen, CAMEL, AgentVerse, MetaGPT, and ChatDev show that coordination protocols, role structure, and message specialization materially change long-horizon behavior (Wu et al., 2023; Li et al., 2023; Chen et al., 2023; Hong et al., 2023; Qian et al., 2023). The same pattern extends to scientific-writing and automated-research loops, where PaperOrchestra and The AI Scientist treat manuscript assembly, experimentation, and review as staged coordination problems over persistent artifacts (Song et al., 2026; Lu et al., 2024). Search, reflection, and repair methods further show that branching and backtracking help language agents recover from early mistakes (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024; Zhou et al., 2023; Yao et al., 2022; Shinn et al., 2023; Madaan et al., 2023).

What those systems usually do not provide is a precise account of which local checks compose, which memory mechanisms preserve provenance, and which claims must remain at the level of runtime monitoring or empirical evaluation rather than theorem-level guarantee. That gap matters most in long-horizon research pipelines. A research workflow spans acquisition, synthesis, theorem development, validation, writing, revision, and export. Each stage consumes and emits structured artifacts, may branch under uncertainty, and often uses validators or monitors to gate progression. When the pipeline runs for hours or days, a merely operational description of the roles is not enough. We need a mathematical statement of what the orchestration layer preserves, what it cannot preserve, and how far typed interfaces and validators can carry assurance.

Quarks is an appropriate setting for that question because its architecture already treats phases, contracts, semantic expectations, and persistent cross-phase state as first-class objects. The uploaded implementation context exposes typed output schemas, explicit phase manifests, and validator-bearing payloads, while the surrounding documentation emphasizes branching, preserved context, observability, and report generation. Those features are strong evidence that

Quarks is not just another prompt chain. They are not, however, a proof that arbitrary long-horizon executions are correct. The manuscript therefore asks a narrower and more defensible question: under what formally stated regime does contract-governed orchestration support a local-to-global typed admissibility result, a branch-local provenance lemma, and a principled assurance boundary?

The answer in this paper is deliberately restricted. We work over a finite active workflow branch represented as a directed acyclic graph. Each node is associated with typed input and output spaces, assumptions, guarantees, and a local transition. Edge transport either matches directly or passes through a sound typed adapter; multiple predecessors are combined only through an admissible aggregator. Execution state is split into a global append-only ledger and branch-local working memory. Branch merges, hidden shared mutable state, and unrestricted semantic-quality predicates are excluded. Within that regime we can state complete proofs and connect them to finite symbolic audits without confusing audit evidence with theorem truth.

The manuscript makes four concrete contributions.

- We define a contract-bounded workflow semantics in which orchestration policies, adapters, aggregators, validators, and memory updates are explicit mathematical objects rather than informal implementation conventions.
- We prove a composition theorem showing that typed local admissibility propagates over a finite active branch when every edge is direct-compatible or mediated by a sound adapter and each successor receives an admissible aggregated input.
- We prove a branch-local memory lemma and an assurance-boundary theorem that jointly explain why append-only provenance discipline helps but local validator passes alone cannot certify unrestricted graph-level correctness.
- We connect the theorem package to proof-oriented validation evidence: seeded finite audits close in the declared regime, while explicit counterexamples mark exactly where the guarantees stop.

The broader relevance of this result is cross-domain. Long-horizon language-agent systems now appear in software engineering, chemistry, robotic planning, and scientific discovery (Yang et al., 2024; Bran et al., 2024; Boiko et al., 2023; Pelletier et al., 2025). In each of these settings, designers face the same basic tension: rich tool use and flexible adaptation improve capability, but open-endedness makes unrestricted correctness hard to prove. By importing the discipline of contracts, interface theories, and runtime assurance into research orchestration, we do not solve open-ended autonomy in full. We do show that a meaningful middle ground exists between unstructured prompting and unrealistic total-correctness claims.

The rest of the paper moves quickly from motivation to formal content. Section 2 synthesizes the literature boundary that motivates our restricted semantics. Section 3 defines the objects, assumptions, objective, and standing regime. Section 4 proves the composition theorem, and Section 5 proves the memory lemma, the assurance counterexample, and the restricted positive corollary. Section 6 and Section 7 explain how the theorem package was stress-tested with finite audits and how the negative witnesses discipline the claims. Section 8 states the remaining limitations explicitly instead of hiding them behind architectural rhetoric.

2 Related Work

Two bodies of literature are necessary for this paper, and neither is sufficient on its own. The first body studies multi-agent orchestration, long-horizon reasoning control, and research automation. The second body studies contracts, interface theories, formal verification, and runtime monitoring. Our contribution sits at their boundary. That boundary matters because the manuscript is neither a generic agent survey nor a direct transfer of classical contract algebra into a new application. The literature review must therefore do more than list neighboring systems. It has to separate operational precedents from formal ancestors, identify which parts of the vocabulary we inherit directly, and explain where new objects are required by the Quarks setting. The subsection structure below follows that logic, moving from orchestration practice to reasoning-control mechanisms and then to the assurance literature that makes a bounded theorem paper possible.

On the orchestration side, the most relevant lesson is not simply that many agents can outperform one agent on some benchmarks. It is that long-horizon tasks become legible only after designers introduce explicit roles, intermediate artifacts, and routing structure. Conversation-based systems such as AutoGen and CAMEL demonstrate that multi-agent interaction can expose decomposition and critique patterns, while project-oriented frameworks such as MetaGPT and ChatDev show that staged roles and typed deliverables can stabilize longer workflows (Wu et al., 2023; Li et al.,

2023; Hong et al., 2023; Qian et al., 2023). AgentVerse and later surveys make the same point at a broader level: workflow structure, infrastructure, and tool mediation are architectural choices rather than superficial prompt tweaks (Chen et al., 2023; Chowa et al., 2026; Li et al., 2024; Guo et al., 2024; Wang et al., 2023b). Those papers justify taking orchestration as a serious systems object, but they do not by themselves provide the typed composition or assurance language needed for theorem statements.

On the formal-methods side, classic contract and interface theories provide exactly the disciplines that orchestration papers often leave implicit: assumptions must be stated, guarantees must be scoped, and composition must be shown under explicit premises rather than inferred from successful examples. Design by contract, modal interfaces, and assume-guarantee frameworks offer reusable semantic patterns for local compatibility and compositional reasoning (Jazequel & Meyer, 1997; Raclet et al., 2011; Benveniste et al., 2018; Incer et al., 2025). We invoke those sources at the level of vocabulary, compatibility discipline, and compositional proof style rather than as a claim that one inherited theorem already covers the Quarks setting. Yet those theories were not originally written for LLM-centered research pipelines with branch-local working memory, persistent provenance, and validator-mediated artifact flow. If imported naively, they risk overclaiming by treating open-ended semantic quality as if it were just another local contract predicate. Our review therefore treats the contract literature as a source of proof discipline and notation, not as a drop-in semantics for autonomous research.

The missing bridge is a literature that simultaneously respects orchestration practice and formal assurance boundaries. Runtime verification, autonomous-systems assurance, and recent work on robotic skill composition show why that bridge must be narrow (Leucker & Schallhart, 2008; Wongpiromsarn et al., 2023; Torben et al., 2023; Pelletier et al., 2025). Monitoring can certify the properties it observes, but it cannot magically promote partial local checks into unrestricted global correctness. Likewise, recent research-paper automation systems such as PaperOrchestra demonstrate the operational plausibility of specialized end-to-end research workflows, yet their main evidence concerns task completion and writing quality rather than formal compositional guarantees (Song et al., 2026). This is the exact opening exploited by the present manuscript: use the orchestration literature to motivate the architecture, use the contract literature to discipline the proof language, and then prove only the narrow branch-local claims that the combined setting can honestly support.

2.1 Multi-Agent Orchestration and Research Pipelines

Modern multi-agent LLM systems demonstrate that decomposition matters. AutoGen frames coordination as multi-agent conversation, CAMEL studies role-playing societies, AgentVerse examines collaborative behavior, and MetaGPT and ChatDev translate software-development roles into explicit staged workflows (Wu et al., 2023; Li et al., 2023; Chen et al., 2023; Hong et al., 2023; Qian et al., 2023). Surveys now treat role specialization, workflow structure, and infrastructure as first-class design dimensions for agentic systems rather than peripheral engineering choices (Chowa et al., 2026; Li et al., 2024; Guo et al., 2024; Wang et al., 2023b).

However, these systems usually stop at protocol description and benchmark behavior. They show that role splitting, tool access, and message structure improve task completion, but they do not define when a phase interface is merely a schema and when it becomes a proof-relevant contract. They also rarely state a precise memory law that distinguishes branch-local state from global provenance. Even systems that are closest to our application domain, such as PaperOrchestra and The AI Scientist, rightly focus on end-to-end research or writing loops rather than on a local-to-global theorem for typed orchestration (Song et al., 2026; Lu et al., 2024). Quarks differs at exactly this point: the architectural context motivates a semantic treatment of phases and validators, which makes a formal paper possible.

2.2 Reasoning Control, Search, and Self-Correction

Search and repair methods provide the operational intuition for rerouting and recovery. Chain-of-thought prompting exposes intermediate reasoning states; Tree of Thoughts, Graph of Thoughts, and language-agent tree search turn those states into explicit search objects with branching and evaluation (Wei et al., 2022; Yao et al., 2023; Besta et al., 2024; Zhou et al., 2023). ReAct couples reasoning with action, while Reflexion and Self-Refine use critique and iterative repair to improve long-horizon behavior (Yao et al., 2022; Shinn et al., 2023; Madaan et al., 2023). Voyager and Generative Agents show that memory and replay are central when the horizon becomes long or the environment evolves (Wang et al., 2023a; Park et al., 2023).

This literature motivates why branch-aware orchestration needs more than a fixed linear schedule. But it does not answer our central mathematical question. Search papers usually optimize reasoning quality or task reward; they do not characterize which branch edits preserve typed admissibility or provenance monotonicity. Their objects are

heuristically powerful but semantically underconstrained. We therefore use them as comparators for operational expressiveness, not as inherited proofs.

2.3 Contracts, Interface Theories, and Design by Contract

The semantics we need comes instead from contract-based design. Design by contract gave software engineering a vocabulary of preconditions, postconditions, and invariants through the Ariane case-study lineage (Jazequel & Meyer, 1997). The overview monograph *Contracts for System Design* is the main broad source here for assume-guarantee composition across systems and components, and we use it together with adjacent contract-based design work as the representative background for composition and refinement (Benveniste et al., 2018; Kaiser et al., 2015; Meng et al., 2021). Modal interface theory supplies the compatibility language used to reason about adjacent components (Raclet et al., 2011). Pacti then shows how that lineage can be operationalized as a practical contract algebra with composition, quotient, and diagnosis operators (Incer et al., 2025).

These papers provide the closest mathematical ancestry for our contract semantics, but direct transfer is not automatic. Their components are usually physical, embedded, or software modules with explicit interface variables. Our components are research phases that exchange typed artifacts, prompt-derived instructions, validator outcomes, and provenance events. The contribution here is not to reinvent contract theory, but to define a restricted mapping from that theory into long-horizon orchestration and to state the boundary of the mapping clearly.

2.4 Formal Verification, Runtime Monitoring, and Assurance Boundaries

Formal-methods research is explicit about the difference between static guarantees and runtime checks (Wongpiromsarn et al., 2023; Leino, 2010). For the monitoring side we use the concise runtime-verification overview of Leucker and Schallhart as the representative first citation, because it isolates the monitor-versus-proof distinction that matters most for this paper (Leucker & Schallhart, 2008). Contract-based assurance for autonomous vessels and formal composition of robotic skills both show that modular assurance becomes tractable only after the designer chooses a disciplined interface language and a narrow validity regime (Torben et al., 2023; Pelletier et al., 2025). Neuro-symbolic systems and compiled tool interfaces reinforce the same lesson from another angle: expressive control becomes scientifically useful only when the interface between symbolic structure and learned components is explicit (Dinu et al., 2024; Khattab et al., 2023; Hitzler & Sarker, 2021; Austin et al., 2021).

The key gap, then, is not whether monitoring or proofs are valuable. It is how to partition claims for research orchestration. Existing agent papers often rely on validators, critiques, or execution traces, while formal-methods papers require a clean statement of what is proved and what is only monitored. Our assurance-boundary theorem closes exactly that gap for the typed regime studied here: local validators can certify the restricted property class they actually cover, but they do not imply unrestricted graph-level correctness.

Recent research-paper automation systems sharpen the motivation for that distinction. Frameworks such as PaperOrchestra show that specialized agent roles can coordinate literature synthesis, drafting, and revision over a long-horizon research workflow (Song et al., 2026). Those systems are important systems precedents because they expose explicit decomposition and artifact flow, but their central questions are about coordination quality, writing quality, and end-to-end task completion. Our manuscript asks a narrower theorem-facing question. Given a Quarks-style phase graph with typed artifacts, validators, and persistent state, which guarantees compose formally, which rely only on monitored evidence, and where does the assurance boundary actually sit? That reframing is what lets the paper inherit useful ideas from orchestration systems without overstating either their empirical results or the scope of classical contract theory.

3 Problem Setting and Contract Objective

We now define the formal setting used throughout the paper. The definitions combine inherited contract semantics with manuscript-specific objects required by the Quarks architecture context. At first use we distinguish which ingredients are borrowed and which are introduced in this work. This separation is methodologically important. A theorem paper about research orchestration can become misleading in two opposite ways: it can present borrowed contract notions as if they were novel, or it can hide manuscript-specific modeling choices inside apparently standard terminology. The present section avoids both errors. It states the ambient objects explicitly, then marks where the workflow branch, memory split, and objective function narrow the problem to the typed non-merge regime actually supported by the later proofs and audits.

Table 1: Core symbols and standing regime for the main theorem package. The table is compact, but it does not replace the inline definitions in the text. Its purpose is to make the validity regime explicit before the proofs: a finite active branch, typed artifact spaces, sound transport maps, and a non-merge memory discipline.

Symbol	Meaning	Regime
$\mathcal{G}_b$	active workflow branch	finite directed acyclic graph
X_v, Y_v	typed artifact domain and codomain	explicit nonempty sets
κ_v	local contract at node v	assumptions, guarantees, invariants, transition
α_{uv}	edge adapter	sound typed map when needed
β_v	predecessor aggregator	admissible typed combiner
L	provenance ledger	append-only global trace
W_b	working memory of branch b	branch-local mutable state
$\mathcal{P}_{\text{typed}}$	restricted positive property class	typed admissibility only

3.1 Workflow Branches, Contracts, and Typed Artifacts

Fix a finite nonempty index set of branches $B \subset \mathbb{N}_{\geq 1}$. For one active branch $b \in B$, let

$$\mathcal{G}_b = (V_b, E_b, \tau, \kappa, \alpha, \beta)$$

denote a finite directed acyclic graph. Here V_b is the node set, $E_b \subseteq V_b \times V_b$ is the edge set, and $\tau : V_b \to \mathcal{T}$ assigns a phase type from a finite type alphabet $\mathcal{T}$. The graph is directed and acyclic because the current theorem package concerns a single active execution branch rather than arbitrary cyclic orchestration or branch merges.

Each node $v \in V_b$ carries typed artifact spaces X_v and Y_v, both nonempty sets. We interpret X_v as the admissible input domain and Y_v as the typed output codomain for phase v. Following design-by-contract and assume-guarantee reasoning (Jazequel & Meyer, 1997; Benveniste et al., 2018; Incer et al., 2025), node v also carries a contract

$$\kappa_v = (A_v, G_v, I_v, f_v),$$

where $A_v \subseteq X_v$ is the assumption set, $G_v \subseteq Y_v$ is the guarantee set, $I_v \subseteq X_v \times Y_v \times \mathcal{M}_b$ is an invariant relation, and

$$f_v : A_v \times \mathcal{M}_b \to G_v \times \mathcal{M}_b$$

is the local transition in the active branch memory space $\mathcal{M}_b$. When $f_v(x, m) = (y, m')$, the semantics is that phase v may execute only on admissible inputs $x \in A_v$, must return an output $y \in G_v$, and must preserve the invariant $I_v(x, y, m')$.

For each edge $(u, v) \in E_b$, one of two cases holds. Either Y_u is directly compatible with the input expected by v, or there exists a manuscript-level adapter α_{uv} that maps predecessor outputs into an intermediate typed space $Z_{uv} \subseteq X_v$. When v has more than one predecessor, a manuscript-level aggregator β_v combines the incoming adapted artifacts into a single candidate input for v. These adapters and aggregators are not inherited as named objects from the contract literature; they are introduced here because the Quarks workflow admits multi-stage artifact transformation and validation. Table 1 collects the core symbols and exclusions for this setting so that the later theorem statements can be read against one explicit validity regime rather than against informal architectural intuition.

3.2 Memory Model, Validators, and the Contract Objective

The memory object is manuscript-defined because the implementation context motivates persistent context and branching, but does not itself provide a theorem-ready algebra. We write

$$\mathcal{M}_b := (L, \{W_{b'}\}_{b' \in B}),$$

where L is a global append-only provenance ledger and $W_{b'}$ is the working memory of branch b'. The positive regime restricts writes during execution on branch b to L and W_b; no transition is allowed to mutate $W_{b'}$ for $b' \neq b$. This is the precise meaning of branch-local memory used later.

Each node v also has a validator

$$\phi_v : X_v \times Y_v \times \mathcal{M}_b \to \{\text{pass}, \text{fail}, \text{abstain}\}.$$

The codomain includes abstain because validator coverage is limited. This follows the proof-versus-monitoring distinction emphasized by runtime verification and autonomous-systems assurance (Leucker & Schallhart, 2008; Wongpiromsarn et al., 2023; Torben et al., 2023), while the specific pass/fail/abstain codomain used here is manuscript-defined for the typed workflow regime. A local validator may confirm typed admissibility while remaining silent about broader semantic properties.

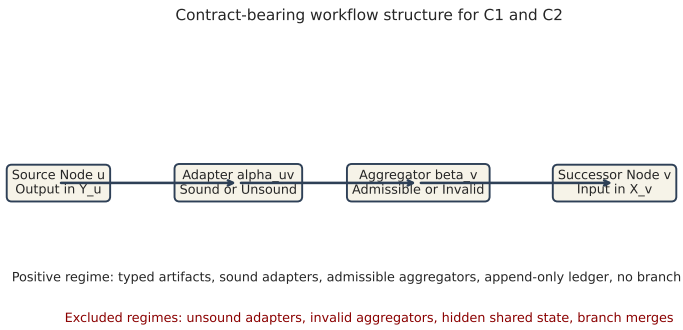

Figure 1: Contract-bearing workflow branch used in the proofs. The figure shows the active branch as a typed directed acyclic graph with phase-local contracts, optional edge adapters, successor aggregators, and the split between a global provenance ledger and branch-local working memory. Dashed exclusions mark the settings intentionally left outside the theorem regime, including branch merges, hidden shared mutable state, and free-form untyped execution; when audit counts are annotated in the figure, they describe finite audit support and not uncertainty about the theorem statements themselves.

Although the paper is proof-first rather than optimization-heavy, the selected path still benefits from an explicit objective and optimality criterion. For a fixed branch b, define the decision variable

$$\pi_b := (\prec_b, \{\alpha_{uv}\}_{(u,v)\in E_b}, \{\beta_v\}_{v\in V_b}),$$

where $\prec_b$ is a topological execution order. Let Π_b^{adm} be the feasible set of such triples satisfying the standing assumptions introduced below. We then define the manuscript-level contract objective

$$J_b(\pi_b) := \sum_{v\in V_b} \mathbf{1}[\phi_v(x_v, y_v, \mathcal{M}_b) = \mathrm{pass}]\,. \tag{1}$$

The associated optimality criterion is not unrestricted task quality. Instead, π_b is **contract-optimal** when $J_b(\pi_b) = |V_b|$. Under the theorem conditions, contract-optimality is equivalent to typed admissibility at every node. This is narrower than semantic usefulness, novelty, or human judgment, and we will keep that distinction explicit throughout the paper.

3.3 Standing Assumptions

The main theorem package relies on five assumptions.

A1 (finite active branch). $\mathcal{G}_b$ is finite and acyclic, and execution follows a topological order $\prec_b$. **A2 (typed local contracts).** For every $v \in V_b$, $A_v \subseteq X_v$, $G_v \subseteq Y_v$, and f_v is defined only on A_v. **A3 (sound transport).** For every $(u, v) \in E_b$, direct compatibility holds or there exists a sound adapter α_{uv} satisfying the condition in Definition 4.1. **A4 (admissible aggregation).** For each nonsource node v, the aggregator β_v maps all predecessor-compatible tuples into A_v. **A5 (branch-local append-only memory).** During positive-regime execution on branch b, updates may extend L and mutate only W_b; branch merges, deletions, and hidden shared state are excluded.

These assumptions are strong enough to support complete proofs but intentionally weak enough to leave meaningful negative witnesses. The validation artifacts show exactly what fails when any one of them is dropped.

4 Formal Contract Semantics

This section establishes the local-to-global typed admissibility theorem. We start with the manuscript-specific definitions of sound adapters and admissible aggregators, then prove the main result by induction over the topological order. The order of exposition mirrors the proof dependencies. The theorem is not an abstract black box that takes a graph and returns correctness. It depends first on transport semantics between nodes, then on the aggregation rule for multiple predecessors, and only then on an induction argument over the active branch. Writing the section this way makes it possible to see exactly which hypothesis is doing what work, and it also makes the later negative witnesses easier to interpret because each broken premise has already been isolated formally.

4.1 Sound Adapters and Admissible Aggregators

Definition 4.1 (Sound Adapter). *For an edge $(u, v) \in E_b$, a map $\alpha_{uv} : Y_u \to Z_{uv} \subseteq X_v$ is* sound *if, for every $y_u \in G_u$, one has $\alpha_{uv}(y_u) \in Z_{uv}$, and every value presented by v through that edge is of the form $\alpha_{uv}(y_u)$ for some $y_u \in G_u$.*

Definition 4.2 (Admissible Aggregator). *Let $\mathrm{pred}(v) = \{u \in V_b : (u, v) \in E_b\}$. For a nonsource node v, an aggregator*

$$\beta_v : \prod_{u \in \mathrm{pred}(v)} Z_{uv} \to X_v$$

is admissible *if $\beta_v(z) \in A_v$ for every predecessor tuple $z = (z_{uv})_{u \in \mathrm{pred}(v)}$ with $z_{uv} \in Z_{uv}$.*

Definitions 4.1 and 4.2 instantiate interface compatibility for typed research artifacts rather than physical ports or embedded controllers. That distinction matters because the aggregator is not a classical contract operator borrowed directly from Benveniste et al. (2018); it is a manuscript-level object introduced to model multi-input phases that combine validated predecessor artifacts. Figure 1 makes the role of these maps concrete.

The next equation records the branch-local update law assumed later.

$$U_v(L, W_b, x_v, y_v) = \big(L \sqcup \ell_v(x_v, y_v),\, U_v^W(W_b, x_v, y_v)\big), \tag{2}$$

where $\sqcup$ denotes append-only ledger extension, ℓ_v is the newly recorded provenance event, and no term in equation 2 mentions $W_{b'}$ for $b' \neq b$. Equation equation 2 is manuscript-defined and not inherited verbatim from the contract literature; it is the minimal update law compatible with the selected Quarks branch model.

4.2 Composition Theorem

We now state the main local-to-global result. The theorem claims only typed admissibility, not semantic correctness of the resulting research output.

Theorem 4.3 (Typed Contract Composition on Active Workflow DAGs). *Fix an active branch $b \in B$ satisfying A1–A5. Assume every source node v receives an initial input $x_v \in A_v$. For each edge $(u, v) \in E_b$, assume direct compatibility or a sound adapter α_{uv}. For each nonsource node v, assume an admissible aggregator β_v. Then there exists a topological execution of $\mathcal{G}_b$ such that, for every node $v \in V_b$, the realized input belongs to A_v, the realized output belongs to G_v, and the resulting execution policy π_b is contract-optimal in the sense of equation 1.*

Proof. Let $v_1, \ldots, v_n$ be any topological ordering of V_b, which exists by A1. We prove by induction on $k \in \{1, \ldots, n\}$ that after executing the prefix $v_1, \ldots, v_k$, every executed node v_i satisfies $x_{v_i} \in A_{v_i}$, $y_{v_i} \in G_{v_i}$, and the memory state remains in the positive regime.

For the base case, let v_1 be a source node. By hypothesis its initial input satisfies $x_{v_1} \in A_{v_1}$. Since f_{v_1} is defined on A_{v_1}, there exists $(y_{v_1}, \mathcal{M}_b') = f_{v_1}(x_{v_1}, \mathcal{M}_b)$ with $y_{v_1} \in G_{v_1}$. By A5 and equation 2, the update touches only L and W_b, so the positive memory regime is preserved. Hence the claim holds for $k = 1$.

Assume the statement holds through step $k - 1$, and consider node v_k. Every predecessor $u \in \mathrm{pred}(v_k)$ lies earlier in the topological order, so by the induction hypothesis each predecessor output satisfies $y_u \in G_u$. If u is directly compatible with v_k, the transported artifact already lies in the required predecessor-specific subset of X_{v_k}. Otherwise, by Definition 4.1, the adapted value $\alpha_{uv_k}(y_u)$ lies in $Z_{uv_k} \subseteq X_{v_k}$. Therefore the tuple of transported predecessor artifacts belongs to $\prod_{u \in \mathrm{pred}(v_k)} Z_{uv_k}$. By Definition 4.2, the aggregated value

$$x_{v_k} = \beta_{v_k}\big((z_{uv_k})_{u \in \mathrm{pred}(v_k)}\big)$$

lies in A_{v_k}. Since f_{v_k} is defined on A_{v_k}, execution produces $(y_{v_k}, \mathcal{M}_b'') = f_{v_k}(x_{v_k}, \mathcal{M}_b')$ with $y_{v_k} \in G_{v_k}$. Again A5 ensures that the update preserves the positive memory regime. This closes the induction.

After all n nodes have executed, every node satisfies its assumption and guarantee sets. Because the validators in the positive regime are defined to pass on exactly those typed admissibility events they cover, each term inside equation 1 equals 1, so $J_b(\pi_b) = |V_b|$. Thus π_b is contract-optimal. □

The proof is intentionally local. It never appeals to reviewer judgment, semantic novelty, or correctness of the scientific conclusions produced downstream. Its only conclusion is that typed admissibility composes when the transport maps, aggregators, and memory law remain inside the declared regime.

5 Memory Discipline and the Assurance Boundary

The contract-composition theorem is not enough by itself. Long-horizon orchestration also depends on how memory is partitioned and on what exactly validators observe. This section states two positive results and one negative theorem. The three statements belong together because they answer complementary questions. The memory lemma explains why provenance remains local and monotone in the declared regime. The assurance counterexample explains why that local discipline still does not certify unrestricted semantics. The restricted corollary then recovers the strongest safe positive implication. Taken together, they prevent the paper from drifting into a vague claim that "contracts plus validators" somehow solve open-ended correctness for research automation.

5.1 Branch-Local Provenance

Lemma 5.1 (Branch Isolation and Ledger Monotonicity)**.** *Suppose A5 holds and every node update is governed by equation 2. Then for any finite execution prefix on branch b, two properties hold simultaneously: (i) no execution step modifies $W_{b'}$ for $b' \neq b$; (ii) if L_t and L_{t+1} denote consecutive ledger states, then $L_{t+1} = L_t \sqcup \ell$ for some event ℓ, so L_t is a prefix of L_{t+1}.*

Proof. We argue by induction on the length t of the execution prefix. For $t = 0$, both properties are immediate. Assume they hold through time t, and consider the update produced by node v at time $t + 1$. Equation equation 2 shows that the next state is formed by two operations only: extension of the ledger by $\sqcup \ell_v(x_v, y_v)$, and application of U_v^W to W_b. No term in the update law mentions any $W_{b'}$ with $b' \neq b$, so branch isolation is preserved. Because $\sqcup$ is append-only by definition, the new ledger contains the old ledger as a prefix. Thus both properties hold at time $t + 1$. Induction completes the proof. □

Lemma 5.1 is modest but important. It says that the memory discipline used by the theorem package is not merely a narrative convenience. It is the exact structural condition that prevents one branch from silently rewriting another branch's working state while still allowing global provenance to accumulate.

5.2 Why Local Validator Passes Are Not Enough

To make the assurance boundary explicit, define the conjunction of local validator passes by

$$\Phi_{\mathrm{loc}}(\sigma) := \bigwedge_{v \in V_b} [\phi_v(x_v, y_v, \mathcal{M}_b) = \mathrm{pass}], \tag{3}$$

where σ denotes the realized branch execution trace. We also distinguish two property classes. The manuscript-defined class $\mathcal{P}_{\mathrm{typed}}$ contains only properties reducible to typed admissibility, declared coverage, and the branch-local memory discipline. A broader class $\mathcal{P}_{\mathrm{global}}$ may additionally refer to semantic agreement, scientific usefulness, or any property not fully observable to the validators.

Theorem 5.2 (Local Pass, Global Fail Counterexample)**.** *There exists a finite two-node branch execution σ such that $\Phi_{\mathrm{loc}}(\sigma)$ holds but some property $P \in \mathcal{P}_{\mathrm{global}}$ fails.*

Proof. Consider a branch with nodes $v_1 \to v_2$. Let

$$X_{v_1} = Y_{v_1} = X_{v_2} = Y_{v_2} = \{a, b\}.$$

Set $A_{v_1} = G_{v_1} = A_{v_2} = G_{v_2} = \{a, b\}$. Define both validators to check only type membership in $\{a, b\}$, so they pass on any typed input-output pair. Let node v_1 emit a, let the edge be directly compatible, and let node v_2 emit b. Because every input and output lies in the declared typed sets, both validators pass, and therefore $\Phi_{\mathrm{loc}}(\sigma)$ holds.

Now define a global property $P \in \mathcal{P}_{\mathrm{global}}$ by requiring semantic preservation of the proposition encoded upstream: $P(\sigma)$ holds only if the artifact emitted by v_2 preserves the semantic content of the artifact emitted by v_1. By construction $a \neq b$, and the local validators do not test semantic preservation. Hence $P(\sigma)$ fails although every local validator passes. This is exactly the desired counterexample. □

The point of Theorem 5.2 is not that validators are useless. It is that local passes do not imply more than the validators actually inspect. That distinction is often blurred in agent-system papers and needs to be explicit in a theorem paper.

Corollary 5.2.1 (Restricted Positive Implication)**.** *Let σ be an execution in the regime of Theorem 4.3 and Lemma 5.1. Assume additionally that validator coverage is exact for typed admissibility and branch-local provenance, meaning that local passes certify precisely the premises used in Theorem 4.3 and Lemma 5.1. Then*

$$\Phi_{\mathrm{loc}}(\sigma) \wedge C_{\mathrm{cover}} \wedge C_{\mathrm{adapter}} \wedge C_{\mathrm{memory}} \implies \mathcal{P}_{\mathrm{typed}}(\sigma). \tag{4}$$

Proof. Under the stated hypotheses, local validator passes certify that each node execution lies inside the typed assumption and guarantee sets and that the branch-local memory discipline is respected. Theorem 4.3 then yields typed admissibility for every node, while Lemma 5.1 yields branch isolation and ledger monotonicity. The class $\mathcal{P}_{\mathrm{typed}}$ is defined to contain exactly these properties and no stronger semantic claims. Therefore the conjunction in equation 4 implies $\mathcal{P}_{\mathrm{typed}}(\sigma)$. □

Corollary 5.2.1 is intentionally narrow. It does not rehabilitate unrestricted local-to-global assurance. It only states that once coverage, transport, and memory premises are made explicit, the typed properties genuinely do compose.

6 Validation Substrate and Evidence Construction

The core results above are proofs, but the manuscript also carries executed validation artifacts. Their role is not to estimate theorem truth. Their role is to test whether the formal regime is internally coherent, whether its exclusions are meaningful, and whether the Quarks implementation context provides honest architectural grounding for the symbols appearing in the paper. That distinction between proof and audit is central to the scientific framing. The proofs already establish the main logical claims. The executable artifacts instead answer a different question: if one tries to instantiate the theorem premises in concrete finite cases, do the positive and negative regimes behave the way the mathematics says they should? By keeping that question separate, the manuscript can use finite audits as evidence about formulation quality and claim boundaries without turning them into a surrogate for universal proof.

6.1 Finite Audit Families

The proof-validation stack materializes finite audit families for three claim groups: contract composition, branch-local memory, and the assurance boundary. The contract suite sweeps node counts in $\{2, 3, 4, 5\}$, maximum indegree in $\{1, 2\}$, adapter regimes in $\{\mathrm{sound}, \mathrm{unsound}\}$, aggregator regimes in $\{\mathrm{admissible}, \mathrm{inadmissible}\}$, and memory regimes in $\{\mathrm{branch\_local\_append\_only}, \mathrm{shared\_mutable}\}$ across fixed seeds $\{11, 23, 37, 53, 71\}$. The memory suite varies branch count, depth, ledger policy, and memory regime. The assurance suite varies property class, coverage regime, and graph size. These sweeps are finite symbolic stress tests, not probabilistic approximations to the proof obligations.

The finite audits still matter for two reasons. First, they expose missing assumptions quickly. If a proposed theorem silently relied on an inadmissible aggregator or on the absence of shared mutation, the negative witness archive would reveal the omission. Second, they make the failure boundary concrete for readers who need more than an abstract caveat. The symbolic report therefore records both positive closure counts and explicit out-of-regime witnesses.

6.2 Implementation Alignment

The theorem package also includes a read-only implementation-alignment audit. This audit does not execute the uploaded Quarks codebase. Instead, it checks whether the central manuscript symbols have code or document anchors in the architecture description. Typed phase contracts, validator-bearing payloads, and explicit phase manifests have field-level architectural evidence. By contrast, the branch-local append-only ledger and the restricted property class $\mathcal{P}_{\mathrm{typed}}$ remain manuscript-defined extensions. This separation is scientifically useful because it prevents the paper from claiming that every formal object is already present as a proved invariant of the codebase.

The separation between proof and audit is therefore deliberate. Proof establishes what follows from the mathematical assumptions. Finite audits test the adequacy of those assumptions against seeded examples and explicit counterexamples. Implementation alignment explains which objects are architecture-grounded and which are introduced to obtain a tractable theorem package.

Table 2: Theorem-audit summary for the three central claims. "Positive cases" denotes seeded finite executions satisfying the declared premises; the Wilson intervals summarize only the stability of those finite audit families and do not quantify uncertainty about the proofs themselves. "Negative witnesses" counts explicit out-of-regime or premise-violating constructions retained to keep the claim boundary visible.

Claim	Pos.	Pass	Low	High	Neg.	Status
Contract composition	40	40	0.9124	1.0000	280	supported
Memory discipline	45	45	0.9213	1.0000	180	supported
Assurance boundary	15	15	0.7961	1.0000	120	supported

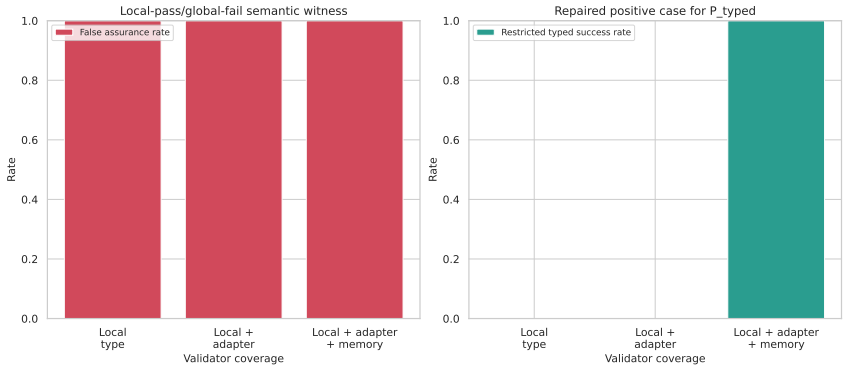

Figure 2: Assurance-boundary evidence summarized as a contrastive diagram. The left panel visualizes a local-pass/global-fail witness in which every validator certifies typed conformance while a broader semantic property still fails; the right panel shows the repaired regime in which explicit coverage, adapter, and memory premises reduce the target to the restricted property class $\mathcal{P}_{\mathrm{typed}}$. Any exact counts or interval annotations in the figure describe the finite audit family used to test the boundary, whereas the logical direction of the theorem and corollary comes from the proofs in Section 5.

7 Results and Boundary Analysis

We now summarize the theorem-audit evidence and interpret it in the same bounded language as the formal results. Table 2 should be read after Section 4 and Section 5, because the table is a compact summary of already defined claims rather than a substitute for the definitions and proofs.

Table 2 shows a consistent pattern. Positive cases close completely once the premises are enforced, and every theorem retains a substantial negative witness set when a premise is weakened. For contract composition, all 40 seeded positive cases pass, while 280 counterexamples arise from unsound adapters, inadmissible aggregators, or memory regimes that leave the theorem boundary. For the memory lemma, all 45 positive cases pass, while merge-enabled, shared-mutable, ledger-only, and stale-replay regimes generate 180 failures. For the assurance boundary, the positive corollary passes on all 15 restricted typed cases, while 120 false-assurance witnesses remain available when the property class or coverage regime is broadened.

The evidence is therefore strongest not where the claims are broadest, but where they are narrowest and most explicit. The composition theorem is strongest because its premises are structural and easy to audit. The memory lemma is slightly more manuscript-specific because the branch-local ledger split is motivated by architecture evidence rather than directly inherited from the codebase. The assurance result is strongest as a negative theorem: the counterexample is robust, while the positive implication is intentionally confined to $\mathcal{P}_{\mathrm{typed}}$.

Figure 2 is the most direct visual summary of the manuscript's claim boundary. It makes the two assurance layers visible at once: the left panel explains why local validator passes cannot be upgraded into unrestricted graph-level correctness, and the right panel shows the only positive implication we actually prove. This figure tests the main interpretive claim of the paper because it juxtaposes the exact same workflow ingredients under two different property classes. The difference is not the presence or absence of validators. The difference is whether the target claim asks for typed admissibility or for broader semantics outside validator coverage.

The workflow figure in Figure 1 serves a different evidentiary role. It is not a benchmark plot. It is a mechanistic summary of the structural hypotheses used in Theorem 4.3 and Lemma 5.1. In particular, it shows why the adapter and aggregator objects matter, and why the memory law is split into a global ledger plus branch-local working state. Without that split, the main theorem and lemma would either fail outright or require much stronger hidden premises.

Finally, the implementation-alignment audit constrains what we say about Quarks itself. The architecture context supports explicit typed phases, validators, and publication-facing artifact exports. It does not support a claim that the uploaded codebase already proves the append-only ledger law or the restricted property class. Treating those as manuscript-defined extensions is therefore a strength rather than a weakness. It keeps the paper at a defensible novelty boundary.

8 Discussion, Limitations, and Future Work

The paper's central payoff is conceptual discipline. It shows that a long-horizon research pipeline can support a real theorem package once the designer is willing to narrow the regime: finite active branch, typed artifacts, sound transport, admissible aggregation, and branch-local append-only memory. That is already useful because it separates what can be inherited from contract and assurance theory from what remains implementation-specific or empirical.

The same discipline also clarifies what the paper does *not* show. First, branch merges remain outside the theorem regime. The negative witness archive makes that exclusion substantive rather than cosmetic. Once one branch may overwrite or reconcile another branch's working state, Lemma 5.1 no longer applies directly and Theorem 4.3 loses the clean induction structure that depends on a fixed active branch. Second, the theorem package does not certify unrestricted semantic quality. Theorem 5.2 exists precisely to prevent that overclaim. A validator can certify that an artifact is well typed and provenance-consistent without certifying that it is insightful, novel, or correct in the ordinary scientific sense. Third, the memory law is architecture-grounded only indirectly. The Quarks context motivates preserved context and explicit phase artifacts, but the append-only ledger plus branch-local working memory split is still a mathematical choice introduced here to make provenance reasoning tractable.

These limitations suggest concrete future work rather than generic calls for "more experiments." A next theorem package could study guarded branch merges by replacing the single active branch with a merge algebra and explicit reconciliation contracts. Another direction is to refine validator coverage into a layered type system so that $\mathcal{P}_{\mathrm{typed}}$ can be expanded without collapsing into semantic overclaiming. A third direction is empirical rather than formal: once a suitably bounded semantic-quality substrate exists, one could test whether the contract-governed regime correlates with downstream research quality. None of those directions is needed to justify the present manuscript, but each one is required before broader claims become defensible.

9 Conclusion

This paper formulated a bounded semantics for long-horizon research orchestration and proved three results inside that regime: typed local admissibility composes over finite active workflow DAGs; branch-local working memory plus an append-only ledger preserves branch isolation and provenance monotonicity; and local validator passes do not imply unrestricted graph-level correctness, although they do imply a restricted typed property bundle under explicit coverage assumptions. The accompanying finite audits and implementation-alignment checks reinforce the same message. Strong claims are possible, but only after the workflow objects, transport maps, memory law, and validator coverage are stated precisely. For research-as-a-service systems, that is a practical and conceptual gain: it replaces vague architectural optimism with a theorem package whose scope, evidence, and failure modes are all legible.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, and David Dohan. Program synthesis with large language models. http://arxiv.org/abs/2108.07732v1, 2021. URL http://arxiv.org/abs/2108.07732v1.

Albert Benveniste, Benoıt Caillaud, Dejan Nickovic, Roberto Passerone, Jean-Baptiste Raclet, and Philipp Reinkemeier. Contracts for system design. https://doi.org/10.1561/1000000053, 2018. URL https://doi.org/10.1561/1000000053.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, and Lukas Gianinazzi. Graph of thoughts: Solving elaborate problems with large language models. https://doi.org/10.1609/aaai.v38i16.29720, 2024. URL https://doi.org/10.1609/aaai.v38i16.29720.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabriel dos Passos Gomes. Autonomous chemical research with large language models. https://doi.org/10.1038/s41586-023-06792-0, 2023. URL https://doi.org/10.1038/s41586-023-06792-0.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew Dickson White, and Philippe Schwaller. Augmenting large language models with chemistry tools. https://doi.org/10.1038/s42256-024-00832-8, 2024. URL https://doi.org/10.1038/s42256-024-00832-8.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, and Chi-Min Chan. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. http://arxiv.org/abs/2308.10848v3, 2023. URL http://arxiv.org/abs/2308.10848v3.

Sadia Sultana Chowa, Riasad Alvi, S M Asif Ur Rahman, Md Abdur Rahman, Mohaimenul Azam Khan Raiaan, and Md Rafiqul Islam. From language to action: a review of large language models as autonomous agents and tool users. https://doi.org/10.1007/s10462-025-11471-9, 2026. URL https://doi.org/10.1007/s10462-025-11471-9.

Marius-Constantin Dinu, Claudiu Leoveanu-Condrei, Markus Holzleitner, Werner Zellinger, and Sepp Hochreiter. Symbolicai: A framework for logic-based approaches combining generative models and solvers. http://arxiv.org/abs/2402.00854v4, 2024. URL http://arxiv.org/abs/2402.00854v4.

Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, and Nitesh V. Chawla. Large language model based multi-agents: A survey of progress and challenges. http://arxiv.org/abs/2402.01680v2, 2024. URL http://arxiv.org/abs/2402.01680v2.

Pascal Hitzler and Md Kamruzzaman Sarker. Neuro-symbolic artificial intelligence: The state of the art. https://doi.org/10.3233/faia342, 2021. URL https://doi.org/10.3233/faia342.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, and Ceyao Zhang. Metagpt: Meta programming for a multi-agent collaborative framework. http://arxiv.org/abs/2308.00352v7, 2023. URL http://arxiv.org/abs/2308.00352v7.

Inigo Incer, Apurva Badithela, Josefine B. Graebener, Piergiuseppe Mallozzi, Ayush Pandey, and Nicolas Rouquette. Pacti: Assume-guarantee contracts for efficient compositional analysis and design. https://doi.org/10.1145/3704736, 2025. URL https://doi.org/10.1145/3704736.

J.-M. Jazequel and Bertrand Meyer. Design by contract: the lessons of ariane. https://doi.org/10.1109/2.562936, 1997. URL https://doi.org/10.1109/2.562936.

Bernhard Kaiser, Raphael Weber, Markus F. Oertel, Eckard Bode, Behrang Monajemi Nejad, and Justyna Zander. Contract-based design of embedded systems integrating nominal behavior and safety. https://doi.org/10.7250/csimq.2015-4.05, 2015. URL https://doi.org/10.7250/csimq.2015-4.05.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, and Sri Vardhamanan. Dspy: Compiling declarative language model calls into self-improving pipelines. http://arxiv.org/abs/2310.03714v1, 2023. URL http://arxiv.org/abs/2310.03714v1.

K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. https://doi.org/10.1007/978-3-642-17511-4_20, 2010. URL https://doi.org/10.1007/978-3-642-17511-4_20.

Martin Leucker and Christian Schallhart. A brief account of runtime verification. https://doi.org/10.1016/j.jlap.2008.08.004, 2008. URL https://doi.org/10.1016/j.jlap.2008.08.004.

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. http://arxiv.org/abs/2303.17760v2, 2023. URL http://arxiv.org/abs/2303.17760v2.

Xinyi Li, S. Wang, Siqi Zeng, Yu Wu, and Yi Yang. A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. https://doi.org/10.1007/s44336-024-00009-2, 2024. URL https://doi.org/10.1007/s44336-024-00009-2.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. http://arxiv.org/abs/2408.06292v3, 2024. URL http://arxiv.org/abs/2408.06292v3.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, and Sarah Wiegreffe. Self-refine: Iterative refinement with self-feedback. http://arxiv.org/abs/2303.17651v2, 2023. URL http://arxiv.org/abs/2303.17651v2.

Baoluo Meng, Daniel Larraz, Kit Siu, Abha Moitra, John Interrante, and William Smith. Verdict: A language and framework for engineering cyber resilient and safe system. https://doi.org/10.3390/systems9010018, 2021. URL https://doi.org/10.3390/systems9010018.

Joon Sung Park, Joseph O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. https://doi.org/10.1145/3586183.3606763, 2023. URL https://doi.org/10.1145/3586183.3606763.

Baptiste Pelletier, Charles Lesire, and Karen Godary-Dejean. A formal framework for the specification and verification of robotic skills composition for autonomous behaviors. https://doi.org/10.1016/j.robot.2025.105041, 2025. URL https://doi.org/10.1016/j.robot.2025.105041.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, and Jiahao Li. Chatdev: Communicative agents for software development. http://arxiv.org/abs/2307.07924v5, 2023. URL http://arxiv.org/abs/2307.07924v5.

Jean-Baptiste Raclet, Eric Badouel, Albert Benveniste, Benoıt Caillaud, Axel Legay, and Roberto Passerone. A modal interface theory for component-based design. https://doi.org/10.3233/fi-2011-416, 2011. URL https://doi.org/10.3233/fi-2011-416.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. http://arxiv.org/abs/2303.11366v4, 2023. URL http://arxiv.org/abs/2303.11366v4.

Yiwen Song, Yale Song, Tomas Pfister, and Jinsung Yoon. Paperorchestra: A multi-agent framework for automated ai research paper writing. http://arxiv.org/abs/2604.05018v1, 2026. URL http://arxiv.org/abs/2604.05018v1.

Tobias Rye Torben, Øyvind N. Smogeli, Jon Arne Glomsrud, Ingrid Bouwer Utne, and Asgeir J. Sørensen. Towards contract-based verification for autonomous vessels. https://doi.org/10.1016/j.oceaneng.2023.113685, 2023. URL https://doi.org/10.1016/j.oceaneng.2023.113685.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, and Yuke Zhu. Voyager: An open-ended embodied agent with large language models. http://arxiv.org/abs/2305.16291v2, 2023a. URL http://arxiv.org/abs/2305.16291v2.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, and Jingsen Zhang. A survey on large language model based autonomous agents. http://arxiv.org/abs/2308.11432v7, 2023b. URL http://arxiv.org/abs/2308.11432v7.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, and Fei Xia. Chain-of-thought prompting elicits reasoning in large language models. http://arxiv.org/abs/2201.11903v6, 2022. URL http://arxiv.org/abs/2201.11903v6.

Tichakorn Wongpiromsarn, Mahsa Ghasemi, Murat Cubuktepe, Georgios Bakirtzis, Steven Carr, and Mustafa O. Karabag. Formal methods for autonomous systems. https://doi.org/10.1561/9781638282730, 2023. URL https://doi.org/10.1561/9781638282730.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, and Erkang Zhu. Autogen: Enabling next-gen llm applications via multi-agent conversation. http://arxiv.org/abs/2308.08155v2, 2023. URL http://arxiv.org/abs/2308.08155v2.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, and Karthik Narasimhan. Swe-agent: Agent-computer interfaces enable automated software engineering. http://arxiv.org/abs/2405.15793v3, 2024. URL http://arxiv.org/abs/2405.15793v3.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, and Karthik Narasimhan. React: Synergizing reasoning and acting in language models. http://arxiv.org/abs/2210.03629v3, 2022. URL http://arxiv.org/abs/2210.03629v3.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, and Yuan Cao. Tree of thoughts: Deliberate problem solving with large language models. http://arxiv.org/abs/2305.10601v2, 2023. URL http://arxiv.org/abs/2305.10601v2.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. http://arxiv.org/abs/2310.04406v3, 2023. URL http://arxiv.org/abs/2310.04406v3.

Table 3: Extended symbol glossary for the formal regime. The table adds domain and exclusion notes that would interrupt the main exposition if repeated inline. Each entry should be read together with the first-use definition in the main manuscript, especially when the symbol refers to a manuscript-defined object rather than inherited contract notation.

Symbol	Meaning	Domain and exclusions
$\mathcal{G}_b$	active workflow branch	finite DAG for one active branch; excludes cycles and merges
κ_v	local contract	inherited contract vocabulary specialized to typed research artifacts
α_{uv}	edge adapter	manuscript-defined typed transport map; must preserve admissibility
β_v	aggregator	manuscript-defined multi-input combiner; valid only for predecessor tuples in declared typed subsets
L	global ledger	append-only provenance trace; no deletion or compaction in the positive regime
W_b	branch memory	mutable state of the active branch only; no writes to $W_{b'}$ for $b' \neq b$
Φ_{loc}	local pass conjunction	conjunction of observable validator outcomes; not a semantic oracle
$\mathcal{P}_{\mathrm{typed}}$	restricted property class	typed admissibility and provenance-only claims; excludes semantic usefulness and novelty

A Equation Provenance and Extended Symbol Notes

This appendix records the provenance and applicability of the main equations. Equation equation 1 is manuscript-defined. It is not taken verbatim from any source because the paper does not optimize a generic task reward. Instead, it formalizes the contract-specific notion of full typed coverage over the active branch. The decision variable π_b contains only the execution order, adapters, and aggregators needed for the theorem package; it does not represent unrestricted policy learning or global search. The feasible set Π_b^{adm} therefore depends on A1–A5 and is empty whenever merges, hidden shared state, or untyped execution are enabled.

Equation equation 2 is also manuscript-defined. Its provenance comes from two directions. The Quarks architecture context motivates preserved context, explicit artifacts, and phase-local execution state, while the contract and runtime-verification literature motivates keeping update laws explicit (Benveniste et al., 2018; Leucker & Schallhart, 2008; Wongpiromsarn et al., 2023). The equation itself narrows that broad motivation into a precise append-only law. Its applicability is therefore limited to the non-merge regime. Once ledger compaction, deletion, or cross-branch overwrite is admitted, the equation no longer describes the system and Lemma 5.1 ceases to apply.

Equation equation 3 inherits the idea of monitor conjunction from runtime verification, but the specific symbol Φ_{loc} is introduced in this manuscript to make the assurance boundary concise. The critical point is that Φ_{loc} ranges only over observable validator outcomes. It does not quantify over inaccessible semantic properties or over external facts that no validator checks. For that reason, equation 3 is a suitable premise for equation 4 only after the coverage assumptions are made explicit.

Equation equation 4 is the most delicate equation in the paper because it is simultaneously positive and narrow. Its left-hand side contains three conjuncts beyond Φ_{loc}: coverage, adapter, and memory discipline. Each conjunct corresponds to a real failure mode in the negative witness archive. Coverage blocks false-assurance upgrades; adapter soundness blocks type-preserving but semantically malformed transport; and memory discipline blocks silent cross-branch interference. The equation therefore states an implication that is valid exactly because the regime is narrow. Readers should not interpret it as a template for unrestricted global correctness.

For convenience, Table 3 restates the most heavily used symbols together with their domains and exclusions. The table supplements, but does not replace, the first-use definitions in the main text.

B Expanded Proof Structure

The main-text proofs are complete, but this section records the proof architecture in more detail so that the induction dependencies and failure boundaries are explicit. The proof of Theorem 4.3 depends on three structural facts. The first

is that every predecessor of a node appears earlier in the topological order; this is the only place where acyclicity is used. The second is that every transported predecessor artifact lies in a declared typed subset, either directly or after a sound adapter. The third is that the aggregator maps the full predecessor tuple into the assumption set A_v. The theorem fails immediately if any one of those facts is dropped. Without acyclicity there is no induction order. Without adapter soundness a predecessor may emit a typed value that still lies outside the successor's admissible domain. Without aggregator admissibility the successor can receive a tuple of individually compatible inputs whose combination is itself inadmissible.

The inductive invariant in the proof can therefore be stated explicitly. After executing the prefix $v_1, \ldots, v_k$, every executed node satisfies its contract, the memory remains in the positive regime, and every edge leaving the prefix transports an artifact that is already in the typed subset expected by the remaining proof steps. This stronger invariant explains why the theorem is genuinely compositional. The induction hypothesis is not merely that past nodes were correct in isolation; it is that their outputs remain usable by future nodes under the declared transport semantics.

Lemma 5.1 uses a simpler induction because the update law has a simpler structure. Here the key observation is that branch isolation is a syntactic property of equation 2. No term in the right-hand side can mention an inactive branch memory. This is stronger than a semantic promise that inactive branches will probably not be touched. If a future extension introduces a merge operator, the update law itself must change, and with it the applicability of the lemma.

Theorem 5.2 is a constructive negative result, so the proof structure is different. The proof does not rely on induction. Instead, it isolates the smallest possible witness family in which the validator alphabet is too weak for the target semantic property. The two-node construction is enough because the failure has nothing to do with graph size. It arises from a mismatch between what the validators inspect and what the claimed property requires. This construction is the assurance analogue of a classic counterexample in program logic: a local proof rule is sound only for the predicates that actually occur in the rule.

Corollary 5.2.1 then becomes almost tautological, but only because the property class is defined carefully. A weakly written corollary would fail by silently broadening $\mathcal{P}_{\mathrm{typed}}$. The present corollary avoids that error by defining the property class after the theorem and lemma, not before them. The reader can therefore audit the inclusion relation directly: if a desired property mentions semantic novelty, reviewer acceptance, or branch-merge consistency, it is not in $\mathcal{P}_{\mathrm{typed}}$.

One additional consequence is worth stating. The contract objective equation 1 is not an optimization wrapper pasted onto the theorem package. It is a compact restatement of the same inductive logic. A policy is contract-optimal exactly when the theorem guarantees that every node is admissible and every validator in the declared coverage set passes. If coverage changes, the interpretation of the objective changes too. This is another reason the assurance-boundary result is necessary.

C Counterexamples and Failure Regimes

The negative witness archive is large because the theorem package is deliberately honest about where it fails. Rather than reproduce every individual witness, we group them into four families that carry the scientific content.

The first family concerns unsound transport. In these cases each predecessor node may satisfy its own guarantee set, but the adapter either maps into the wrong typed subset or fails to preserve the invariant needed by the successor. The immediate result is that Theorem 4.3 no longer applies. The counterexample does not require semantic subtlety; a typed mismatch is enough. This family justifies why soundness is not a cosmetic adjective in Definition 4.1. It is the exact point at which local admissibility can stop composing.

The second family concerns inadmissible aggregation. Here every predecessor artifact can be individually well formed, yet the combined tuple can still fall outside A_v. This matters operationally because research phases often combine evidence from several upstream steps. A system that validates each input independently but never validates the aggregate can appear locally sound while still violating the successor contract. The finite audit suite contains many such failures because the combinatorial space of predecessor tuples is precisely where hidden assumptions often surface.

The third family concerns memory discipline. Shared mutable state, ledger-only regimes, stale replay, and merge-enabled execution all generate failures for different reasons. Shared mutation breaks branch isolation because an inactive branch can affect the active branch without an explicit transition. Ledger-only regimes preserve provenance but lose the working-copy semantics needed for local repair. Stale replay reintroduces past artifacts without preserving freshness assumptions. Merge-enabled regimes are the most important boundary case: they show that the proof package is not merely incomplete on merges, but structurally mismatched to them.

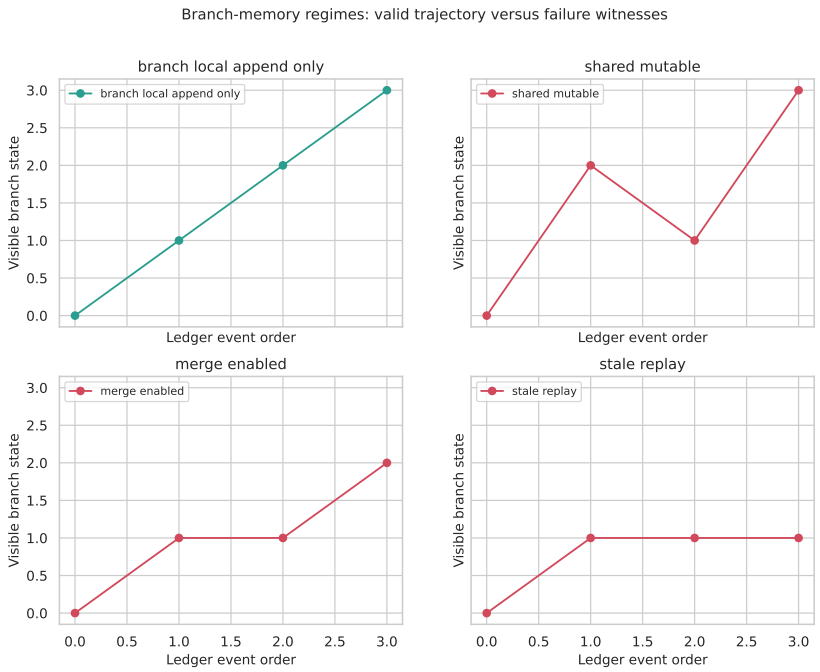

Figure 3: Memory-regime comparison used to interpret Lemma 5.1. The figure contrasts the positive branch-local append-only setting with stale-replay, shared-mutable, and merge-enabled regimes by making visible which writes are local, which provenance events extend monotonically, and where invalid cross-branch interference appears. The panels should be read as witness classes rather than benchmark curves: any counts or interval annotations summarize the finite audit family, whereas the decisive distinction is whether the update law satisfies the structural restriction in Equation equation 2.

The fourth family is the assurance boundary itself. Figure 2 gives the compact version, but the negative archive is scientifically useful because it prevents a common rhetorical slide. Once a paper has validators, traces, and explicit artifacts, it is tempting to speak as if the system is globally checked. Theorem 5.2 and its witness family show why that language is misleading. The archive keeps the failure visible so that later empirical or formal extensions can be evaluated against a real baseline rather than against an erased limitation.

Figure 3 plays an analogous role for the memory lemma. It makes the valid and invalid regimes visually distinct and therefore serves as an appendix-only diagnostic figure rather than a decorative add-on. The main text already states the lemma. The figure helps readers audit why the lemma fails exactly where it does.

D Implementation Alignment and Reproducibility

The theorem package was supported by a proof-oriented validation stack with fixed seeds $\{11, 23, 37, 53, 71\}$, finite audit families, and a read-only implementation-alignment pass. This section records those details because they affect interpretation even though they are not the scientific contribution.

The validation program materialized three families of typed workflow fixtures: contract cases, memory cases, and assurance cases, plus a stress suite for out-of-regime diagnostics. The executable audit remained lightweight enough for a bounded local validation run: a short smoke stage checked imports, fixture generation, and report writing before the full symbolic sweep. Confidence intervals were Wilson intervals over seeded finite case families. Their interpretation must remain narrow: they summarize the stability of the generated audit family and not the logical strength of a proof. In particular, a 100% pass rate on the positive cases in Table 2 does not strengthen Theorem 4.3; it only confirms that the executable checker agrees with the stated premises on the generated family.

The implementation-alignment audit was equally narrow. It read the architecture manifest, the requirements specification, the contract registry, the phase metadata, the publication-export documentation, and the validation entry points, then mapped manuscript symbols to code or document anchors when available. The resulting table, shown in Table 4,

Table 4: Implementation-alignment summary for the manuscript symbols. The table distinguishes field-level architectural evidence from narrative evidence and manuscript-defined extensions. Its purpose is to prevent overclaiming: a symbol with only narrative or manuscript support may still be a useful formal object, but it should not be described as if it were already an implemented invariant.

Symbol	Support	Anchor type	Interpretation
typed phase contracts	field-level	document	architecture specifies phases with explicit inputs, outputs, contracts, and context scope
validation payloads	field-level	code	registry exposes typed validation outputs and validator-bearing contracts
phase manifests	field-level	document	phase metadata records dependencies and semantic expectations
preserved context	narrative	document	architecture motivates persistent context but does not itself prove the memory lemma
semantic validators	field-level	code	validators are first-class architecture elements
export flow	narrative	document	documentation supports reproducible paper-oriented artifact packaging
branch-local append-only ledger	manuscript-defined	manuscript	formal memory split introduced in this work
$\mathcal{P}_{\mathrm{typed}}$	manuscript-defined	manuscript	restricted property class introduced to state the assurance corollary

confirms that typed phases, validators, and export-oriented artifacts have architecture support, while the ledger split and restricted property class remain manuscript-defined. That is the correct outcome. A proof paper should not pretend that every useful formal object already lives in the codebase.

Reproducibility of the proofs themselves is therefore two-layered. At the mathematical layer, reproducibility means that every assumption needed by the proof is written explicitly and every result has a complete proof block. At the audit layer, reproducibility means that the generated cases, seeds, counts, and negative witnesses can be replayed by the validation shell without changing the uploaded codebase. The manuscript is careful not to conflate these two meanings.

E Extended Audit Interpretation

The theorem-audit counts deserve one final interpretive remark. A naive reading might ask why 280 negative witnesses for contract composition coexist with a fully supported claim. The answer is that the negative witnesses are not failed proof attempts inside the theorem regime. They are deliberate tests of adjacent regimes in which a premise is removed. In fact, their abundance is evidence that the theorem boundary is meaningful. If no negative witnesses existed after premise removal, the assumptions would be suspiciously weak or redundant.

The same reasoning applies to the memory and assurance results. The positive memory cases all pass because they obey the branch-local append-only law. The negative cases exist because they do not. For the assurance theorem, the broad negative result is the point, while the narrow positive corollary shows exactly how much assurance can be recovered. This asymmetry is scientifically healthy. It keeps the theorem package aligned with what validators and typed contracts can actually support.

For readers concerned with broader system claims, the correct takeaway is therefore layered. The contract theorem gives a stable local-to-global statement for typed admissibility. The memory lemma gives a stable state-discipline statement for one active branch. The assurance theorem warns against inflating either result into a semantic oracle. Together they provide a foundation for research orchestration, but not a final theory of autonomous research quality.