PROTOGUARD: SAFE AND TOKEN-EFFICIENT LLM AGENT INTERACTION THROUGH FORMAL PROTOCOLS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agent teams are commonly coordinated through workflow graphs, yet structural and input/output type checks do not ensure **interaction safety**: agents may deadlock, fail to propagate decisions, or act prematurely. Even with a sound interaction plan, runtime deviations remain possible for LLM agents. We introduce PROTOGUARD, a framework that specifies multi-agent interactions as formal protocols and connects static validation with runtime governance. Building on *multiparty session types* (MPST), a formalism for modelling interactions with rigorous guarantees of communication safety and progress, PROTOGUARD uses an LLM to draft a global protocol, checks and refines it for formal validity under MPST, and derives a role-specific contract for each agent. The resulting contracts provide *protocol-governed execution* through agent prompts, deterministic monitoring, pre-delivery gating, and protocol-aware scheduling. Runtime enforcement keeps agent interaction within the validated protocol, preserving its guarantees by construction without constraining task-specific agent decisions. We evaluate protocol-governed execution on diverse multi-agent benchmarks and LLMs. Across the evaluated benchmark–model settings, macro-averaged goal-completion rate increases from 4.1% under the skills-only baseline (role-specific instructions without protocol support) to 98.9% under the full configuration, while runtime enforcement eliminates delivered protocol violations. Mean execution time decreases from 249.6 s to 47.6 s, and runtime token cost per trial run falls by 91.5%. These results demonstrate that validated and enforced interaction protocols can improve the correctness, safety, and efficiency of LLM-agent coordination, suggesting formal specifications as a principled execution layer for stochastic LLM agents.

1 INTRODUCTION

LLM agents are increasingly organised into teams in which specialised roles communicate and act toward a shared objective. Yet strong individual capabilities do not by themselves ensure reliable multi-agent coordination (Zhang et al., 2026; Cemri et al., 2025), which requires *coordination correctness*, including communication structure, interaction ordering, decision propagation, and branch-dependent behaviour. Contemporary agent systems commonly encode such coordination through role-specific instructions or workflow graphs (Wu et al., 2023; LangChain, Inc., 2024; Microsoft, 2025). Tools such as Microsoft Agent Framework and Google Cloud Workflows (Microsoft, 2025; Google Cloud, 2026) support specifying and executing workflows, but their validation does not in itself establish the communication dependencies required for coordination correctness.

Consider the SDLC release-gate task in Figure 1(a), where seven LLM agents with role personas adapted from GitHub's Awesome Copilot collection (GitHub, 2026b) coordinate a software release. An Author submits an artifact; four independent reviewers assess quality, security, architecture, and responsible AI; a Merger aggregates their verdicts; and DevOps executes the decision. The release policy permits deployment with zero objections, triggers revision and another review round with one objection, and withholds the release with two or more. Because reviews are asynchronous, the Quality and Responsible-AI `OK`s may arrive before the Security and Architecture `NO`s. Without a protocol constraint and enforcement barrier requiring all four verdicts, the Merger may treat the currently available verdicts as complete and choose `Deploy`. The late `NO`s cannot retract a release that exposes an unauthenticated administrative endpoint and a dependency cycle. The danger is

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

therefore a premature, irreversible action caused by incomplete fan-in, not merely an uninformed role waiting.

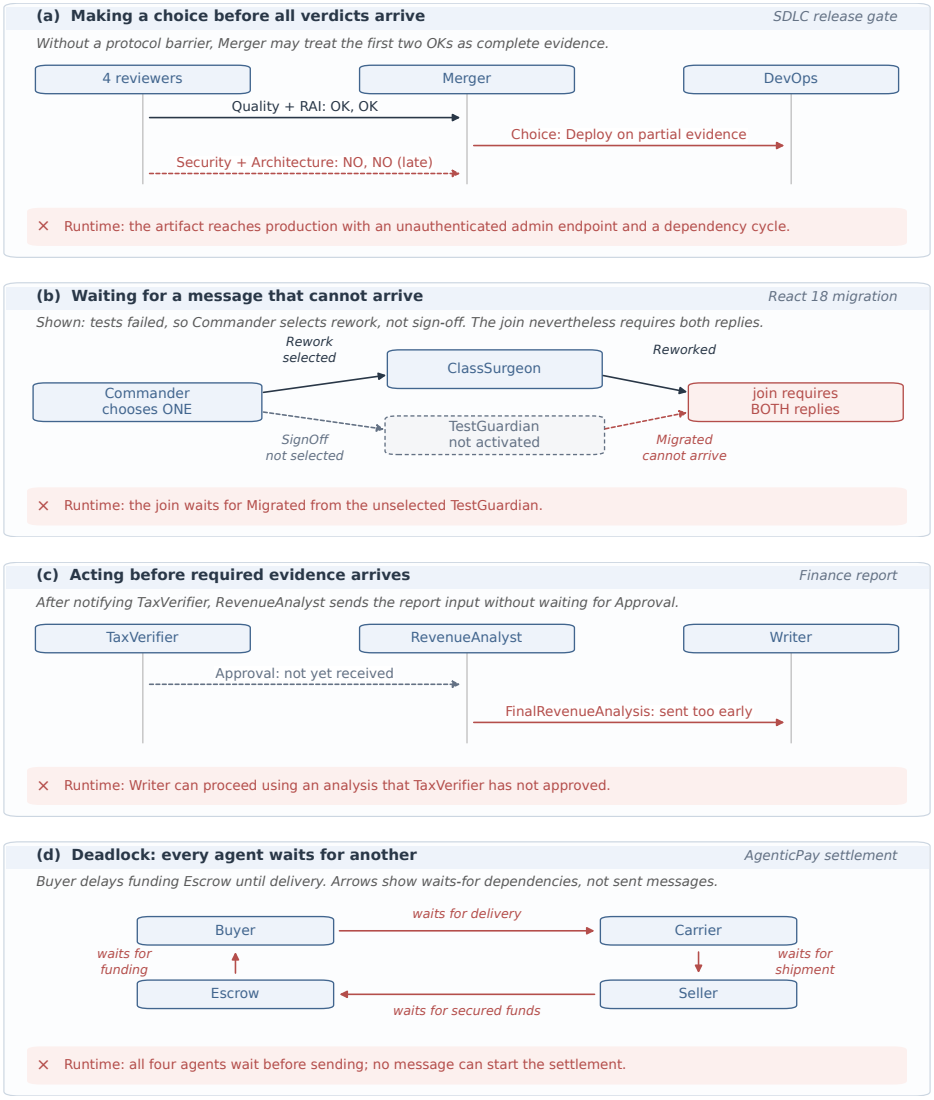

Figure 1: Coordination failures: (a) making a choice before asynchronous review fan-in completes; (b) waiting for a reply that cannot arrive; (c) reporting before approval; (d) circular waiting deadlock. Details in Appendix C.

Figure 1 further shows the limitations of workflow-level validation for diverse interaction structures, including impossible waits, premature actions, and circular dependencies. Representing such interactions as *global protocols* makes the communication dependencies among roles explicit and amenable to formal analysis. This motivates treating multi-agent coordination as a formal specification rather than solely as workflow structure or prompt context. Multiparty session types (MPST) (Honda et al., 2008; 2016; Scalas & Yoshida, 2019) provide such a formalism, specifying interactions among multiple roles and supporting static validation of communication safety and progress for conforming executions. These communication guarantees, however, presuppose participant conformance, whereas stochastic LLM agents may deviate from the prescribed interaction during execution. A valid protocol alone is therefore insufficient: reliable multi-agent coordination requires both static validation of the interaction protocol and runtime governance of agent execution.

We introduce **PROTOGUARD**, a framework for validated and governed LLM-agent interaction through a formal interaction specification. PROTOGUARD takes an LLM-drafted MPST global protocol, with diagnostics from an MPST toolchain guiding refinement until static validation succeeds. The resulting global protocol is projected into role-local contracts that govern agent interaction at runtime. These contracts determine agent instructions and provide the basis for deterministic monitoring, enforcement, and protocol-aware scheduling, linking the validated specification to protocol-governed execution. PROTOGUARD governs agent interaction relative to the validated protocol, not the semantic correctness of agent-generated content.

We evaluate protocol-governed execution on four multi-agent benchmarks spanning diverse application scenarios and multiple underlying LLMs. Across these settings, validated and enforced interaction protocols yield substantial gains in coordination correctness, interaction safety, and efficiency, while runtime enforcement prevents protocol violations from reaching recipients. On the challenging SDLC example, goal-completion rate increases from 4.7% with role-specific instructions to 95.6% under the full PROTOGUARD configuration, while runtime token cost per successful run falls from approximately 1.41M to 33.7K, a 97.6% reduction. Collectively, these findings support formal interaction specifications as a basis for governing stochastic multi-agent execution.

Our contributions are threefold. First, we develop a *validator-guided protocol construction pipeline* that integrates LLM drafting, formal static validation, and diagnostic-guided refinement. Second, we introduce *protocol-governed execution*, in which deterministic runtime controls for monitoring, enforcement, and scheduling are derived from role-local contracts. Finally, we provide a *systematic evaluation* across diverse multi-agent tasks and models, measuring coordination correctness, interaction safety, and efficiency and analysing the effects of individual runtime mechanisms.

2 RELATED WORK

Multi-agent LLM Coordination and Orchestration. Multi-agent LLM systems coordinate specialised agents through conversational interaction (Wu et al., 2023; Hong et al., 2024) and graph- or workflow-based orchestration (LangChain, Inc., 2024; Microsoft, 2025). Industrial methodologies such as AWS AI-DLC (Amazon Web Services, 2026) likewise structure software delivery as phase-gated agent workflows, enforcing stage ordering and human approval through deterministic hooks. Recent work has also explored structured procedural representations for individual agents: Graph-SkillEvo (Sun et al., 2026) represents agent skills as graph-structured natural-language artifacts and optimises their workflow structure through evolutionary search. Other studies examine coordination under varying communication topologies and team sizes (Grötschla et al., 2025), distributed information (Zhang et al., 2026), asynchronous long-horizon planning (Gonzalez-Pumariega et al., 2025), and resource contention under different communication structures and action protocols (Hasan & BusiReddyGari, 2026). Complementary failure analyses characterise recurring breakdowns in inter-agent coordination (Cemri et al., 2025). Workflow design and empirical analysis alone do not establish communication safety or deadlock freedom. PROTOGUARD addresses this gap through statically validated multiparty protocols that govern execution.

Formal Specification and Enforcement for LLM Agents. A growing body of work applies specifications and runtime constraints to LLM agents. AgentSpec (Wang et al., 2026) provides programmable rules and enforcement strategies over agent actions, while Agent-C (Kamath et al., 2025) targets temporal safety properties over tool-call sequences. Path-based governance (Kaptein et al., 2026) constrains behaviour along execution paths, and VIGIL (Li et al., 2026) checks traces against temporal, argument, and value-flow policies. VeriGuard (Miculicich et al., 2025) more closely couples offline specification and verification with online monitoring, while ST-WebAgentBench (Levy et al., 2026) evaluates policy compliance of individual agents. PROTOGUARD differs in its multiparty formulation: role-specific constraints are derived from a checked team-level interaction rather than specified independently for individual agents or execution traces. TraceFix (Xia et al., 2026) model-checks LLM-written per-agent PlusCal with TLC but leaves step order to prompts at runtime; PROTOGUARD, using local role contracts projected from the global protocol, deterministically gates every message and value-dependent decision before delivery and invokes only enabled roles.

Formal Protocols for Distributed Interaction. Our formal foundation is Multiparty Session Types (MPST) (Honda et al., 2008; 2016), which relate global protocols to participant-local behaviour and provide communication safety and progress for conforming executions. Monitored session

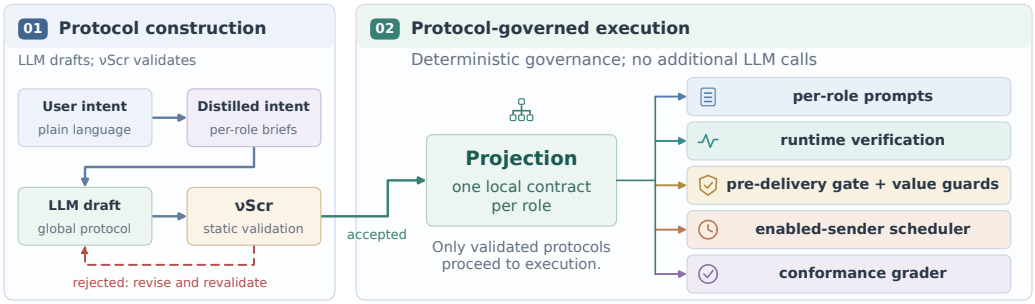

Figure 2: The PROTOGUARD framework. Protocol construction refines LLM-drafted global protocols through static validation; protocol-governed execution projects the validated protocol into role-local contracts for agent prompting and deterministic runtime governance.

types (Bocchi et al., 2013; 2017) facilitate runtime conformance checking through local monitoring. Session types further enable constraints over communicated values (Bocchi et al., 2010; Chen & Honda, 2012), while gradual session types (Igarashi et al., 2017) mediate between typed protocols and dynamically typed endpoints. PROTOGUARD brings this global-to-local discipline to stochastic LLM-agent coordination, supported by practical MPST tooling (Yoshida et al., 2021).

3 MULTIPARTY SESSION TYPES

Our framework, PROTOGUARD, builds on *multiparty session types* (MPST) (Honda et al., 2008; 2016; Scalas & Yoshida, 2019), a formalism for describing communication among multiple participants with established guarantees of communication safety and progress. An MPST *global protocol* specifies the participating roles, message exchanges, sequencing, branching, and recursion. A global protocol is *projectable* when a *local protocol* can be derived for each role, characterising the messages that role may send and receive throughout the interaction.

Projectability requires compatibility among the role-local protocols. In particular, when a participant's behaviour depends on a choice made by another role, the selected branch must be observable to that participant, a property known as *knowledge of choice*. For a well-formed global protocol, conforming participants do not become blocked waiting for messages that cannot be produced (Honda et al., 2008; 2016). Independent interactions remain unordered, preserving concurrency permitted by the global protocol. Runtime conformance can be verified against projected local protocols (Bocchi et al., 2013; 2017), while stateful assertions (Chen & Honda, 2012) allow protocol constraints to depend on values accumulated across interactions.

We instantiate MPST with νSCR (Yoshida et al., 2021), a practical protocol specification and verification toolchain supporting validation and projection. Global protocols accepted by νSCR are projected into role-local protocols that form the contracts used by PROTOGUARD for agent configuration and runtime governance in Section 4.

4 PROTOGUARD: PROTOCOL-GOVERNED LLM AGENT COORDINATION

As shown in Figure 2, PROTOGUARD operates in two phases: *protocol construction* produces a validated global protocol, and *protocol-governed execution* derives role-local contracts for agent configuration, monitoring, enforcement, scheduling, and conformance grading. We illustrate the framework through the SDLC example introduced in Section 1, with its release-gate interaction depicted in Figure 3(a): Architecture alone returns `NO`, triggering revision; four fresh `OK` verdicts on the corrected artifact then permit deployment.

4.1 PROTOCOL CONSTRUCTION

Protocol Drafting. An LLM drafts the global protocol and a refinement sidecar encoding value-dependent policy; frontier LLMs know Scribble (Yoshida et al., 2013), the decade-old predecessor

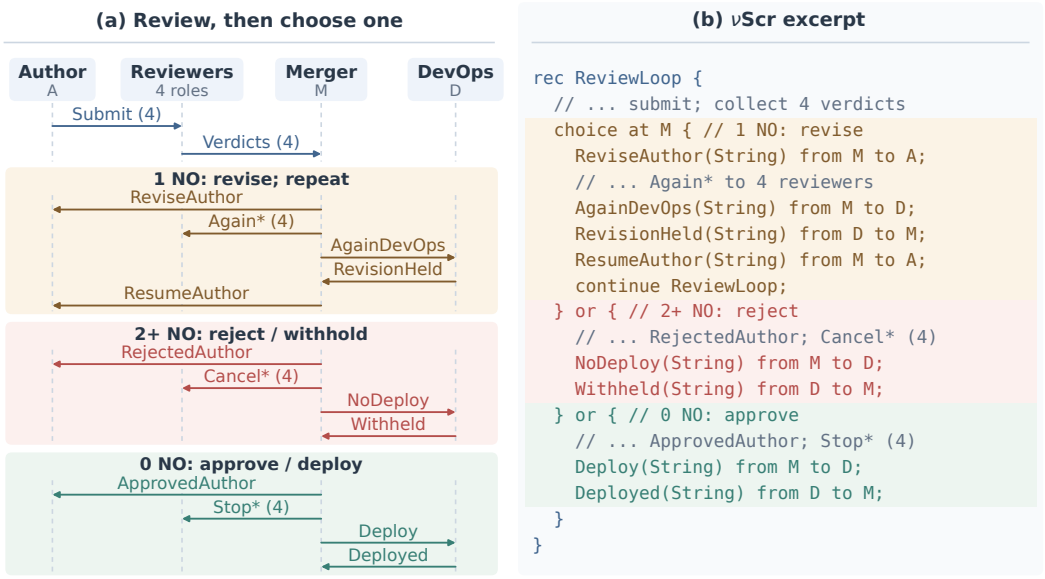

Figure 3: SDLC release-gate interaction and corresponding νSCR representation. (a) The release-gate interaction, with all three branches shown and four independent reviewers grouped for display. (b) An excerpt of the corresponding νSCR protocol; comments abbreviate required messages and declarations are omitted. Verdict-count conditions are encoded in the refinement sidecar.

of νSCR . Figure 3(b) shows an excerpt; a specialized SLM drafter for faster, validity-oriented drafting is separate work (Appendix A).

Static Validation. PROTOGUARD submits each candidate global protocol to νSCR for validation of the structural conditions required for projection, with diagnostics returned for violations. In SDLC, Omitting `AgainDevOps` leaves DevOps uninformed of the revision branch before sending `RevisionHeld`, violating knowledge of choice.

Protocol Refinement. Failed static validation triggers iterative refinement: the drafter incorporates νSCR diagnostics into repairs and resubmits candidates until static validation succeeds. In SDLC, restoring `AgainDevOps` enables the `RevisionHeld` / `ResumeAuthor` handshake before the next review round. The complete νSCR specification of the SDLC example is in Appendix B.

4.2 PROTOCOL-GOVERNED EXECUTION

Figure 4 summarises the contract-derived execution mechanisms in protocol-governed execution.

Contract Projection and Agent Configuration. Once static validation succeeds, νSCR projects the global protocol into a local protocol L_r for each role r. The same L_r determines that role's instructions and runtime contract. In SDLC, SecurityReviewer receives `Submit`, sends `SecurityVerdict`, and awaits Merger's branch notification without access to the other reviewers' contracts.

Runtime Verification and Enforcement. During execution, each role's projected contract is represented as an extended finite-state machine (EFSM). A deterministic monitor checks each proposal's peers, label, and payload against its live state without LLM calls. PROTOGUARD enforces these decisions through a pre-delivery gate that tests a copy of the monitor state, committing state only upon delivery; rejected proposals are blocked and return diagnostics without changing live state. In SDLC, reordered verdicts remain admissible, while approval after only three replies is blocked.

Value-Dependent Enforcement. Structural conformance alone does not capture decisions conditioned on communicated values. A refinement sidecar augments the monitor with payload predicates and stateful choice guards. Guards fail closed on missing evidence and bind verdicts to each review round, preventing stale evidence from enabling a decision. In SDLC, one `NO` permits revision, whereas approval and deployment require four fresh `OK` verdicts.

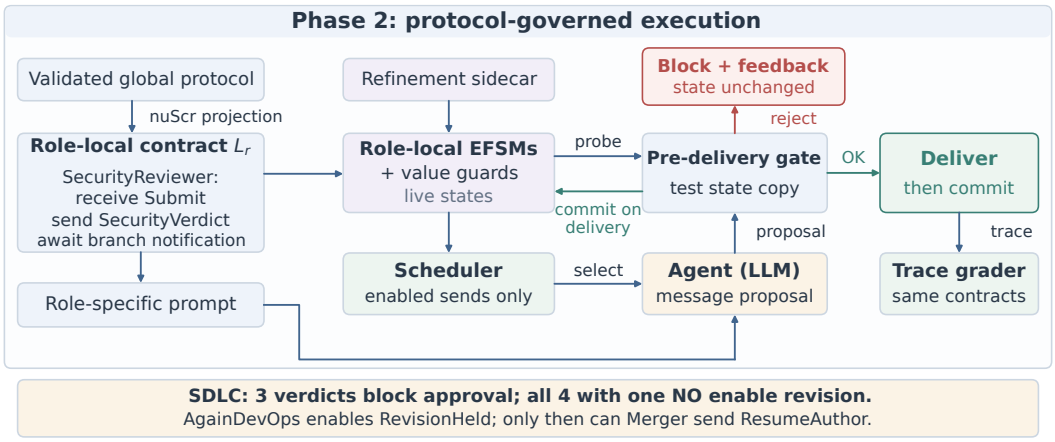

Figure 4: Contract-derived mechanisms for protocol-governed execution. Projected role-local contracts determine agent prompts, EFSM monitors, value guards, and enabled-sender scheduling. The pre-delivery gate commits state only for accepted messages, while conformance grading replays traces against the same contracts. All governance mechanisms are deterministic and require no additional LLM calls.

Protocol-Aware Scheduling. Runtime enforcement constrains message delivery but does not determine which roles are eligible to act. The scheduler invokes only roles with an enabled outgoing transition in the live monitor state, avoiding LLM calls to waiting roles. In SDLC, reviewers become independently eligible after `Submit`; Merger cannot decide before all four verdicts arrive, and DevOps cannot acknowledge a branch before receiving its notification.

Conformance Grading. After execution, PROTOGUARD assesses conformance by replaying the recorded interaction trace against the projected contracts. Replay applies the same structural and value constraints, checking the revision handshake and branch notifications; in SDLC, `Deployed` alone does not establish completion.

5 EXPERIMENTS

We evaluate protocol-governed execution on four multi-agent benchmarks spanning complementary coordination challenges and multiple underlying LLMs. Specifically, we address three research questions: **(RQ1)** To what extent does protocol-governed execution improve coordination correctness and interaction safety across different coordination structures and models? **(RQ2)** To what extent does protocol governance improve token and execution-time efficiency? **(RQ3)** Which runtime mechanisms account for the observed gains?

5.1 EXPERIMENTAL SETUP

Benchmarks. We consider four multi-agent tasks spanning diverse domains and coordination structures (Table 1): AgenticPay (Liu et al., 2026), React 18 (GitHub, 2026a), Finance, and an SDLC release gate. Our AgenticPay-derived task models a four-party transaction among a buyer, seller, carrier, and escrow service, where individually plausible payment and delivery constraints can induce circular waiting and require an escrow-first interaction order. React 18 describes a six-role software-migration workflow involving migration, testing, rework, and sign-off through branching and iterative repair. Finance, motivated by a business use case we encountered in industry, is a six-role financial reporting workflow where revenue classification determines a value-dependent audit path before report generation. SDLC, our running example introduced in Section 1, represents a seven-role release process where four independent reviewers assess an artifact and a Merger aggregates their verdicts to determine deployment (zero objections), revision (one objection), or withholding (two or more objections). The challenge is coordinating concurrent reviews and asynchronous verdict aggregation: Merger must collect all four current-round verdicts before deciding and notify all affected roles.

Table 1: Coordination challenges represented by the four multi-agent benchmarks.

Benchmark	Roles	Coordination challenge	What it stresses
AgenticPay	4	Circular wait / settlement ordering	deadlock avoidance
React 18	6	Rework vs. sign-off	branching + recursion
Finance	6	Approval-dependent reporting	ordering + value dependency
SDLC	7	Concurrent reviews	async verdict aggregation

Execution Evaluation. We assess the three RQs through five progressively governed configurations (C1–C5). C1 (*no protocol*) is the skills-only baseline, relying on role-specific instructions alone; C2 (*global protocol*) provides the validated global protocol as context; C3 (*local projection*) instead supplies each role its local contract; C4 (+ *enforcement*) adds the pre-delivery gate; and C5 (+ *scheduling*), the full PROTOGUARD configuration, additionally enables protocol-aware scheduling. Each successive configuration introduces one mechanism while task inputs, roles, artifacts, and execution limits remain fixed, enabling controlled comparisons between adjacent configurations.

Metrics. We consider goal-completion grading (GCG), goal-completion rate (GCR), protocol violations, catastrophic actions, token efficiency, and execution time. GCG asks whether the specified interaction goals were satisfied, whereas GCR asks whether the complete execution succeeded under the benchmark's criteria. Specifically, Finance and SDLC additionally assess policy application, branch execution, and terminal behaviour; AgenticPay and React 18 use their reported all-goals-pass criteria. For the latter two benchmarks, the same score therefore appears in both rows of Figure 5; it is not a separate, stricter grade. Appendix F details these checks and their limitations. Catastrophic actions are benchmark-specific failure flags, including premature actions and, where specified, stalled completion; they are recorded separately from GCR. Token efficiency divides the total tokens consumed across all trials of a configuration, including trials that failed, by that configuration's number of successful executions, so a configuration's cost reflects tokens spent on both successes and failures. Execution time is the wall-clock duration from trial start to completion or failure. Token and timing measurements cover runtime execution under C1–C5; one-time protocol construction, reused across all trials, is excluded.

Models and Experimental Scale. We aim to test whether protocol support provides consistent benefits across model families and capability tiers, not to compare the latest releases. We therefore select flagship and smaller models from proprietary and open-weight families: GPT-5.4 (OpenAI, 2026), GPT-5-mini (OpenAI, 2025), DeepSeek-V4-Pro (DeepSeek-AI, 2026), and DeepSeek-V4-Flash (DeepSeek-AI, 2026). Each benchmark is evaluated under configurations C1–C5 with 30 repetitions per model–configuration pair. Finance and AgenticPay each comprise 600 trials. SDLC spans three scenarios – approval, withholding, and revision followed by approval – while controlling reviewer-specific evidence rather than scripting reviewer verdicts, yielding 1,800 trials. React 18 follows the same design, with a small number of cells containing fewer than 30 repetitions (Appendix F). Execution failures are retained as unsuccessful trials.

5.2 EFFECTIVENESS OF PROTOCOL-GOVERNED EXECUTION

We first assess protocol-governed execution in terms of coordination correctness and interaction safety (RQ1) and token and execution-time efficiency (RQ2). Throughout Sections 5.2–5.3, we report equal-weight macro-averages across the 16 benchmark–model combinations for each configuration, with timing summaries based on the per-combination medians. These averages retain each benchmark's completion and catastrophe criteria rather than imposing a uniform success or safety predicate. Figures 5–6 present the detailed results; complete numerical data are provided in Appendices D–F.

Coordination Correctness. Protocol-governed execution substantially improves end-to-end coordination. Macro-averaged GCG rises from 5.5% under C1 to 99.0% under C5; macro-averaged GCR rises from 4.1% under C1 to 98.9% under C5. Providing protocol context helps, but enforcement and scheduling yield the highest completion rates. C1's low completion is consistent with the challenge of coordinating cross-role dependencies, evidence-dependent decisions, rework, and consistent termination through role-specific instructions alone, without a validated interaction protocol.

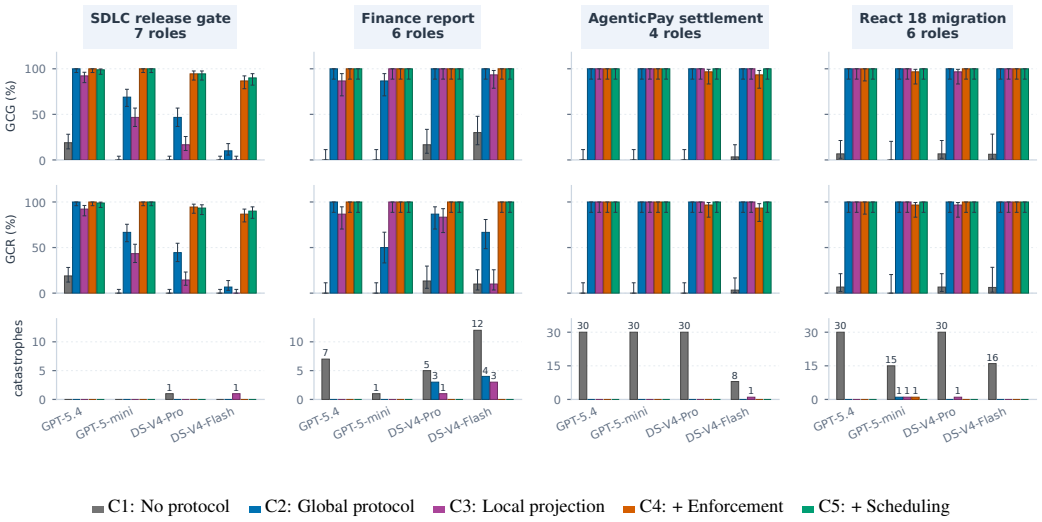

Figure 5: Coordination correctness and interaction safety across benchmarks, models, and configurations C1–C5. Rows report GCG, GCR, and catastrophic actions; rate bars show Wilson 95% confidence intervals. Full results are provided in Appendices D–F.

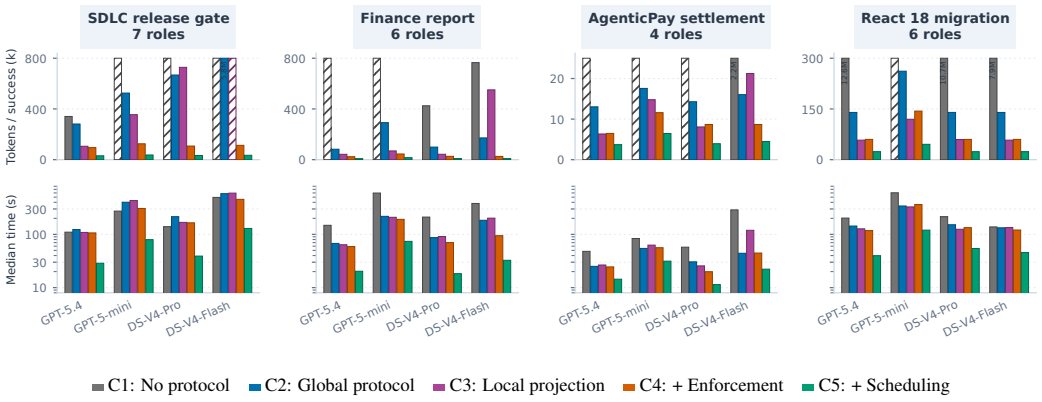

Figure 6: Token and execution-time efficiency under protocol-governed execution across benchmarks, models, and configurations C1–C5. Rows report total tokens (including failed trials) divided by trials passing benchmark-specific GCR checks, and median wall-clock execution time. Hatched token bars indicate zero successes (undefined ratio).

Interaction Safety. Protocol governance prevents invalid communication from reaching recipients. No delivered protocol violations are recorded under C4 or C5, although agents can still propose invalid actions despite receiving local contracts. The macro-averaged catastrophic-trial rates are 50.7%, 1.7%, 1.5%, 0.2%, and 0% under C1–C5, respectively. These use benchmark-specific criteria, including stalled completion where specified, rather than a uniform irreversible-action criterion. Thus, within the evaluated executions, enforcement eliminates delivered protocol violations, while the full configuration records no catastrophic trials.

Efficiency. Protocol-governed execution substantially reduces execution time: the mean of the 16 per-combination median execution times falls from 249.6 s under C1 to 47.6 s under C5, an 80.9% reduction. Scheduling provides an additional token-efficiency gain, reducing mean tokens per success by approximately 66.3% from C4 to C5. We do not average tokens per success for C1 or C3, where some combinations have no successful trials. Pooling all trials instead, tokens per success fall from 3.49M under C1 to 24.0K under C5 (99.3%).

Overall, protocol-governed execution improves coordination correctness, prevents delivered protocol violations, and reduces runtime computational cost on the evaluated benchmarks and models.

5.3 Component Ablation

We then analyse which components of protocol-governed execution account for the observed gains (RQ3). Configurations C1–C5 in Figures 5 and 6 progressively introduce global protocol context, local projection, runtime enforcement, and protocol-aware scheduling, with other conditions fixed.

Global Protocol and Local Projection. Providing a validated protocol as context improves coordination over the no-protocol baseline (C1) but does not ensure conformance. The global-protocol configuration C2 reaches 88.3% macro-averaged GCG and 82.6% GCR; role-local contracts under C3 reach 83.3% GCG and 76.7% GCR. Projection reduces the protocol context exposed to each agent, but does not by itself improve completion over global context or prevent delivered violations. These results distinguish protocol information from enforcement: global and local specifications can guide agent behaviour, while neither guarantees conformance without runtime enforcement.

Runtime Enforcement. Enforcing the projected contracts at delivery substantially improves coordination correctness and interaction safety. Adding the pre-delivery gate from C3 to C4 raises macro-averaged GCG from 83.3% to 98.0% and GCR from 76.7% to 98.0%, with no recorded delivered protocol violations. The benchmark-specific catastrophic-trial rate falls from 1.5% to 0.2%, but is not yet zero. Contract availability therefore does not imply conformance: providing agents with a validated contract guides interaction, whereas pre-delivery enforcement prevents structurally or value-inconsistent actions from affecting other roles.

Protocol-Aware Scheduling. Scheduling provides a further efficiency gain by restricting invocation to roles with enabled outgoing transitions. Whereas C4 gates outputs without eligibility-based role selection, C5 additionally avoids invoking ineligible roles, reducing opportunities for idle or premature actions. Macro-averaged GCG rises from 98.0% to 99.0% and GCR from 98.0% to 98.9%, with zero delivered violations in both configurations; the mean of per-combination median execution times falls from 145.6 s to 47.6 s (a 67.3% reduction). Scheduling thus complements enforcement by suppressing idle and premature invocations.

The ablation reveals distinct roles for the runtime mechanisms. Global and local protocol specifications provide agents with structured interaction context, with local projection reducing role-irrelevant context and token overhead. Enforcement makes the projected contracts binding at communication boundaries, while scheduling suppresses unnecessary and premature invocations without compromising coordination correctness or interaction safety. The observed gains therefore arise from the composition of these mechanisms rather than from protocol prompting alone.

6 Conclusion, Limitations, and Future Work

We presented ProtoGuard, a framework for protocol-governed coordination of LLM agents that connects static protocol validation with deterministic runtime governance. LLM-drafted global protocols are validated against MPST requirements and projected into role-local contracts for agent prompting, monitoring, pre-delivery enforcement, scheduling, and conformance grading. Across the evaluated benchmarks, protocol-governed execution substantially improves coordination correctness, prevents delivered protocol violations, and reduces runtime token and execution-time costs. These results establish formal interaction specifications as a foundation for governing stochastic multi-agent execution through validated and enforceable coordination protocols.

Several limitations bound these findings. Protocol conformance guarantees adherence to the specified interaction structure, but not the semantic correctness or factual accuracy of message contents. Static validation likewise does not establish that a protocol faithfully captures the intended task. The runtime evaluation takes validated protocols as given; translating intents into faithful protocols is separate work, with a preliminary study in Appendix A.

Future work includes extending formal protocol governance along two dimensions. First, richer protocol models incorporating concurrency, dynamic participation, recovery, and interaction constraints can broaden applicability to larger and more heterogeneous agent systems. Second, direct linkage between accepted protocols and downstream runtime campaigns will enable evaluation of protocol-governed execution across increasing team sizes, interaction complexity, and model capabilities. More broadly, these directions advance formal specifications from coordination artifacts toward a reusable execution substrate for reliable multi-agent LLM systems.

AI USE STATEMENT

Generative AI tools, including GitHub Copilot, assisted with manuscript drafting and editing, literature and source discovery, reference formatting, and LaTeX preparation. Generative AI was also used to draft natural-language benchmark material. Within the research workflow, LLMs draft and refine global protocols and support the exploratory semantic assessments reported separately in Appendix A; the evaluated agent teams are themselves LLM-based, as specified in Section 5.1. These uses are distinct from the deterministic static validation and runtime verification mechanisms. The authors have reviewed the AI-assisted work and take responsibility for the final submission, including its text, claims, references, and research artifacts.

ETHICS STATEMENT

This work studies protocol-governed interaction in controlled multi-agent benchmarks. Protocol conformance is not a guarantee of factual accuracy, unbiased decisions, security, or suitability for deployment in financial or software-release systems. A formally valid protocol can still encode an inappropriate policy, and a conforming agent can produce incorrect or harmful content. Real-world use therefore requires human oversight, task-specific risk assessment, appropriate access controls, and independent evaluation of message contents and downstream actions. Use and redistribution of third-party agent definitions and benchmark materials must respect their licenses and attribution requirements; shared prompts and traces should exclude credentials, personal data, and confidential information.

REPRODUCIBILITY STATEMENT

Section 4 describes protocol construction and protocol-governed execution, and Section 5.1 specifies the benchmarks, execution configurations, models, metrics, and trial design. Appendix A details the separate drafter's training and evaluation procedure; Appendix B provides the SDLC protocol; and Appendix C documents the coordination-defect examples and their checks. Appendices D–F report results by model, scenario, and benchmark. Recomputing metrics from recorded traces is distinct from rerunning the LLM experiments: model stochasticity, API availability, provider updates, and service load can affect fresh outputs and timings. The trained-drafter pilot and runtime evaluations use distinct artifact lineages, as discussed in Appendix A. The anonymized supplementary material provides the code, protocols, and per-trial execution traces for all four benchmarks, with a script that recomputes the trace-derived table entries.

REFERENCES

Amazon Web Services. AI-DLC workflows: AI-driven development life cycle. `https://github.com/awslabs/aidlc-workflows`, 2026.

Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR)*, pp. 162–176, 2010.

Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. In *Formal Techniques for Distributed Systems (FMOODS/FORTE)*, pp. 50–65, 2013.

Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theoretical Computer Science*, 669:33–58, 2017.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL `https://neurips.cc/virtual/2025/poster/121528`.

Tzu-Chun Chen and Kohei Honda. Specifying stateful asynchronous properties for distributed programs. In *Proceedings of the 23rd International Conference on Concurrency Theory (CONCUR)*, pp. 209–224, 2012.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

GitHub. awesome-copilot: React 18 migration agents. https://github.com/github/awesome-copilot, 2026a. react18-upgrade plugin, added April 9, 2026.

GitHub. Awesome GitHub Copilot: SDLC agent personas. https://github.com/github/awesome-copilot/tree/318066d2213b510e89b500ed0d53506c54093ddc, 2026b. Revision 318066d2213b510e89b500ed0d53506c54093ddc, August 17.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for LLM agents. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OhUoTMxFIH.

Google Cloud. Workflows overview. https://docs.cloud.google.com/workflows/docs/overview, 2026. Accessed September 22, 2026.

Florian Grötschla, Luis Müller, Jan Tönshoff, Mikhail Galkin, and Bryan Perozzi. Agentsnet: Coordination and collaborative reasoning in multi-agent llms. *CoRR*, abs/2507.08616, 2025. doi: 10.48550/ARXIV.2507.08616. URL https://doi.org/10.48550/arXiv.2507.08616.

Najmul Hasan and Prashanth BusiReddyGari. Dpbench: Structural determinants of multi-agent llm coordination under simultaneous resource contention, 2026. URL https://arxiv.org/abs/2602.13255.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pp. 273–284, 2008.

Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1):9:1–9:67, 2016.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Atsushi Igarashi, Peter Thiemann, Vasco T. Vasconcelos, and Philip Wadler. Gradual session types. *Proceedings of the ACM on Programming Languages (ICFP)*, 1:38:1–38:28, 2017.

Adharsh Kamath, Sishen Zhang, Calvin Xu, Shubham Ugare, Gagandeep Singh, and Sasa Misailovic. Enforcing temporal constraints for LLM agents. *CoRR*, abs/2512.23738, 2025. doi: 10.48550/ARXIV.2512.23738. URL https://doi.org/10.48550/arXiv.2512.23738.

Maurits Kaptein, Vassilis-Javed Khan, and Andriy Podstavnychy. Runtime governance for ai agents: Policies on paths, 2026. URL https://arxiv.org/abs/2603.16586.

LangChain, Inc. LangGraph: Balancing agent control with agency. https://www.langchain.com/langgraph, 2024.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. ST-WebAgentBench: A benchmark for evaluating safety and trustworthiness in web agents. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://iclr.cc/virtual/2026/poster/10009904.

Ying Li, Yanju Chen, Hongbo Wen, Bosi Zhang, Hanzhi Liu, Peiran Wang, Yu Feng, and Yuan Tian. VIGIL: runtime enforcement of behavioral specifications in AI agent skills. *CoRR*, abs/2606.26524, 2026. doi: 10.48550/ARXIV.2606.26524. URL https://doi.org/10.48550/arXiv.2606.26524.

Xianyang Liu, Shangding Gu, and Dawn Song. AgenticPay: A multi-agent LLM negotiation system for buyer-seller transactions. *arXiv preprint arXiv:2602.06008*, 2026. URL https://arxiv.org/abs/2602.06008.

Microsoft. Microsoft agent framework: Workflows. https://learn.microsoft.com/en-us/agent-framework/workflows/, 2025.

Lesly Miculicich, Mihir Parmar, Hamid Palangi, Krishnamurthy Dj Dvijotham, Mirko Montanari, Tomas Pfister, and Long T. Le. Veriguard: Enhancing LLM agent safety via verified code generation. *CoRR*, abs/2510.05156, 2025. doi: 10.48550/ARXIV.2510.05156. URL https://doi.org/10.48550/arXiv.2510.05156.

OpenAI. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/, 2025. Accessed: 2026-09.

OpenAI. Introducing GPT-5.4. https://openai.com/index/introducing-gpt-5-4/, March 2026. Accessed: 2026-09.

Alceste Scalas and Nobuko Yoshida. Less is more: Multiparty session types revisited. In *Proceedings of the ACM on Programming Languages (POPL)*, volume 3, pp. 30:1–30:29, 2019.

Rui Sun, Zhi Zheng, Zhenkun Wang, and Zhichao Lu. GraphSkillEvo: Evolutionary optimization of graph-structured agent skills. *arXiv preprint arXiv:2609.21749*, 2026. URL https://arxiv.org/abs/2609.21749.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th International Conference on Software Engineering (ICSE)*, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Shuren Xia, Qiwei Li, Taqiya Ehsan, and Jorge Ortiz. TraceFix: Repairing agent coordination protocols with TLA+ counterexamples. In *Proceedings of the ACM Conference on AI and Agentic Systems (CAIS)*, 2026. doi: 10.1145/3786335.3813159.

Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The Scribble protocol language. In *Trustworthy Global Computing (TGC)*, pp. 22–41, 2013.

Nobuko Yoshida, Fangyi Zhou, and Francisco Ferreira. Communicating finite state machines and an extensible toolchain for multiparty session types. In Evripidis Bampis and Aris Pagourtzis (eds.), *Fundamentals of Computation Theory - 23rd International Symposium, FCT 2021, Athens, Greece, September 12-15, 2021, Proceedings*, volume 12867 of *Lecture Notes in Computer Science*, pp. 18–35. Springer, 2021. doi: 10.1007/978-3-030-86593-1_2. URL https://doi.org/10.1007/978-3-030-86593-1_2.

Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, and Wenyuan Jiang. SILO-BENCH: A scalable environment for evaluating distributed coordination in multi-agent LLM systems. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29379–29398, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1354. URL https://aclanthology.org/2026.acl-long.1354/.

The appendices provide detailed results and the artifact documents trace-based metric reproduction.

A PRELIMINARY INTENT-TO-νSCR SLM STUDY

As a preliminary study separate from PROTOGUARD's core contribution, we explore a small language model (SLM) for intent-to-νSCR drafting. The aim is to reduce drafting overhead without replacing PROTOGUARD's static validation or protocol-governed execution. Drafter training and intent faithfulness are a separate research direction, not prerequisites for the runtime mechanisms evaluated in the main text.

A.1 TRAINING METHOD

We fine-tune Qwen2.5-Coder-7B-Instruct with LoRA (rank 32, scaling factor 64, dropout 0.05), keeping the base weights frozen. Intent–νSCR pairs are qualified by static validation and faithfulness assessment before completion-only next-token training. We use BF16, learning rate 5×10^{-5}, batch size 2 with eight accumulation steps, and a 2,048-token limit.

Ten development intents, role-name variants of one synthetic two-worker-join workflow, guide checkpoint selection but not gradient updates. Each permits one greedy draft and up to five compiler-guided repairs, stopping at the first static pass; faithfulness is assessed afterwards.

A.2 DEVELOPMENT RESULTS

Table 2: Drafting results on ten development intents from one workflow family.

Model / revision	**Mean F1**	**Total tokens**
V0 — unadapted base	–	19,620
V1 — previous checkpoint, fresh-loaded	92.86	9,357
V7 — selected	**93.59**	**2,206**

F1 covers six statically valid V1 drafts and all ten V7 drafts; "–" denotes unavailable F1. Tokens include input, output, and failed attempts, but exclude data qualification and faithfulness assessment.

V7 passes both checks on all ten first drafts, with mean F1 93.59 and no repairs. It uses 2,206 native tokens, about 221 per intent: 88.8% fewer than the unadapted base and 76.4% fewer than V1, including their failed attempts. These are observed token reductions, not equal-success cost comparisons. Fresh-loading confirmation reproduces the F1 and token totals on the same cases.

On one H100, the selected training lineage records 56.2 s of Trainer runtime; generating the ten V7 drafts takes 20.4 s in comparison and 36.3 s in confirmation. These timings exclude model loading and subsequent validation and assessment; the training figure also excludes other development runs. The modest token and compute requirements suggest potential for low-cost drafting, not measured end-to-end cost or latency savings.

This remains an unfinished pilot, not a completed evaluation: the development cases guided selection, and generalisation to unseen workflows is unestablished. High development F1 with short, repair-free drafts is a promising starting point for broader held-out evaluation. The pilot and runtime evaluations use distinct artifact lineages; its results do not establish the semantic adequacy of the exact protocols used in the runtime experiments.

A.3 STATIC VALIDATION AND FAITHFULNESS ASSESSMENT

νSCR checks well-formedness, role consistency, and projection. A separate GPT-5.4 assessor checks required behaviours and source-compatible commitments against the intent, citing evidence. Faithfulness F1 is the harmonic mean of requirement recall and commitment precision; it is an LLM-assisted assessment, not a formal guarantee.

Static validity alone does not establish correspondence with the intent. We therefore measure requirement recall R to detect omitted behaviours and commitment precision P to detect unsupported or incompatible additions; their harmonic mean F1 penalises imbalance. These are evaluation met-

rics, not the training loss. For N source requirements and M draft commitments:

$$R = 100\frac{\sum_{i=1}^{N} r_i}{N}, \qquad P = 100\frac{\sum_{j=1}^{M} \min(g_j, c_j)}{M}, \qquad \text{F1} = \frac{2PR}{P+R}, \tag{1}$$

where r_i measures requirement coverage, and g_j and c_j measure a commitment's grounding and compatibility. Credits are 1, 0.5, or 0 for full, partial, or no satisfaction; taking $\min(g_j, c_j)$ requires both grounding and compatibility. Scores lie on a 0–100 scale; F1 is zero when $P = R = 0$. Unknown required assessments leave the corresponding aggregate unavailable rather than assigning zero. Table 2 reports mean per-draft F1 over the stated scored drafts.

A draft is accepted only if it is statically valid, covers every required obligation with compatible commitments, passes the behaviour witnesses, and has no blocking semantic fault.

B THE SEVEN-ROLE RELEASE-GATE PROTOCOL

The global protocol, accepted by νSCR, reads as follows. νSCR projects it to 25 states and 26 transitions for the Merger.

```
rec ReviewLoop {
  Author -> {Quality, Security, Arch, RAI}: Submit(artifact)
  {Quality, Security, Arch, RAI} -> Merger: Verdict(text)   // any arrival order
  choice at Merger {
    // exactly one NO: revise
    Merger -> Author: ReviseAuthor;  Merger -> reviewers: Again;
    Merger -> DevOps: AgainDevOps;   DevOps -> Merger: RevisionHeld;
    Merger -> Author: ResumeAuthor;  continue ReviewLoop
  } or {
    // two or more NO: reject and withhold
    Merger -> Author: RejectedAuthor; Merger -> reviewers: Cancel;
    Merger -> DevOps: NoDeploy;       DevOps -> Merger: Withheld   // terminal
  } or {
    // zero NO: approve and deploy
    Merger -> Author: ApprovedAuthor; Merger -> reviewers: Stop;
    Merger -> DevOps: Deploy;         DevOps -> Merger: Deployed   // terminal
  }
}
```

C SOURCES AND CHECKS FOR FIGURE 1

The four panels are deliberately reduced examples, not additional benchmark runs. Panel (a) instantiates a two-blocker SDLC hazard: without an enforced receive barrier, the Merger can treat the earlier `OK`s as complete and choose `Deploy` before the Security and Architecture verdicts arrive. Panel (b) comes from the exclusive-choice/AND-join example in the React 18 case notes. Panel (c) isolates the approval-before-reporting obligation in the finance protocol `v1.scr` and its sequence policies S1–S2 in `v1.policy`: approval precedes final analysis, which precedes report generation. Panel (d) comes from the four-role circular wait in the AgenticPay settlement case notes; the repair follows that case's escrow-first `v1.scr`. This is a second cause of an impossible wait, not a fourth interaction-error category. The unchecked settlement benchmark used round-robin polling, not a MAF workflow reproduction.

Minimal, payload-free reproductions are provided beside the manuscript in `figure_protocols`. We checked the protocols with νSCR. For (a), the projected Merger contract cannot reach its choice before receiving all four current-round verdicts, and the value guard permits `Deploy` only when their `NO` count is zero. For (b), static validation rejects the bad protocol, while the repair requires only the selected reply and explicitly notifies the unselected role to terminate. Both finance orderings are structurally accepted, demonstrating that static session typing alone does not infer the approval policy.

Table 3: Checks for Figure 1: projected-contract enforcement in (a) and (c), static validation diagnostics in (b), and cyclic local-order constraints in (d).

	Type / source	Runtime failure	Detection and its scope
(a)	Premature choice / SDLC	Merger chooses `Deploy` from early `OK`s while two blocker verdicts remain in flight	The Merger contract requires four receives before the choice; the value guard then forbids `Deploy` unless their `NO` count is zero.
(b)	Impossible wait / React 18	One branch runs; the join waits for both replies	The checker rejects the choice-plus-both-replies protocol: *wait-for cycles* involving Commander and the unselected role.
(c)	Premature action / Finance	Analyst sends final analysis before receiving approval	The correct protocol projects to receive-`Approval` before send-`FinalRevenueAnalysis`; the runtime gate disallows an early send.
(d)	Circular wait / AgenticPay	All four agents wait before sending any message	The four local receive-before-send constraints form a cycle with no initial action; no global ordering can preserve them all. Funding Escrow first breaks the cycle.

For (c), the projected RevenueAnalyst endpoint is:

```
SEND StandardRevenueNotification TO TaxVerifier;
RECEIVE Approval FROM TaxVerifier;
SEND FinalRevenueAnalysis TO Writer;
```

After the first send, the only enabled transition is receipt of `Approval`; `FinalRevenueAnalysis` is not enabled until that receipt occurs. This is the precise runtime check. Conversely, deleting an obligation from a protocol, or deliberately specifying an incorrect business ordering, can leave a formally valid but unfaithful protocol; neither the figure nor the checker results claim otherwise.

For (d), the faulty local rules impose the strict ordering

$$\texttt{DeliverGoods} \prec \texttt{FundEscrow} \prec \texttt{FundsSecured} \prec \texttt{ShipGoods} \prec \texttt{DeliverGoods}.$$

The script `figure_protocols/d_check.py` reproduces this cycle and verifies that no initial message is enabled; reversing Buyer's funding/delivery dependency yields the escrow-first order. This is a local-order consistency check, not a reproduced static validation rejection. Merely listing the four messages in a linear νSCR protocol would drop at least one of these dependencies and would not faithfully encode the deadlock. In the repaired settlement protocol, `FundEscrow` precedes delivery, but `ReleasePayment` still follows `ConfirmReceipt`: depositing money with Escrow is distinct from paying Seller.

D PER-MODEL SDLC RESULTS

All execution-result tables use configurations C1–C5 defined in Section 5.1. Their artifact identifiers are C1: `skills`, C2: `globalvalid`, C3: `localvalid`, C4: `localvalid_gate`, and C5: `localvalid_sched`. These execution configurations are distinct from the trained-drafter pilot in Appendix A and the SDLC scenario identifiers S1, S2, and S4. GCG, GCR, success, and Clean are reported as percentages, rounded to one decimal place where needed; CAT, Delivered, and Blocked remain counts. Pale shading identifies the full configuration C5, not a per-column optimum.

Table 4: SDLC release gate over 1,800 trials (360 per configuration). Delivered counts rule-breaking messages that reached their receivers. Blocked counts rejected actions, including 21 and 9 rejected WAIT responses under C4 and C5, respectively; it is not exclusively a count of intercepted messages. *C1 runs no monitor, so its zero is only what was observed.

Config.	GCR (n=360)	CAT	Delivered	Blocked	Tokens/success	$/success
C1	4.7%	1	0*	—	1,410,213	2.069
C2	54.4%	0	3,536	—	605,097	0.286
C3	37.5%	1	7,901	—	302,226	0.379
C4	95.3%	0	**0**	3,908	110,075	0.143
C5	**95.6%**	0	**0**	29	**33,705**	**0.036**

Table 5: GCR and average tokens per trial by model (SDLC, n=90 per cell). The only C1 successes are GPT-5.4 on the approve scenario.

	GPT-5.4		GPT-5-mini		DS-V4-Pro		DS-V4-Flash	
Config.	GCR	tok/run	GCR	tok/run	GCR	tok/run	GCR	tok/run
C1	18.9%	64,377	0%	66,734	0%	59,262	0%	75,999
C2	100%	281,301	66.7%	350,204	44.4%	296,714	6.7%	389,547
C3	92.2%	97,556	43.3%	153,765	14.4%	105,182	0%	96,837
C4	100%	95,054	100%	125,181	94.4%	101,371	86.7%	97,904
C5	98.9%	30,211	100%	36,963	93.3%	30,708	90%	30,945

E PER-SCENARIO SDLC RESULTS

Table 6: Configurations C1–C5 for each SDLC scenario (four models pooled, n=120 per cell). S1 takes the approve branch, S2 the reject-and-withhold branch, and S4 the revision loop followed by approval. S4 is the scenario with the deployment hazard, and both catastrophes in the corpus are in C1–C3.

	S1 (approve)			S2 (reject)			S4 (revise, then approve)		
Config.	GCG	GCR	CAT	GCG	GCR	CAT	GCG	GCR	CAT
C1	14.2%	14.2%	0	0%	0%	0	0%	0%	1
C2	68.3%	67.5%	0	55.8%	53.3%	0	45%	42.5%	0
C3	55%	55%	0	31.7%	29.2%	0	30%	28.3%	1
C4	87.5%	87.5%	0	99.2%	99.2%	0	99.2%	99.2%	0
C5	87.5%	87.5%	0	100%	100%	0	100%	99.2%	0

F FINANCE, SETTLEMENT, AND REACT 18 RESULTS

Benchmark-specific GCR. We use GCR for the fraction of trials passing each benchmark's completion criterion, not for an identical predicate across tasks. Finance and SDLC retain their distinct GCG and GCR grades. For AgenticPay, GCR reuses the final-selected all-goals-pass grade, called GCG in the benchmark report; first-attempt GCG remains a separate retry diagnostic. React 18 likewise contributes its reported all-goals-pass success rate. This reporting convention does not add checks to the original graders, and completion does not imply protocol conformance, which is measured separately.

AgenticPay's goals check positive funding and payment amounts and minimal string payloads for the remaining milestones; React 18's goal predicates are `True`, so they do not establish that tests or a build passed. The shared goal matcher checks goals independently, not their cross-goal order; ordering violations are assessed separately. AgenticPay's final completion rates may include up to three attempts per trial, while the first-attempt column excludes retries.

Table 7: Finance report, 600 trials (120 per configuration, four models pooled). Clean is the percentage of traces with no delivered rule-breaking message. Blocked counts proposals the gate refused.

Config.	GCG	GCR	CAT	Clean	Delivered	Blocked	Tokens/trial
C1	11.7%	5.8%	25	—	—	—	89,291
C2	96.7%	75.8%	7	38.3%	232	—	107,208
C3	95%	70%	4	36.7%	609	—	49,078
C4	100%	100%	0	100%	0	203	30,082
C5	100%	100%	0	100%	0	2	10,237

Table 8: AgenticPay settlement, 600 trials (120 per configuration, four models pooled). The runner permits up to three attempts per trial in every configuration. The first-attempt column uses the first attempt only. CAT counts trials in which any attempt froze with no delivered message, shipped before funds were secured, released payment before receipt was confirmed, or declared completion before delivery or payment; of the 98 C1 catastrophes, 93 froze and 6 took an unsafe step. The C1 rule-breaking count is for diagnosis, since that configuration never received the protocol vocabulary.

Config.	GCR (final)	GCG (first)	CAT	Delivered	Blocked	Total tokens	Tokens/ success
C1	0.8%	0%	98	237	—	4,159,780	4,159,780
C2	100%	96.7%	0	5	—	1,826,661	15,222
C3	100%	87.5%	1	195	—	1,512,398	12,603
C4	97.5%	91.7%	0	0	67	1,035,832	8,853
C5	100%	95%	0	0	0	556,618	4,638

Table 9: React 18 migration: GCR under the benchmark's all-goals-pass criterion, catastrophe count, and average tokens per trial, by model. Trial counts are 30 per cell, except C1 with GPT-5-mini (n=15) and DeepSeek-V4-Flash (n=16), and C4 with GPT-5.4 (n=25). Every delivered C1 message was off-protocol on every model. CAT counts a trial that declares completion without both a prior test result and a prior sign-off, declares completion more than once, or never reaches completion (see Section 5.2).

	GPT-5.4			GPT-5-mini			DS-V4-Pro			DS-V4-Flash		
Config.	GCR	CAT	tok/run	GCR	CAT	tok/run	GCR	CAT	tok/run	GCR	CAT	tok/run
C1	6.7%	30	837,273	0%	15	765,893	6.7%	30	714,935	6.3%	16	492,927
C2	100%	0	139,719	100%	1	261,679	100%	0	139,754	100%	0	139,680
C3	100%	0	57,738	100%	1	119,549	96.7%	1	57,615	100%	0	57,754
C4	100%	0	59,988	96.7%	1	138,869	100%	0	59,897	100%	0	59,955
C5	100%	0	23,700	100%	0	45,409	100%	0	23,633	100%	0	23,728