RELIC: FROM MULTI-AGENT COLLABORATION TO PERSISTENT ORGANIZATIONAL CAPABILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multiple agents may often conflict in an organization: for example, one coding agent changes an interface in a repository, but another continues to develop on the old version where existing tests become stale. A conversation can resolve the episode, but when the participants change, what makes the lesson continue to govern the team? We introduce RELIC, which turns recurring collaboration failures into organization-owned, executable protocols. Members reflect on visible work, propose rules, and govern their adoption. Adopted protocols bind triggers, responsibilities, required evidence, and execution consequences to the runtime, while remaining open to revision and retirement. In one traced case, repeated integration friction produces an interface-review rule that governs later pull requests and is revised as work continues. Across 360 controlled runs over ten software workloads and three models, Relic raises complete-contract delivery from 14.06% to 19.76% (+5.71 percentage points) over a matched structured team without the protocol lifecycle, improving all four verified production endpoints in every model stratum. Under fresh-member transfer, behavioral correctness is 25.4% with no inherited protocol, 34.6% with the same rules provided as readable text, and 41.2% with executable bindings, a +6.5-point advantage over text alone. On the full CooperBench benchmark, after excluding broken benchmark pairs, Relic achieves 367/477 (76.9%), establishing the best reported result among peer-structured systems. On the fixed 48-pair same-model subset, Relic also exceeds Solo (29/48 vs. 26/48), reversing the coordination loss exhibited by the official peer baseline. Together, these results show how collaboration experience can become persistent organizational state that remains useful beyond the members who created it.

1 INTRODUCTION

Consider a developer assigning several coding agents to one repository. Agent A changes a shared interface, agent B updates a client using the old interface, and a third agent runs tests. Each makes progress, yet integration can break the client. Branches may lag hundreds of commits behind the integrated main branch (*mainline*); overlapping edits can overwrite another agent's work, and earlier tests may concern outdated code. Prime Intellect reports a related incident: GPT-5.6 Sol observed unexpected shared-code changes, suspected subagent interference, and gave research advisors read-only instructions (Bakouch & Prime Intellect, 2026). Shared work needs rules specifying who may modify which artifacts, when to synchronize, and what evidence integration requires (Figure 1).

Can communication solve this problem? ChatDev, MetaGPT, and AutoGen support role-based dialogue and workflows (Qian et al., 2023; Hong et al., 2023; Wu et al., 2023). Suppose A and B agree to synchronize mainline, rerun tests, and obtain review before merging. Even with perfect compliance, replacing B with C does not automatically establish the agreement for C. Anthropic reports that fresh coding-agent sessions lack prior memory; undocumented partial work leaves successors guessing, while structured handoffs add orchestration overhead (Young, 2025; Rajasekaran, 2026). Collaborative memory preserves experience (Zhang et al., 2025); the question remains how an agreement becomes an obligation for whoever occupies the role.

Could reusable skills preserve the agreement? Shared skill libraries retain procedures across sessions, including executable code, as in Voyager (Wang et al., 2023). A testing skill can run a suite; its availability alone does not assign an independent reviewer or require current evidence before integration. The additional requirement is to bind learned rules to shared work: specify when

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.

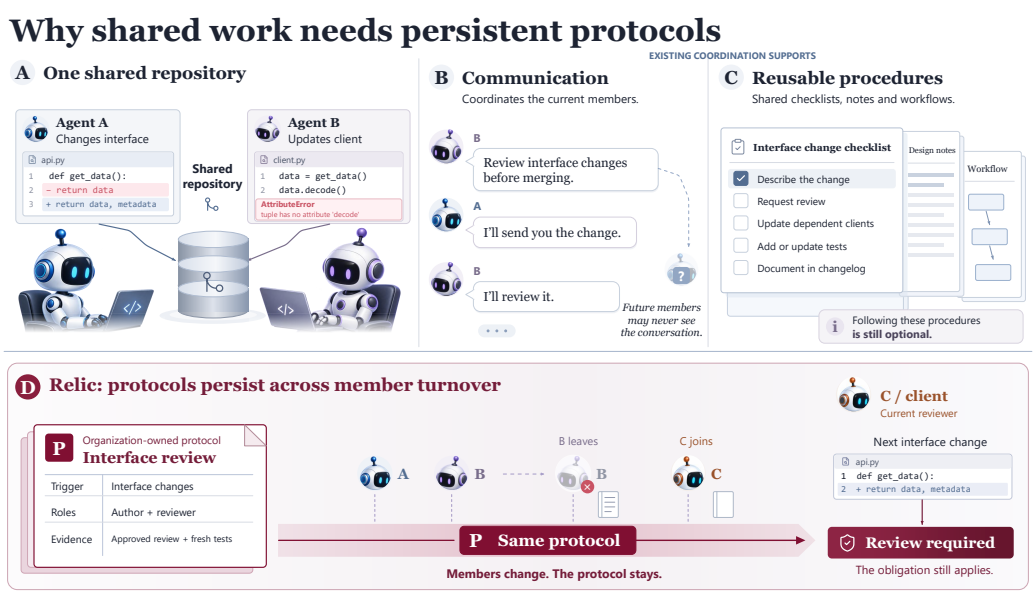

Figure 1: **Why shared work needs organizational protocols.** An illustrative interface change breaks a client developed against the old interface. Communication establishes agreements; memory and skills preserve knowledge and procedures. Relic retains an organization-owned, runtime-bound review protocol: agent B is replaced by agent C, while agent A and shared work continue and the review obligation still applies.

they apply, who is responsible, and how compliance and revisions are governed. Skills can help members fulfill these obligations. Long-horizon collaboration thus produces both software and ways of working: retaining the latter lets an organization apply its experience to future work.

We introduce RELIC, which turns collaboration experience into **organizational capability**: learned ways of working represented by governed, executable protocols outside member-local state. Members propose rules from visible work friction and submit them for validation and approval. **Runtime binding** connects adopted rules to the code that selects actions, assigns responsibilities, and checks shared work. The rules remain revisable and can govern fresh members without their authors' private histories (Section 3).

Contributions. First, we formulate organizational capability as learned, governed, executable protocol state outside member-local memory, allowing organization-owned rules to persist across member turnover. Second, across 360 controlled runs spanning three models, Relic improves pooled complete contracts by 5.71 pp over the matched structured team, with gains on all four verified endpoints in every model stratum. Across the Terra+Opus 60-run Relic census, 280 autonomous protocol lineages enter sustained use. Third, two controls isolate executability: a prose-only ablation retains online rule proposal and governance but loses 7.18 pp on complete contracts; content-matched transfer to fresh members gains 6.5 pp from executable binding over prose alone. Finally, external tests show portability: ProgramBench improves mean behavioral correctness by 6.752 pp, while on the full CooperBench benchmark Relic achieves 367/477 (76.9%) after excluding broken benchmark pairs, establishing the best reported peer-structured result. On the fixed same-model 48-pair subset, Relic also exceeds Solo (29/48 vs. 26/48), reversing the coordination loss observed for the official peer system.

2 RELATED WORK

Communication and shared workflows. CAMEL, ChatDev, and AutoGen coordinate specialized agents through dialogue and delegation (Li et al., 2023; Qian et al., 2023; Wu et al., 2023); MetaGPT incorporates human-designed operating procedures into agent workflows (Hong et al., 2023). MultiAgentBench evaluates collaboration, ProtocolBench compares communication protocols, and CooperBench examines coordination over interacting code changes (Zhu et al., 2025; Du et al.,

2026; Khatua et al., 2026). Relic asks how experience from collaboration becomes governed rules for subsequent work, extending the focus from coordinating actions to learning how collective work should proceed.

Memory and reusable skills. Reflexion and ExpeL retain feedback and extracted experience for later reasoning (Shinn et al., 2023; Zhao et al., 2023). G-Memory retrieves collaboration trajectories and cross-trial insights (Zhang et al., 2025), while Voyager accumulates executable skills (Wang et al., 2023). These mechanisms preserve knowledge and procedures. Relic focuses on cross-member obligations and their runtime binding: a skill supplies a testing procedure, while a protocol assigns responsibility for producing and reviewing the evidence required for shared work.

Self-evolving agent organizations. Meta-Team learns improvements to agent behavior, coordination, and team organization (Hao et al., 2026). OneManCompany builds an organizational layer around portable agent identities and dynamic recruitment (Yu et al., 2026); TheBotCompany adapts teams during continuous software development (Lyu et al., 2026). Relic focuses on the learned protocol itself: its evidence-grounded proposal, governed adoption, execution, and revision. Rule content evolves while model parameters and the decision and execution substrate remain fixed. Fresh-member transfer and text-only controls test whether retained protocols remain useful and whether runtime binding adds value beyond readable rules.

Organizational routines and governance. Routines and dynamic capabilities explain how collectives retain and reconfigure ways of working (Nelson & Winter, 1982; Feldman & Pentland, 2003; Teece et al., 1997). Computer-supported cooperative work (CSCW) examines coordination of interdependent tasks (articulation work) and artifacts shared across groups (boundary objects) (Schmidt & Bannon, 1992; Ackerman, 2000; Lee, 2007). Relic makes learned working rules explicit, revisable organizational objects outside member-local state. Its evaluation separately measures whether they enter sustained practice, improve verified outcomes, and remain useful after member replacement.

3 Relic: From Experience to Organizational Protocols

Relic adds a governed protocol lifecycle to long-horizon collaboration. Members propose rules from observed problems; validation and approval precede runtime binding, revision, or retirement. Figure 2 follows one interface-review protocol from its origin to execution and new-member reuse.

3.1 What an Organizational Protocol Contains

Relic separates **member-local state** (private memory, private workspaces, and experience), **shared work state** (code, documents, tasks, and messages), and **protocol state** (learned rules and role mappings). Shared work can outlast a member; protocols carry obligations for later occupants of a role.

A protocol specifies its trigger and scope, responsible roles, required steps and evidence, affected actions or artifacts, execution consequences, and revision or retirement state. The example in Figure 2 requires an author and reviewer to check interface changes against a written interface specification and current test evidence. Its response combines review priority and readiness checks. Readable text explains the obligations; runtime bindings apply them.

Members have partial views: only accessible information they read or retrieve enters their context. Private workspaces and explicit sharing determine how information reaches each member. Evaluator-side audit records support measurement and replay separately from member context (Appendix C.1).

3.2 From Work Friction to Governed Protocols

Repeated interface failures can prompt reflection and a proposal naming the problem, visible evidence, affected work, and responsible roles. Validation checks evidence grounding and compatibility with supported actions and checks; approval determines whether the proposal becomes a shared rule. Adoption compiles its structured specification into runtime bindings. Amendments repeat this governance process; retirement disables the rule while preserving its history. Rule content changes online; model parameters and validation/execution code remain fixed.

For measurement, *formation* requires adoption followed by repeated use across independent organizational use units and distinct times, sustained for a minimum duration. All revisions of one protocol within a run count as one *lineage*. Stronger evidence links execution or enforcement by other members to work-object changes and independently evaluated outcomes. These records establish sustained practice; controlled outcome comparisons test its benefit (Appendix C.4.3; Algorithm A3).

3.3 How Protocols Govern Later Work

At scheduling step t, member i has feasible actions $\mathcal{C}_{i,t}$ constructed from member-visible state. The **Structured Decision Layer (SDL)** scores them from work, member, and applicable protocol features (Figure 2). Its stochastic branch converts these scores into selection probabilities:

$$u_{i,t}(a) = \sum_{k=1}^{32} w_k(i,t)\,\phi_k(a, x_{i,t}) + g_{i,t}(a),$$
$$\pi_{i,t}(a) = \frac{\exp\big((u_{i,t}(a) + \epsilon_{i,t,a})/\tau\big)}{\sum_{a' \in \mathcal{C}_{i,t}} \exp\big((u_{i,t}(a') + \epsilon_{i,t,a'})/\tau\big)}. \tag{1}$$

Here $x_{i,t}$ contains visible work and applicable protocols; ϕ_k measures progress, role fit, coordination, evidence, or protocol obligations. Weights $w_k(i,t)$ combine fixed coefficients with member profile/skill inputs. The adjustment $g_{i,t}$ combines authority, reputation, coding preference, and behavioral constraints. Stochastic selection samples from $\pi_{i,t}$, with temperature τ and reproducible random perturbations $\epsilon_{i,t,a}$; deterministic selection chooses a highest-scoring action. Figure 2 abbreviates the first line of Equation 1 as $u = \sum w\phi + g$.

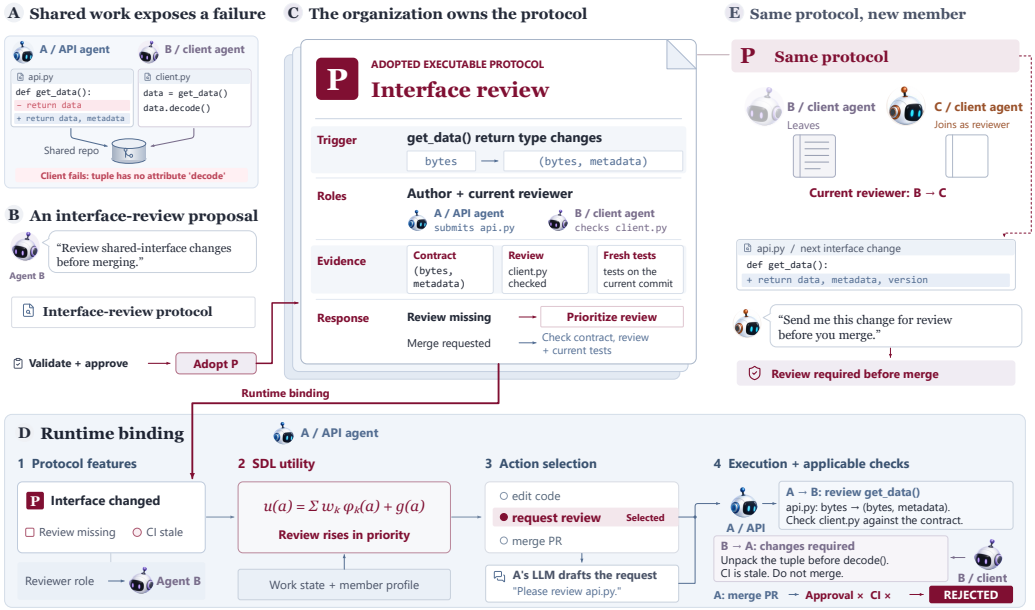

Figure 2: **How Relic forms, binds, and retains a protocol.** Visible work evidence becomes an organization-owned protocol through governed adoption. Protocol and work/member features feed SDL scoring (Equation 1); role mappings and applicable checks connect selected actions and generated content to shared work. The same protocol P governs a replacement reviewer with empty private memory.

From protocol to decision. An adopted interface-review rule supplies responsibility and evidence obligations to features ϕ_k. These can change relative utilities $u_{i,t}$ and probabilities $\pi_{i,t}$ in Equation 1, prioritizing review or evidence work. The scoring map stays fixed; protocols change its inputs. Skill-backed inputs also evolve through the shared member-learning path.

From decision to execution. Role mappings identify who acts. When an action needs code or messages, the LLM supplies them to its handler—the code implementing that action. Routine

selection itself needs no LLM call. SDL scores set action priorities; applicable execution checks update or reject work-object transitions, and protocol events record subsequent use or enforcement. The action registry and handlers remain fixed as protocols evolve (Appendices C.3.1 and C.3.2).

3.4 Member-Independent Reuse

Figure 2 retains protocol P as reviewer B leaves and C joins with empty private memory: the obligation follows the role. Replacement resets private memory, workspaces, sandboxes, message-read state, commitments, reflections, wishes, and accumulated experience. Shared work can remain (Figure 1). The reported transfer experiment goes further: a fresh target product receives only the designated protocols and role mappings, without source-product code, source-world state, or private histories.

Text supplies the frozen rules as reusable prose to model-mediated work, including code editing; **Exec** exposes the same readable content and additionally compiles it into runtime bindings. Their comparison isolates the added value of executable organizational binding on the shared SDL backbone (Appendix G).

Controlled configurations. We use **B0** for a reflective single agent and **B1** for an eight-member long-horizon team with shared work and peer review. **B2** adds SDL selection, profile conditioning, and member-local updates to skills, reputation, and authority; its protocol lifecycle is disabled. Relic (**B3**) shares this backbone and enables the governed protocol lifecycle: members propose rules from recurring friction; validated and approved rules become persistent organizational protocols whose bindings affect later action selection, responsibility routing, and evidence checks. The selector, model parameters, and execution code remain fixed. B3–B2 tests this added lifecycle.

4 Evaluation: Formation, Effectiveness, and Transfer

Persistent work, interacting tasks, and recurring coordination demands motivate our evaluation of formation, verified delivery, executable binding, fresh-member transfer, and external portability. The main study tests multi-member software production with B0 as the single-agent reference; ProgramBench attaches a frozen protocol layer to a single coding agent; CooperBench evaluates two-agent delivery of interacting features.

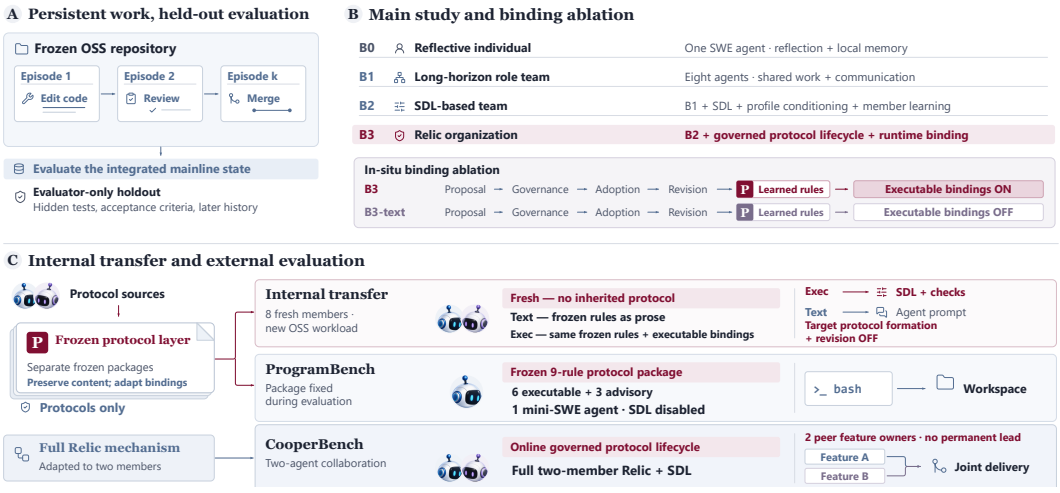

Figure 3: **Evaluation design.** B0–B3 share frozen workloads and 336 steps. B3-text retains governed rule creation and readable rules but removes bindings; Text/Exec hold rule content fixed. ProgramBench tests protocol portability without SDL; CooperBench tests two-agent interacting-feature delivery.

Workloads and outcomes. Five tasks build specified programs from starter skeletons; five repair or extend frozen repository versions toward later-version requirements. Their interdependent implementation, testing, review, and integration create repeated coordination demands. Related acceptance cases form a *contract*, complete only when every case passes on mainline. *Exposed* and *held-out* cases test visible and withheld requirements. *Evaluator-confirmed seeded issues* are initial benchmark

problems verified as resolved relative to failing starter behavior; "seeded" concerns task construction, not repeated-run seeds. Workspace scores separate local changes from integrated delivery. Private acceptance tests and reference implementations remain hidden (Appendix E).

Conditions and design. GPT-5.6 Terra, Claude Opus 4.6, and DeepSeek-V4-Flash each run ten workloads with three random seeds under B0–B3, totaling 360 controlled runs. Relic (B3) and the structured control (B2) share model, tools, task-visible information, SDL parameters, member learning, and execution code; their comparison tests the governed protocol lifecycle. All conditions use 336 scheduling steps, with token expenditure recorded for each run (Appendices F and D.5).

Binding, transfer, and external evidence. A separate 30-run Claude Opus 4.6 ablation compares B3 with **B3-text**, its prose-only variant, which retains proposal, governance, adoption, revision, and readable rules but removes learned-protocol executable bindings. Text and Exec then provide a content-matched test: each adds 30 GPT-5.6 Terra runs with fresh members and the same frozen protocol package; only Exec receives executable bindings, while new target protocol proposal, adoption, and revision are disabled. Fresh reuses the 30 corresponding B2 runs. ProgramBench compares mini-SWE-agent with and without executable protocols on the same 25 tasks, without SDL. CooperBench evaluates the complete two-member Relic architecture on the full benchmark against released peer and Team references, while a fixed 48-pair same-model subset provides the direct Solo–Peer coordination comparison.

5 Results

We first test verified delivery, then examine learned protocols, runtime binding, and transfer.

Verified software-production outcomes — 360 runs

Metric	B0	B1	B2	B3	B3–B2
Complete contracts on mainline ↑	8.22% [4.69, 12.32]	15.97% [11.01, 21.57]	14.06% [9.92, 21.09]	**19.76%** [14.23, 25.92]	**+5.71 pp** [2.97, 8.55]
Held-out cases on mainline ↑	6.86% [0, 16.79]	8.17% [0, 20.88]	9.57% [1.16, 26.48]	**17.28%** [4.46, 45.02]	**+7.72 pp** [2.67, 17.63]
Exposed cases on mainline ↑	11.97% [6.72, 18.16]	22.56% [14.95, 31.40]	19.31% [14.00, 31.94]	**25.89%** [17.47, 35.09]	**+6.58 pp** [3.41, 10.14]
Evaluator-confirmed seeded issues ↑	10.69% [6.02, 16.18]	20.46% [13.73, 28.25]	17.91% [12.16, 24.16]	**24.87%** [17.30, 33.14]	**+6.96 pp** [4.15, 10.83]
Avg. tokens / run ↓	**3.385M** [3.216, 3.565]	45.200M [42.993, 47.550]	4.274M [3.861, 4.693]	7.259M [6.278, 8.510]	**+2.985M** [1.870, 4.364]
Tokens / confirmed issue ↓	4.757M [3.404, 5.449]	40.069M [32.443, 49.793]	**4.010M** [3.179, 5.327]	5.630M [3.994, 7.508]	**−0.448M** [−1.573, +0.659]

Table 1: **Main study (three models; 360 runs).** Brackets: 95% CIs. B3–B2 is paired; cost/issue means use arm-specific eligible blocks, with 20 common blocks for the contrast.

5.1 Executable Protocols Improve Verified Delivery

Relic **improves all four verified endpoints** over the structured control (Table 1). Across all 360 runs, complete contracts rise from 14.06% to 19.76% (+5.71 pp, 95% CI [2.97, 8.55]); held-out, exposed, and confirmed-issue gains are +7.72 [2.67, 17.63], +6.58 [3.41, 10.14], and +6.96 [4.15, 10.83] pp, respectively. Every endpoint also has a positive B3–B2 point estimate in each of the three model strata (Table 41). Appendix I.9 reports the robustness analyses.

Figure 4 tracks declarations, verified mainline delivery, and locally passing work across all three models.

5.2 What Protocols Does the Organization Learn?

Across the Terra+Opus 60-run Relic census, we record 497 autonomous protocol proposals, counting one rule and all its revisions as a single within-run lineage. Of these, 393 remain adopted at the endpoint and 280 satisfy the sustained-use criterion in Section 3.2; 32 additionally link execution by other members to shared-state changes and independently evaluated outcomes. Review/merge, release engineering, and evidence governance account for 87.1% of formed lineages. These rules organize the transition from local changes to verified shared delivery. Appendix I.5.5 gives the lineage census.

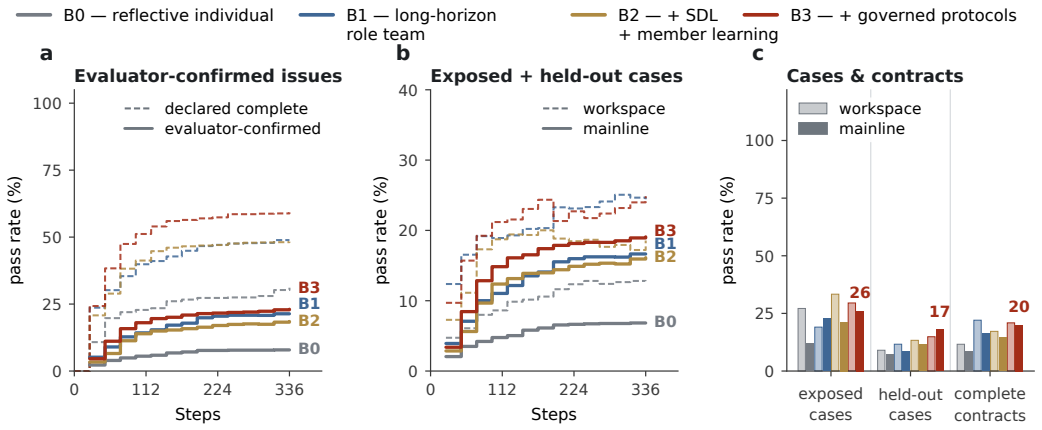

Figure 4: **Verified production outcomes across three models.** Declared and verified completion, workspace and mainline trajectories, and endpoint pass rates.

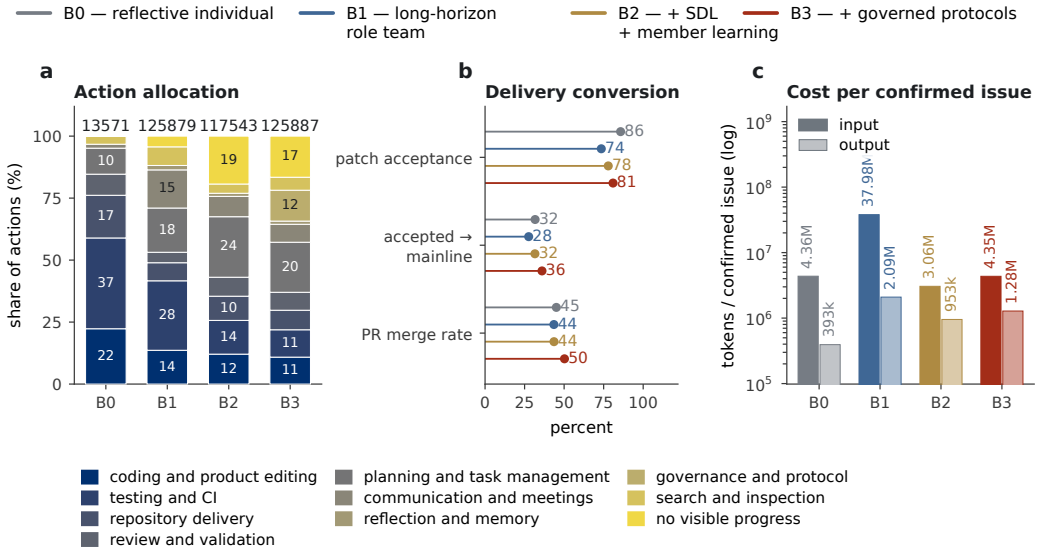

Figure 5: **Work allocation, delivery, and resource use across three models.** Action shares, delivery conversion, and input/output tokens per evaluator-confirmed issue across B0–B3.

Figure 5 links these rules to work allocation and delivery. Relic moves 36% of accepted work to mainline versus 32% for the structured control; its pull-request (PR) merge rate is 50% versus 44%. The single agent has the highest patch-acceptance rate (86%, versus Relic's 81%) but the lowest complete-contract score. Local acceptance alone does not ensure delivery. Section 6 traces one protocol's content and execution.

Across the three-model pooled accounting, Relic averages 7.259M tokens/run versus the control's 4.274M (+2.985M, 95% CI [1.870M, 4.364M]). Tokens per evaluator-confirmed issue average 5.630M for B3 and 4.010M for B2 over each arm's eligible model–workload blocks; the paired B3–B2 estimate over 20 common model–workload blocks is −0.448M (95% CI [−1.573M, +0.659M]; Appendix H.2.4). The Terra+Opus matched-spend analysis additionally selects the last saved Relic checkpoint within each matched control run's realized token spend. Under this matched-spend sensitivity, all four verified endpoints remain positive: +7.45 pp for complete contracts, +5.08 pp for held-out cases, +11.72 pp for exposed cases, and +6.21 pp for evaluator-confirmed issues (Appendix I.7.3).

5.3 Executable Binding and Member-Independent Transfer

Table 2 compares fresh members receiving no inherited rules (Fresh), a frozen protocol package as prose (Text), or identical content with runtime bindings (Exec). Source selection, runtime adaptation, and wording are matched between Text and Exec; neither learns or revises target protocols.

Internal protocol transfer					
Metric	**Fresh**	**Text**	**Exec**	**Exec–Fresh**	**Exec–Text**
Behavioral-case pass rate ↑	25.4% [13.2%, 41.1%]	34.6% [24.1%, 46.4%]	**41.2%** [23.4%, 54.5%]	**+15.8 pp** [9.2, 31.8]	**+6.5 pp** [0.7, 15.6]
Avg. tokens / run (M) ↓	**2.381** [1.747, 3.189]	2.615 [2.213, 3.033]	2.448 [2.199, 2.720]	**+0.067** [−0.569, +0.532]	**−0.167** [−0.467, +0.073]
Tokens / evaluated case ↓	176k [106k, 271k]	154k [96k, 265k]	**144k** [94k, 243k]	**−32k** [−112k, 31k]	**−10k** [−33k, 4k]
Tokens / verified pass ↓	691k [488k, 1,419k]	444k [271k, 935k]	**350k** [196k, 958k]	**−342k** [−818k, −14k]	**−94k** [−152k, 54k]

Table 2: **Internal protocol transfer.** Brackets give 95% confidence intervals; Exec–Fresh and Exec–Text report Exec minus Fresh and Exec minus Text, respectively.

Exec passes 41.2% of behavioral cases, exceeding Text's 34.6% by 6.5 pp (95% confidence interval [0.7, 15.6]) and Fresh's 25.4% by 15.8 pp. **Runtime binding adds value to the same readable content**; Table 2 reports costs, with complete outcomes in Appendix I.8.

Binding ablation during rule creation. A separate 30-run Claude Opus 4.6 experiment compares Relic with its prose-only variant (B3-text) on matched workloads and random seeds. The variant retains proposal, governance, adoption, revision, and readable rules but removes learned-protocol bindings. Complete contracts fall from 22.20% to 15.03%, a +7.18 pp paired advantage for Relic (95% confidence interval [+3.54, +10.91]). Held-out, exposed, and confirmed-issue outcomes have the same direction. Every B3-text run ends with readable rules; learned-protocol bindings, automatic use logging, and protocol-specific enforcement are disabled. Both conditions independently develop their rules from $t = 0$, while Text/Exec transfer holds realized content fixed. Appendix I.10 reports all four endpoint intervals and the binding manipulation check.

5.4 External Software Benchmark Extensions

A. CooperBench — full benchmark		
System	**Successful pairs**	**Role in comparison**
Relic, 2-member	**367/477 (76.9%)**	**peer organization; no fixed lead**
Released Peer / coop+git	282/477 (59.1%)	strongest released peer reference
Team no-proto	354/477 (74.2%)	strongest released lead–member reference
B. CooperBench — same-model 48-pair coordination check		
System	**Successful pairs**	**Role in comparison**
Relic, 2-member (Claude Opus 4.6)	**29/48**	**two peer feature owners; no fixed lead**
Official Solo (Claude Opus 4.6)	26/48	one agent; both features
Official Peer (Claude Opus 4.6)	13/48	two peers; split features
C. ProgramBench — same 25 tasks		
System	**Mean behavioral pass**	
Official mini-SWE-agent	64.164%	
+ executable protocols	**70.916%**	

Table 3: **External benchmark extensions.** CooperBench full-benchmark results exclude broken benchmark pairs under the same exclusion set for every compared system; audit details and raw results are reported in Appendix M. ProgramBench settings are reported in Appendix N.

On **CooperBench**, Relic evaluates the full benchmark with two peer feature owners and no permanent lead. After excluding broken benchmark pairs under the same exclusion set for every compared system, Relic reaches **367/477 (76.9%)**, compared with 282/477 (59.1%) for the strongest released peer reference and 354/477 (74.2%) for the strongest released hierarchical Team reference. Relic

therefore exceeds the released peer reference by 17.8 pp and the hierarchical Team reference by 2.7 pp. On the fixed 48-pair same-model subset, Official Peer falls from Solo's 26/48 to 13/48, whereas Relic reaches 29/48, reversing the *coordination curse* on this controlled comparison.

On **ProgramBench**, six executable and three advisory rules cover originality, evidence freshness, verification, and submission readiness. This frozen package raises mini-SWE-agent's mean behavioral pass rate on 25 tasks from 64.164% to 70.916% (+6.752 pp; +10.5% relative; Appendix N).

6 Case Study: From Repeated Friction to a Governing Rule

In one matched main-study case, we trace an integration problem into a learned protocol, holding model, random seed, tools, and horizon fixed.

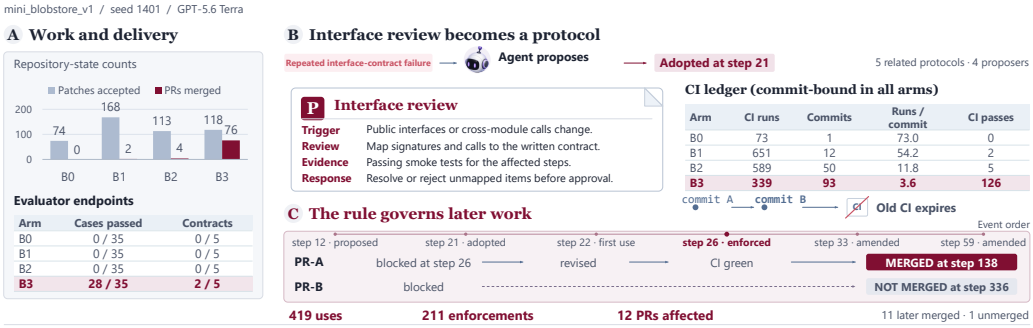

Figure 6: **What the organization learned.** A matched unit shows local delivery, convergent verification rules, and one rule's enforcement and revision. $t12$ denotes scheduling step 12.

Before: activity without sustained delivery. The role-based team (B1) accepted 168 patches but merged only two PRs; Relic accepted 118 and merged 76 (Figure 6a). Local activity therefore did not reliably become shared delivery.

Formation: repeated friction becomes a rule. Repeated interface and verification failures produced an interface-review protocol requiring changed public interfaces to be mapped to the written contract and backed by current test evidence before approval (Appendix J.5.1).

After: the rule governs later work. The protocol was proposed at step 12, adopted at step 21, first enforced at step 26, and amended at steps 33 and 59. It records 211 enforcements across 12 PRs and 419 uses through step 336; 11 affected PRs later merged, including PR-A after its initial block (Figure 6c; Appendix J).

7 Discussion, Future Work, and Conclusion

Discussion. Relic represents organizational capability as shared rules that are formed from experience, executed, revised, and reused by later members. B3–B2 tests the governed lifecycle, B3-text tests its executable bindings, and content-matched Text/Exec tests reuse by fresh members. B3 and Transfer Exec also instantiate learned and fixed-rule regimes. Verified gains recur across model strata, persist in the two-model workload-omission and matched-spend analyses, and extend to external software benchmarks. Rule quality, revision, and retirement shape how this organizational layer develops over time (Appendix A).

Future work. Our evaluation focuses on software-production settings; broader domains, selective protocol transfer and forgetting, rule-quality diagnosis, alternative decision layers, and human–organization interaction remain important directions (Appendix B).

Conclusion. Relic turns collaboration experience into governed, executable protocols held outside member-local histories. Across the main study, binding ablation, content-matched transfer, fresh-member reuse, and external benchmarks, the results support persistent executable organizational state as a useful layer for long-horizon, persistent multi-agent collaboration. The organizational state remains explicit and revisable.

Ethics Statement

The human-centered component of this work consists of formative paired walkthroughs of the P2 and P3 interfaces by four members of the author team who did not participate in the design or implementation of Relic or the two interfaces. Participants were informed in advance that their interaction experience and feedback would be used for research purposes and reported as part of this study. These walkthroughs are reported as formative internal observations of the implemented interfaces.

Reproducibility Statement

The appendices document the experimental conditions, frozen workload construction and qualification, model and decoding configurations, random seeds, run protocol, prompts, protocol representations, metric definitions, and statistical procedures used in the reported experiments. They additionally specify evaluator isolation, artifact and configuration identities, checkpointing and replay, retry and recovery policies, and condition-conformance checks. The CooperBench and ProgramBench extensions are documented separately with their fixed evaluation sets, system configurations, and scoring procedures. These materials are intended to support reproduction and inspection of the reported experimental settings, comparisons, and analyses.

AI Use Statement

Generative AI tools, including ChatGPT, Codex, Claude, and Claude Code, assisted with benchmark construction, methodological and experimental design, implementation and debugging, analysis of experimental outputs, and manuscript drafting and editing. For the constructed workloads, AI assistance included producing technical specifications, reference implementations, tests, and verifiers from human-selected functional briefs; retained benchmark cases were subsequently qualified against frozen starter and reference implementations before evaluation. The authors reviewed AI-assisted artifacts and analyses, verified the reported results, and made all final research decisions. The authors take responsibility for the manuscript and associated research artifacts.

References

Mark S. Ackerman. The intellectual challenge of CSCW: The gap between social requirements and technical feasibility. *Human–Computer Interaction*, 15(2–3):179–203, 2000. doi: 10.1207/S15327051HCI1523_5.

Elie Bakouch and Prime Intellect. Measuring autonomous AI research. Prime Intellect Blog, August 2026. URL https://www.primeintellect.ai/blog/measuring-autonomous-research. Published August 14, 2026.

CooperBench. CooperBench: Official implementation. GitHub repository, 2026a. URL https://github.com/cooperbench/CooperBench. Accessed 2026-09-14.

CooperBench. CooperBench coordination study: Agent trajectories. Hugging Face Dataset, 2026b. URL https://huggingface.co/datasets/CooperBench/team-trajectories. Accessed 2026-09-14.

Hongyi Du, Jiaqi Su, Jisen Li, Lijie Ding, Yingxuan Yang, Peixuan Han, Xiangru Tang, Kunlun Zhu, and Jiaxuan You. ProtocolBench: Which LLM MultiAgent protocol to choose? In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2510.17149.

Martha S. Feldman and Brian T. Pentland. Reconceptualizing organizational routines as a source of flexibility and change. *Administrative Science Quarterly*, 48(1):94–118, 2003. doi: 10.2307/3556620.

Zhezheng Hao, Tianfu Wang, Huanshuo Dong, Ziyan Liu, Hong Wang, Xiankun Lin, Qiang Lin, Can Wang, Hande Dong, and Jiawei Chen. Evolve as a team: Collaborative self-evolution for LLM-based multi-agent systems, 2026. URL https://arxiv.org/abs/2605.29790.

Sirui Hong, Mingchen Zhuge, Jiaqi 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. *arXiv preprint arXiv:2308.00352*, 2023. URL https://arxiv.org/abs/2308.00352.

Arpandeep Khatua, Hao Zhu, Peter Tran, Arya Prabhudesai, Frederic Sadrieh, Johann K. Lieberwirth, Xinkai Yu, Yicheng Fu, Michael J. Ryan, Jiaxin Pei, and Diyi Yang. CooperBench: Why coding agents cannot be your teammates yet, 2026. URL https://arxiv.org/abs/2601.13295.

Charlotte P. Lee. Boundary negotiating artifacts: Unbinding the routine of boundary objects and embracing chaos in collaborative work. *Computer Supported Cooperative Work*, 16(3):307–339, 2007. doi: 10.1007/s10606-007-9044-5.

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. *arXiv preprint arXiv:2303.17760*, 2023. URL https://arxiv.org/abs/2303.17760.

Wenhan Lyu, Yue Xiao, Yixuan Zhang, and Yifan Sun. Self-organizing multi-agent systems for continuous software development, 2026. URL https://arxiv.org/abs/2603.25928v1.

Richard R. Nelson and Sidney G. Winter. *An Evolutionary Theory of Economic Change*. Harvard University Press, Cambridge, MA, 1982.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023. URL https://arxiv.org/abs/2307.07924.

Prithvi Rajasekaran. Harness design for long-running application development. Anthropic Engineering, March 2026. URL https://www.anthropic.com/engineering/harness-design-long-running-apps. Published March 24, 2026.

Kjeld Schmidt and Liam Bannon. Taking CSCW seriously: Supporting articulation work. *Computer Supported Cooperative Work*, 1(1–2):7–40, 1992. doi: 10.1007/BF00752449.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://papers.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

David J. Teece, Gary Pisano, and Amy Shuen. Dynamic capabilities and strategic management. *Strategic Management Journal*, 18(7):509–533, 1997. doi: 10.1002/(SICI)1097-0266(199708)18:7⟨509::AID-SMJ882⟩3.0.CO;2-Z.

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

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

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2405.15793.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. ProgramBench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

Justin Young. Effective harnesses for long-running agents. Anthropic Engineering, November 2025. URL https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents. Published November 26, 2025.

Zhengxu Yu, Yu Fu, Zhiyuan He, Yuxuan Huang, Ka Yiu Lee, Meng Fang, Weilin Luo, and Jun Wang. From skills to talent: Organising heterogeneous agents as a real-world company, 2026. URL https://arxiv.org/abs/2604.22446.

Guibin Zhang, Muxin Fu, Guancheng Wan, Miao Yu, Kun Wang, and Shuicheng Yan. G-Memory: Tracing hierarchical memory for multi-agent systems, 2025. URL https://arxiv.org/abs/2506.07398.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners, 2023. URL https://arxiv.org/abs/2308.10144.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. MultiAgentBench: Evaluating the collaboration and competition of LLM agents. *arXiv preprint arXiv:2503.01935*, 2025. URL https://arxiv.org/abs/2503.01935.

APPENDIX CONTENTS

A Broader Discussion: Governance, Rule Quality, and Organizational Trade-offs **15**

A.1 Evidence Map: What Each Experiment Establishes 15

A.2 Organizational Capability as a Systems Layer 15

A.3 Autonomous Formation or Fixed Protocol Packages? 15

A.4 The Organizational Maturity Gap . . 15

A.5 Governance Trades Cost for Reliability 16

A.6 Strong Execution Raises the Stakes of Rule Quality 16

A.7 From Rule Formation to Evidence-based Meta-governance 16

B Research Agenda for Persistent AI Organizations **16**

B.1 Structured Decision Layers 16

B.2 Organizations as a First-class Evaluation Unit 16

B.3 Capability and Capacity Taxonomies 16

B.4 Transfer and Selective Forgetting of Organizational State 17

B.5 Capability Ecology and Failures of Organizational Learning 17

B.6 Human–Agent Organization Interaction 17

B.7 Toward Agentic Sociology 17

C Extended Method and Formal Specification **17**

C.1 State and Member Information . . . 17

C.2 Persistent Organizations and Organizational Learning 19

C.3 Structured Decision Layer and Execution 19

C.4 Capability Formation and Governance 21

C.5 Core Experimental Invariants 22

D Execution-Grounded Organizational Environment **22**

D.1 Environment and Work State 22

D.2 Shared Work, Visibility, and Communication 23

D.3 Action Surface and Execution Validation 23

D.4 Software-Production Lifecycle and Verification 23

D.5 Time, Scheduling, and Resource Accounting 24

D.6 Organizational Runtime and Protocol Activation 24

D.7 Logging, Checkpointing, and Replay 24

E Benchmark Curation **24**

E.1 Evaluation Scope and Workload Taxonomy 25

E.2 Workload Sourcing and Eligibility . . 26

E.3 Freezing and Manifest Construction . 27

E.4 Task and Evaluator Construction . . 28

E.5 Quality Control and Preflight 29

E.6 Completed Main Study and Evaluation Coverage 29

F Experimental Conditions and Run Protocol **30**

F.1 Exact B0–B3 Definitions 30

F.2 Roster, Functions, and Decision Priors 31

F.3 Prompts and Information Controls . . 32

F.4 Matched Experimental Controls . . . 33

F.5 Run Protocol 34

F.6 Condition-Conformance Tests 36

G Internal Protocol Transfer and Executable Binding **36**

G.1 Protocol Source and Bundle Freezing 36

G.2 Fresh-Member Initialization 37

G.3 Organizational-State Arms 37

G.4 Transfer Payload 38

G.5 Content Matching and Binding Control 39

G.6 Transfer Estimands and Evaluation Scope 39

H Metrics and Statistical Analysis **39**

H.1 Canonical Data Sources 39

H.2 Metric Dictionary 40

H.3 Aggregation, Coverage, and Missingness 42

H.4 Estimands 42

H.5 Statistical Analysis and Supplementary Checks 43

I Full Results and Robustness Checks **43**

I.1 Run Inventory 43
I.2 Workload, Seed, and Family Outcomes 44
I.3 Full Endpoint and Time-Series Results 45
I.4 Action Allocation and Delivery Conversion 45
I.5 Organizational-Mechanism Inventory 46
I.6 Model-Conditioned Friction and Institutional Responses 50
I.7 Cost and Efficiency Results 51
I.8 Complete Transfer Results 52
I.9 Robustness Checks 52
I.10 In-Situ Prose-Only Institutionalization Ablation 54

J Case-Study Evidence: Commit-Bound Verification 56
J.1 Case Configuration 56
J.2 Repository and Action Counts 56
J.3 Work Without Delivery 57
J.4 Capability Inventory and Governance 58
J.5 Related Rules for Verification and Evidence Freshness 58
J.6 CI Ledger 59
J.7 One Rule's Governed Lifecycle . . . 60
J.8 Pull-Request-Level Consequences . . 60
J.9 Case Interpretation 61

K Integrity and Reproducibility 61
K.1 Evaluator Isolation 61
K.2 Temporal and Network Controls . . . 61
K.3 Benchmark Identifiers and Artifact Joins 62
K.4 Run-Bundle Schema 62
K.5 Checkpointing and Replay 63
K.6 Run Validity and Recovery 63
K.7 Prompts, Schemas, and Configuration Artifacts 64

L Human–Organization Interfaces 68
L.1 Why Direct Human Participation Becomes Difficult 68
L.2 Moving the Human–Organization Abstraction Boundary 69
L.3 P3: A Liaison-First Hybrid Human–Agent Organization 70
L.4 Bidirectional Translation 70
L.5 Human Attention as an Organizational Resource 71
L.6 Formative Paired Walkthrough Protocol 72
L.7 Paired Internal Walkthrough Results 72
L.8 Authority and Interaction Records . . 73

M External Software Benchmark: CooperBench 73
M.1 Full-Benchmark Scope and Success Criterion 73
M.2 Two-Member Relic Adaptation . . . 73
M.3 Baseline Configurations 74
M.4 Benchmark-Validity Audit 74
M.5 Released-Baseline Sensitivity 78
M.6 Relic Results and Matched-Pair Sensitivity 78
M.7 Frozen 48-Pair Same-Model Coordination Check 79

N External Software Benchmark: ProgramBench 80
N.1 System Configuration 80
N.2 Task Selection and Fixed Evaluation Set 80
N.3 Frozen Protocol Package and Runtime Binding 81
N.4 Runtime Footprint 81
N.5 Scoring and Outcomes 82

A Broader Discussion: Governance, Rule Quality, and Organizational Trade-offs

A.1 Evidence Map: What Each Experiment Establishes

Claim	Evidence line	Key result
Autonomous organizational state forms and persists	Terra+Opus 60-run B3 formation census	497 autonomous proposals; 405 ever adopted; 280 formed lineages, including 32 with strong object/evaluator-linked evidence.
Institutionalization improves verified delivery	B3–B2, 360 controlled runs	Complete +5.71 pp [2.97, 8.55]; held-out +7.72 pp [2.67, 17.63]; exposed +6.58 [3.41, 10.14]; confirmed +6.96 [4.15, 10.83].
Executable binding matters during online formation	B3 vs. B3-text, 30 matched runs	+7.18 pp complete contracts [3.54, 10.91] while proposal, governance, adoption, revision, and readable rules remain active.
Executable binding adds value to fixed readable knowledge	Exec vs. Text, 30 matched targets	Behavioral correctness +6.5 pp [0.7, 15.6] with identical frozen readable rule content.
Organizational state remains useful to fresh members	Exec vs. Fresh	Behavioral correctness +15.8 pp [9.2, 31.8] after member-local state is reinitialized.
Verified gains persist at matched realized token spend	B3 checkpoint under each matched B2 token cap	Complete +7.45 pp; held-out +5.08 pp; exposed +11.72 pp; confirmed issues +6.21 pp; all four 95% intervals remain above zero.
Peer coordination improves externally	CooperBench full benchmark + fixed 48-pair same-model control	Full benchmark: Relic 367/477 (76.9%) vs. Peer 282/477 (59.1%) and Team no-proto 354/477 (74.2%) after excluding broken benchmark pairs; same-model control: Official Peer 13/48, Solo 26/48, Relic 29/48.
Protocol behavior transfers to another harness	ProgramBench, fixed 25 tasks	Mean behavioral pass 64.164% $\rightarrow$ 70.916% (+6.752 pp; +10.5% relative).

Table 4: **Experimental results across formation, binding, transfer, cost, and external benchmarks.**

A.2 Organizational Capability as a Systems Layer

Relic separates model or member capability from the persistent state that governs how multiple members work together. A stronger member can improve local reasoning, implementation, or tool use, while an organization additionally determines how evidence is routed, responsibility is assigned, work is reviewed, and local progress becomes shared delivery. The capability census gives this distinction empirical content: formed lineages are strongly concentrated in review/merge, release engineering, and evidence governance. In this software-production setting, the observed institutional responses concentrate near the boundary between local work and trustworthy, integrated shared delivery. Persistent organizational state therefore acts as an additional systems layer through which recurring coordination problems are represented and acted on.

A.3 Autonomous Formation or Fixed Protocol Packages?

B3 develops its operating rules through proposal, governance, execution, revision, and retirement. Transfer Exec deploys a source-derived frozen rule package to fresh members. These configurations support two deployment modes: adapting organizational rules during work and reusing an established rule package at initialization.

A.4 The Organizational Maturity Gap

Current agent systems exhibit an asymmetry between rapidly improving individual capability and comparatively thin organizational structure. Agents can already write code, use tools, search, plan, and execute long tasks, yet collective work is often coordinated through temporary conversations, manually assigned roles, shared scratchpads, or procedures reconstructed anew in each run. We call this gap an *organizational stone age*: individual agent capability is advancing faster than the persistent structures that organize collective work.

Natural language alone does not close this gap. Agents already possess a language for communicating content; what remains underdeveloped is a durable *language of work* that specifies responsibility, evidence, review, integration, exceptions, and revision in forms that continue to govern future

members. Relic studies executable protocols as one such representation. Other organizational objects—routing policies, role structures, escalation paths, shared abstractions, or resource-allocation rules—may support different forms of persistent collective capability.

A.5 Governance Trades Cost for Reliability

Governance routes responsibility, review, and verification through shared procedures. B3 incurs additional tokens per run, while its matched-spend checkpoints retain gains on all four verified production endpoints (Appendix I.7.3).

A.6 Strong Execution Raises the Stakes of Rule Quality

Rule triggers, evidence requirements, responsibilities, and revision paths determine how organizational experience governs later work. Relic makes these components explicit, supporting rule inspection, amendment, retirement, and reuse across member turnover.

A.7 From Rule Formation to Evidence-based Meta-governance

As organizations grow, forming rules is only the first governance problem. A larger system must also decide who may propose, approve, override, revise, or retire rules; whether a rule applies to one team or the whole organization; how conflicting rules are resolved; and when exceptions are legitimate. Persistent agent organizations therefore eventually require governance of governance.

A natural extension is evidence-based rule deployment. A proposed mechanism could begin in advisory or shadow mode, run within a limited scope, be evaluated against explicit outcomes, expand only after successful trials, and carry revision or sunset conditions. Many familiar human governance ideas–independent review, staged adoption, appeals, exceptions, delegated authority, and sunset clauses–provide useful design hypotheses. Agent organizations also make forms of governance experimentation unusually tractable: organizational state can be logged exactly, policies can be versioned, and in suitable settings a rule can be snapshotted, rolled back, or compared under controlled replay.

B Research Agenda for Persistent AI Organizations

These results motivate a research program on persistent AI organizations, with organizational capability as a common unit of analysis.

B.1 Structured Decision Layers

Future work can compare fixed and learned decision layers, alternative action-space abstractions, and responsibility-, risk-, and cost-aware routing across agent harnesses, domains, and model strengths.

B.2 Organizations as a First-class Evaluation Unit

Persistent AI systems can be evaluated at the level of the organization rather than only the individual model or temporary team. Future benchmarks could vary topology, hierarchy, authority allocation, role specialization, membership turnover, and organizational memory while holding the underlying work environment fixed. Such studies could reveal organization-level failure modes that task-bounded agent benchmarks cannot express.

B.3 Capability and Capacity Taxonomies

A broader evaluation framework should distinguish where capability resides. Model capability concerns what the underlying model can do; agent capability additionally includes tools, memory, and an execution loop; collaborative capability can arise within a temporary team; organizational capability is retained in shared structure that can affect future work independently of the private histories of the members who created it. These layers can be characterized by formation, scope, executability, persistence, transferability, compositionality, and reversibility. A complementary

capacity taxonomy can ask how much work, coordination load, member turnover, and rule complexity each layer can sustain before performance degrades. Such distinctions would make it clearer whether a system improves because of a stronger model, a better agent scaffold, a transient team strategy, or a persistent organizational capability.

B.4 Transfer and Selective Forgetting of Organizational State

Organizational-transfer research can examine partial and gradual turnover, cross-model and cross-domain transfer, interference among transferred mechanisms, and the relative value of transferring protocols, workflows, roles, responsibility maps, or other shared state. Equally important is learning when organizational state should be adapted or deliberately forgotten rather than preserved.

B.5 Capability Ecology and Failures of Organizational Learning

Future work can study how capability families compose, compete, become obsolete, or depend on one another; how these distributions change with model strength and environment; and where recurring friction fails to become institutionalized. A formation funnel from friction exposure through recognition, proposal, adoption, formation, and strong effect evidence could make failures of organizational learning as measurable as successful capability formation.

B.6 Human–Agent Organization Interaction

As agent organizations become more persistent and internally complex, the human interaction target may shift from an individual agent to an organization. Future studies can compare direct human participation, secretary-assisted interaction, and a secretary acting as an organizational interface that summarizes state, surfaces pending decisions, and translates human intent into organizational actions. This raises questions about abstraction, explainability, delegation, approval and veto rights, responsibility attribution, and accountability in multi-human–multi-agent organizations.

B.7 Toward Agentic Sociology

Persistent organizations also create a scale of analysis beyond one team. Multiple agent organizations may specialize, exchange work, depend on shared infrastructure, negotiate interfaces, and develop rules for interaction across organizational boundaries. We use *agentic sociology* as a research lens for studying persistent roles, organizations, norms, institutions, and relations around populations of artificial agents, filling the level between isolated multi-agent interaction and broader multi-organization systems.

At this scale, questions of interoperability and governance separate. A communication protocol may let organizations exchange messages without establishing who owns a task, what evidence satisfies an obligation, which authority may revise a commitment, or how a dispute is resolved. Studying those structures requires benchmarks that span persistent organizations, turnover, cross-organizational dependencies, and human oversight rather than only task-bounded conversations.

Together, these directions suggest a research agenda on how AI organizations make decisions, accumulate and transfer capabilities, govern their own rules, adapt across environments, and remain legible and accountable to humans.

C Extended Method and Formal Specification

This section specifies organizational state, action selection, governance, execution, and capability formation.

C.1 State and Member Information

Let H_t denote the complete recorded history of a run. For member i, let $I_{i,t}^{\mathrm{acc}} \subseteq H_t$ be the information the member is allowed to access, and let $v_{i,t} \subseteq I_{i,t}^{\mathrm{acc}}$ be the bounded context actually read or retrieved

at decision time. The runtime maintains

$$v_{i,t} \subseteq I_{i,t}^{\text{acc}} \subseteq H_t.$$

Private branches, unread messages, private workspaces, and evaluator-only assets therefore need not appear in a member's context even though they exist in the recorded run.

C.1.1 MEMBER-LOCAL STATE

Member-local state contains the information and learned quantities carried by one member: functional role, decision prior, skills, current work, private workspace, private memory, local commitments, and bounded workload state. These quantities can affect that member's later decisions without becoming organization-owned state.

Member-local state used in the experiments

Category	Role in the system
Identity and function	Role, specialty, and stable functional decision prior.
Skill and standing	Skill, reputation, and authority variables used by the structured selector where enabled.
Current work	Active tasks, work status, and local scheduling state.
Private workspace	Member-owned notes, files, drafts, branches, and sandbox outputs.
Private memory	Salient previously observed events retained for later context.
Commitments	Member-specific promises, requests, and unresolved obligations.

Table 5: Member-local state.

C.1.2 ORGANIZATIONAL STATE

We write organizational state as

$$O_t = (W_t, Q_t, R_t),$$

where W_t contains shared work objects, Q_t contains responsibility and decision-right mappings, and R_t contains adopted organizational protocols. The defining distinction is operational: a shared document belongs to W_t; a rule belongs to R_t when it has an active runtime binding that can affect later work.

Organizational state

Part	Contents	Operational role
W_t	Tasks, issues, shared documents, repository artifacts, boards, messages	Persistent shared work and evidence.
Q_t	Task ownership, reviewer/approver responsibility, authority relations	Routes work and assigns organizational responsibility.
R_t	Adopted protocol specifications, lifecycle state, use/violation/enforcement records	Stores learned ways of working that can govern later members.

Table 6: Organizational state.

C.1.3 EXECUTION RECORDS

Evaluator-side provenance records actions, work-object transitions, governance events, and decision traces for measurement and replay. These records preserve the sequence connecting proposal, adoption, later use, and execution consequences. Member decisions continue to use the bounded perception and retrieval interface in Appendix C.1; the complete recorded history supports researcher-side reconstruction of the same trajectory.

C.2 Persistent Organizations and Organizational Learning

A persistent organization is a multi-member system in which learned rules and cross-member responsibilities are stored outside member-local histories and can remain operative when members are replaced. We represent the organization at time t as

$$\mathcal{A}_t = \big(N_t, \{m_{i,t}\}_{i\in N_t}, O_t, (\mathcal{C}, D_\theta, E), \Gamma\big),$$

where N_t is the roster, $m_{i,t}$ is member-local state, O_t is organizational state, $(\mathcal{C}, D_\theta, E)$ is the fixed candidate, decision, and execution substrate, and Γ is the governance process that introduces or revises persistent mechanisms.

C.2.1 Member Learning and Organizational Learning

Let $I_t = (Q_t, R_t)$ denote the institutional component of O_t, containing responsibility and decision-right mappings and adopted protocols. The experiment distinguishes three update paths:

$$\begin{aligned} W_{t+1} &= F_{\text{work}}(W_t, a_t), \\ I_{t+1} &= F_{\text{institution}}(I_t, W_t, a_t, g_t), \\ m_{i,t+1} &= F_{\text{agent}}(m_{i,t}, o_{i,t}, a_t, r_t). \end{aligned} \tag{2}$$

Ordinary work changes shared artifacts; member learning changes a member's local state; organizational learning introduces or revises governed mechanisms whose effects can extend to members who did not create them.

C.2.2 Persistence Across Member Replacement

Member replacement reinitializes private memory, private workspace and sandbox state, message read/acknowledgement state, commitments, reflections, wishes, and accumulated member-local experience. The functional role slots are reinstantiated, and shared work can persist across the replacement. In the reported transfer experiment, a freshly initialized target receives the specified protocol package and responsibility mappings. The target initialization and transferred contents are specified in Appendix G.2.1.

C.3 Structured Decision Layer and Execution

At each decision, the system constructs a feasible candidate set $\mathcal{C}_{i,t}$ from member-visible work state, role constraints, target validity, and the common action surface. Candidate construction is shared by the compared conditions; the action-selection mechanism differs by condition.

C.3.1 Structured Decision Layer (SDL)

B2 and B3 score candidates with the same 32-dimensional, rule-computed feature representation. The feature groups are shown in Table 7.

SDL candidate-feature dimensions	
Feature group	**Dimensions**
Task progress and priority	7
Skill and role fit	4
Social coordination and trust	8
Repository health and evidence quality	6
Protocols and institutional memory	5
Governance-proposal signals	2
Total	**32**

Table 7: SDL feature groups used by B2 and B3.

For candidate a,

$$w_k(i,t) = b_k + \sum_j p_{ij}(t)\beta_{jk}, \qquad s_{\text{base}}(a) = \sum_{k=1}^{32} w_k(i,t)\phi_k(a, x_{i,t}),$$

and

$$u(a) = s_{\text{base}}(a) + A(a) + R(a) + C(a) + \sum_g G_g(a).$$

The stochastic branch samples from

$$P(a) = \frac{\exp((u(a) + \epsilon_a)/0.6)}{\sum_{a'} \exp((u(a') + \epsilon_{a'})/0.6)}, \qquad \epsilon_a \sim \text{Uniform}(-0.05, 0.05),$$

using namespaced seeded randomness. The scoring parameterization is fixed before the reported runs and shared by B2 and B3.

SDL base-weight coefficients		
Group	**Feature**	**Base weight**
Task / progress	progress gain; deadline urgency; task priority; blocker resolution; dependency unlock; demo relevance; customer relevance	$+0.55, +0.30, +0.30, +0.35, +0.20, +0.30, +0.30$
Skill fit	skill match; role affinity; learning gain; low-skill failure risk	$+0.40, +0.30, +0.10, -0.30$
Social / coordination	coordination; clarity; trust gain/risk; conflict risk; visibility; reputation gain/risk	$+0.30, +0.20, +0.25, -0.25, -0.25, +0.20, +0.20, -0.20$
Repository / evidence	repository health; technical debt; review quality; reproducibility; untracked-result risk; claim evidence	$+0.30, -0.30, +0.25, +0.25, -0.20, +0.25$
Protocol	creation potential; use potential; violation risk; enforcement gain; institutional memory	$+0.20, +0.20, -0.40, +0.20, +0.20$
Governance	proposal endorsement; proposal skepticism	$+0.45, +0.45$

Table 8: Complete base-weight vector, grouped for readability.

SDL post-score terms	
Softmax temperature	0.6
Seeded jitter	$\text{Uniform}(-0.05, +0.05)$
Authority bonus	$0.07\times$ domain authority
Reputation bonus	clipped at 0.10
Coding affinity	bounded role–action bonus

Table 9: Post-score terms used by the SDL.

Principal soft-guard penalties	
Repetition	-0.18
Artifact churn	-0.12
Unresolved review	-0.15
Missing new information	-0.20
Opportunity cost	-0.12
Shipping/off-task	-0.50
Unshipped release idle	-1.20

Table 10: Principal soft-guard terms outside the core dot product.

Construct visible feasible candidates $\mathcal{C}_{i,t}$. Apply shared feasibility, cooldown, and pool-size guards. For B0/B1, the model chooses from the candidate set. For B2/B3, extract the 32 features, compute $u(a)$ for each candidate, and select by deterministic maximum or seeded softmax. Record the choice trace for evaluation, then execute the selected action through the common execution substrate.

Algorithm A1: SDL-based collective decision (one member, one step)

C.3.2 EXECUTION OPERATOR

The common execution layer validates the selected action, applies its work-state transition, accounts for costs, and records the resulting events. In standard B3, adopted protocols supply decision features and responsibility signals to the selector, followed by protocol-linked checks and records after the action handler. SDL utilities determine action priorities and selection probabilities; transition validation and execution checks determine the resulting work-object disposition. Shared repository requirements, including review and current-CI evidence, are enforced by the common environment. The action registry and execution handlers remain fixed while adopted protocol specifications and role mappings evolve.

C.4 CAPABILITY FORMATION AND GOVERNANCE

Relic converts visible recurring work friction into proposals. A proposal names the problem, supporting evidence, affected work, required actions, and the intended organizational response. Validation checks grounding, supported actions, scope, duplication, and governance requirements before a proposal can be adopted.

C.4.1 PROPOSAL AND PROTOCOL REPRESENTATION

Proposal and executable-protocol representation	
Group	**Fields**
Identity	proposal type, title, proposer
Evidence	source observations/wishes, target problem, proposed response
Feasibility	required actions and capabilities
Governance	review state, required approvals, support/opposition, adoption score
Lifecycle	amendment target, revision state, retirement state
Protocol content	trigger, scope, responsible roles, required steps/evidence, affected actions/artifacts, violation/exception rules, enforcement response, success metric
Accounting	use, violation, enforcement, revision, and last-use records

Table 11: Proposal and adopted-protocol fields.

C.4.2 GOVERNANCE, ADOPTION, AND RUNTIME BINDING

The reported main-study configuration uses semi-automatic governance: members provide approvals first, while eligible proposals that remain pending for at least 24 steps can receive bounded deadlock recovery. Adoption additionally requires at least two supporters, at least two distinct approvers, a minimum review interval of three steps, and an adoption score of at least 0.6.

Approval-mode semantics	
Auto	Eligible proposals can receive required approvals automatically.
Agent	Required approvals must come from member actions.
Semi-auto	Member approval is primary; bounded deadlock recovery can complete missing approvals after the waiting period.

Table 12: Approval modes supported by the governance layer.

After adoption, a structured protocol specification is registered as organization-owned state. Its trigger, responsibilities, evidence obligations, and affected actions become available to the decision and execution layers. Amendments follow the same governed path; retirement disables future use while retaining lifecycle history.

From visible repeated friction, create a proposal. Validate its grounding and runtime compatibility. Route it for governance. When support, approval, review-time, and adoption-score requirements are satisfied, compile the proposal into a structured protocol and register it as organizational state. Later matching actions can generate use, violation, or enforcement records. Amendments repeat the same governance path; retired protocols cease governing future work while their history remains auditable.

Algorithm A2: Governed organizational-state update

C.4.3 Operational Capability-Formation Criterion

A registered autonomous protocol is counted as formed when it has a valid proposal and adoption record, at least two qualifying post-adoption uses, those uses span at least two recorded organizational units and two distinct steps, and the span from adoption to the last qualifying use is at least 48 steps. Strong evidence additionally links later third-party execution or enforcement to a governed-object state change and independent evaluator evidence.

For each adopted autonomous rule, collect qualifying post-adoption uses. Mark weak formation when the use-count, organizational-unit, distinct-time, and 48-step persistence conditions all hold. Mark strong evidence when weak formation is additionally linked to later third-party execution or enforcement, a governed-object state change, and independent evaluator evidence. The detector reads the recorded ledger and does not modify the run.

Algorithm A3: Offline capability detection (read-only)

C.5 Core Experimental Invariants

Table 13 summarizes the implementation invariants linking member-visible execution, governed updates, controlled comparisons, and measurement. Configuration and access checks are detailed in Appendix F.6.

Experimental invariants	
Member information interface	Member context is built from permitted information that has been read or rendered; evaluator-private assets are held in the evaluator interface.
Cross-member privacy	Sharing explicitly controls the visibility of private branches, workspaces, messages, and sandbox outputs to other members.
Governed institutional updates	Protocol adoption follows proposal validation and the configured review and approval path.
Registered action surface	State mutations occur through registered actions and validated transitions.
Matched B2/B3 substrate	B2 and B3 share the structured selector, candidate construction, member-learning path, task information, and execution substrate; B3 enables institutionalization.
Fresh-member reset	Transfer reinitializes member-local state and applies the designated organizational package and role mappings.
Read-only measurement	Formation detection reads recorded ledgers; evaluators score exported candidate state outside the organizational rollout.

Table 13: Core invariants enforced by the experimental design.

D Execution-Grounded Organizational Environment

This section summarizes the common software-production environment used by all main-study conditions. The environment provides persistent shared work, partial observability, executable work transitions, and a frozen evaluator. Condition-specific organizational learning is layered on top of this common substrate.

D.1 Environment and Work State

Members act on persistent software-production objects: tasks and issues, working-tree artifacts, branches, commits, pull requests, CI records, reviews, shared documents, messages, and release state. Product progress is measured from concrete state transitions.

D.1.1 MAIN-STUDY COMPONENTS

Common environment and condition-specific activation		
Component	**Available**	**Main-study role**
Repository, CI, review, merge, release	yes	Common execution-grounded work substrate.
Tasks, issues, shared board	yes	Common ownership and responsibility substrate.
Channels and direct messages	yes	Common information-friction substrate.
Private workspaces and sandboxes	yes	Common partial-observability substrate.
Structured decision layer	yes	Used for action selection in B2/B3.
Member learning	yes	Enabled in B2/B3.
Protocol proposal, governance, and execution	yes	Institutionalization enabled in B3.
Frozen evaluator	yes	Scores candidate mainline state outside member context.
Token accounting	yes	Measures realized model usage.

Table 14: Environment components relevant to the reported main study.

D.2 SHARED WORK, VISIBILITY, AND COMMUNICATION

The environment separates private, shared, and evaluator-only state. Local edits remain private until committed and exposed through the repository workflow; sandbox outputs remain private until explicitly shared; a shared document can be visible before its full content is opened; and messages become part of a member's context only after the member reads them. Evaluator-only tests, held-out requirements, and references never enter member-visible state.

D.2.1 READ AND RETRIEVAL SEMANTICS

Visibility, delivery, reading, acknowledgement, and memory are distinct. Inbox triage admits at most 25 unread perceivable messages per step. Search operates over frozen internal corpora, repository state, member-local sandbox state, and fixed external-snapshot corpora; no live network retrieval is used in the reported environment.

D.3 ACTION SURFACE AND EXECUTION VALIDATION

The common action surface covers task work, repository operations, verification, communication, meetings, search, documents, artifacts, governance, and release operations. Feasible actions are generated from the member's visible state and role. Selection is followed by transition-specific validation, so an offered action can still fail when its target or precondition has changed.

D.3.1 INVALID ACTIONS AND FAILURES

Execution outcomes		
Failure class	**Where resolved**	**Recorded effect**
Invisible or invalid target	candidate/transition validation	No product mutation; decision is rejected.
Invalid work transition	action-specific validation	Failed execution with recorded reason.
Missing role or approval	action/governance validation	Transition refused.
Malformed model output	structured-output validation	No action mutation; model usage remains counted.
Unavailable action	candidate construction	Action is absent from the feasible set.

Table 15: High-level invalid-action handling used in the experiments.

D.4 SOFTWARE-PRODUCTION LIFECYCLE AND VERIFICATION

Repository-based workloads begin from a frozen initial repository and specify repairs or feature extensions. Construction workloads begin from an unimplemented starter skeleton plus a frozen

specification and public checks. In both cases, the delivery chain is edit → commit → pull request → review/CI → merge, and reported product metrics are evaluated on mainline.

Verification is commit-bound. A CI result records the tested pull-request head and the mainline state against which it was evaluated. Pushing a new commit invalidates earlier evidence for that head, and mainline movement can make a previous result stale before merge. These checks are common environment properties shared by all conditions; Relic can learn organizational rules that route responsibility and evidence around them.

D.5 Time, Scheduling, and Resource Accounting

The main-study horizon is 336 scheduling steps with checkpoints every 24 steps. A scheduling step denotes one simulation update; elapsed execution time is recorded separately in seconds. Members take at most one primary action per step, while background work can complete asynchronously. Runtime records retain the field name `tick` for the scheduling index.

Realized model usage is accumulated from provider receipts, including retry calls. Tool and container compute is unmetered and remains separate from the provider-token totals. Checkpoint resume loads the saved organization and continues under the same run identity, preserving the relation between the trajectory, its scheduling position, and its accumulated resource use.

D.6 Organizational Runtime and Protocol Activation

The shared work layer stores operational artifacts; responsibility mappings identify owners, reviewers, and approvers; adopted protocols add triggers, required evidence, affected actions, responsible functions, exception rules, and lifecycle state. In standard B3, learned rules can reprioritize actions, route responsibility, and participate in protocol-linked execution checks. Amendments follow the same governance path as initial adoption, and retirement stops future application while retaining the recorded history.

D.7 Logging, Checkpointing, and Replay

Each executed action records its actor, action type, success or failure, affected work objects, costs, and resulting events. Governance transitions add proposal, adoption, amendment, and retirement records; protocol execution adds the corresponding use and enforcement events. Checkpoints serialize organizational state, member state, and scheduling state for run recovery.

Case-study timelines are reconstructed from these ledgers and checkpoints. Researcher-facing inspection joins the complete recorded state and event history; member-facing decisions use the bounded perception interface. This separation preserves both the local information used to choose an action and the shared-state consequences used to analyze it.

E Benchmark Curation

This section documents the ten frozen software-production workloads used in the 360-run headline endpoint study: three models, three seeds, and four conditions per workload. W01–W10 are stable identifiers in the frozen analysis `PACK_ORDER`. Table 16 gives their task names, version transitions, and original pack IDs; the same mapping applies to all workload-level results, transfer references, and the selected case. Construction and pre-run qualification are specified below.

Development tasks. Relic was developed and debugged on repositories outside W01–W10 before the reported evaluation. Development records and the 360-run evaluation matrix have separate run identities. Within B3 evaluation runs, online protocol formation and revision remain active components of the method.

E.1 Evaluation Scope and Workload Taxonomy

E.1.1 Frozen-Version Repair and Feature Development

W06–W10 use existing upstream projects with both an initial version and a final reference version frozen during benchmark construction. The complete initial repository is the starter; the final repository is the reference implementation. The exact version pairs appear in Table 16. Feature changes and issue requirements between these versions define the target behaviors, covering both bug fixes and new functionality. Behavioral test cases and verification logic are written for those requirements and executed against both versions. Every retained scored case must fail on the untouched initial version and pass on the frozen final version. The resulting requirements, cases, verifier, and version pair are fixed before organizational evaluation.

Members receive the starter, designated public requirements, and public checks. The final implementation, hidden acceptance tests, and held-out requirements remain evaluator-side during rollout. A verified repair or feature completion requires a retained baseline-failing check to pass on the candidate, using `baseline_status == failed and candidate_status == passed`. The frozen analysis retains the label *repair* for the W06–W10 family; this repository-based family includes both defect repair and feature development.

E.1.2 Cross-Cutting Feature Extension

Cross-cutting requirements modify public interfaces and dependencies across modules. The interface-repair affordance inserts an edit candidate for each importer when an earlier edit removes a public symbol. The parameter-threading affordance similarly supplies edit candidates when a signature gains a parameter that is not forwarded. These candidates expose the dependent implementation work through the common action surface.

The frozen manifests distinguish complete repository snapshots from construction skeletons, while component maps identify the module boundaries touched by each requirement. Cross-cutting extension demand is recorded within the repository-based and construction workload families.

E.1.3 Zero-to-One Construction

W01–W05 are zero-to-one construction workloads grounded in the functionality of existing software. For each workload, the author manually selected an existing project as a functional reference and specified the capabilities of a new, analogous program. The authoring LLM received this functional brief without the original project's source code or existing implementation. It first wrote technical documentation and an explicit feature specification, then implemented a new reference program, and subsequently wrote feature test cases and a verifier from the documented features and that completed implementation.

An unimplemented starter skeleton supplies the new program's package structure and signatures. Every retained scored case is checked against both this starter and the completed reference, requiring failure on the former and success on the latter. Requirements, starter, reference, cases, and verifier are frozen before any evaluated organization begins work. The evaluated members receive the specification, starter, and designated public checks; they do not receive the original project's implementation, the newly authored reference implementation, or evaluator-only test content.

Contracts group related behavioral cases; a complete contract requires every constituent case to pass. Mini Blobstore (W01) has five progressive contracts with seven cases each. Its qualified starter passes 0/35 cases and 0/5 contracts; its reference passes 35/35 and 5/5. All ten workloads satisfy the per-case qualification in Appendix E.5.3.

Dependencies between contracts create recurring implementation, verification, and integration demands. The workload supplies these dependencies without prescribing a particular division of labor.

E.2 Workload Sourcing and Eligibility

E.2.1 Frozen Sources and Specifications

Table 16: **Frozen main-study workload mapping.**

Frozen main-study workload mapping		
ID	**Benchmark workload**	**Frozen pack ID**
W01–W05: function-grounded zero-to-one construction		
W01	Mini Blobstore	`mini_blobstore_v1`
W02	Traffic Watch	`traffic_watch_v1`
W03	TG Automation	`tg_automation_v1`
W04	PDF Reformatter	`pdf_reformatter_v1`
W05	FastAPI Dashboard	`fastapi_dashboard_v1`
W06–W10: upstream version-transition tasks		
W06	Boltons 24.0.0 → 26.1.0	`boltons_v2400_to_v2610`
W07	Celery 5.6.0 → 5.6.3	`celery_v560_to_v563`
W08	Soup Sieve 2.6 → 2.9.1	`soupsieve_v26_to_v291`
W09	cattrs 25.1.0 → 26.1.0	`cattrs_v2510_to_v2610`
W10	Tenacity 8.2.3 → 9.1.4	`tenacity_v823_to_v914`

W01–W10 identify the same workloads in task manifests, result tables, transfer records, and case-study references. Each frozen manifest links the starter, requirements, evaluator assets, reference, qualification result, and freeze record. Original pack IDs and upstream version pairs are retained in Table 16, providing the join from paper-level workload identifiers to the frozen source artifacts.

E.2.2 Inclusion Criteria

A pack is usable only when three conditions hold and `tools/preflight_pack_environment.py --score` reports all three:

1. **the environment works** – every seeded issue names a file an agent can reach and edit, and the merge gate can actually run and refuse;
2. **the verifier works** – every retained scored case fails on the untouched starter and passes on the fixed reference, verified case by case;
3. **the gate says something** – the agent-visible acceptance gate is red on the untouched starter, so it can distinguish work from no work.

Additional structural requirements are enforced by `tools/check_pack_health.py`: every issue must resolve to an editable artifact; held-out requirements must be absent from the agent-visible stream; at least `_MIN_PUBLIC_TESTS = 3` public tests must exist; and no editable file may exceed `_MAX_EDITABLE_BYTES = 200,000`. Dependencies must be pinned in-tree, the licence must permit redistribution, and no task may require a live network.

Each retained pack includes agent-visible acceptance checks that are red on the starter and green on the reference. The retained public-gate coverage is 12/12 for W06, 8/12 for W07, 6/9 for W08, 7/12 for W09, and 9/12 for W10.

E.2.3 Exclusion Criteria

Packs are excluded for unstable evaluation, manual scoring, incompatible licensing, already-satisfied starter behavior, unavoidable future-solution leakage, irreproducible dependencies, or absent organizational demand.

E.3 Freezing and Manifest Construction

E.3.1 Starter Artifacts

For W06–W10 the starter is the complete frozen initial repository version, including its dependency lock files (`uv.lock`, `pdm.lock`, `Cargo.lock` as the project uses); the final version is reserved as the reference. For W01–W05 the starter is an unimplemented skeleton of the new program specified by the authoring LLM from a human-selected functional brief. It supplies package structure and signatures plus public contract tests, without the original project's implementation or the newly authored reference code. Both kinds declare their entry points and their agent-runnable public-test command in the manifest (`entrypoints.cli`, `entrypoints.smoke`, `public_tests.command`, `public_tests.dependencies` with exact pins).

The starter tree is identified by `starter_repo_digest`. Frozen dependency fixtures are documented in `frozen_fixture_exemptions.json`, with schema `frozen_pack_fixture_exemptions_v1`. Its 20 entries record the fixture path, pack and tree, scanner rule, SHA-256, byte count, source reference, and retention reason. The allowlist preserves the byte-identical fixtures required by the pinned dependencies.

E.3.2 Evaluator Artifacts

Each pack contains `tests/hidden/`, including the hidden suite and `specs.json` scoring-unit declaration; `issues/public/` for agent-visible requirements; `issues/heldout/` for evaluator-only requirements; and `reference_repo/` for the reference implementation. The manifest freezes scoring-unit counts, qualification outcomes, and the entry points used by the evaluator.

Hidden assets are loaded into an in-process vault keyed by a 32-byte random token. World serialization retains an `oss_evaluator_vault_binding_v1` token binding; the vault resolves the private content at evaluation time using constant-time comparison. `run_oss_hidden_tests` and `materialize_and_run_oss_hidden_tests` return aggregate fields such as `passed`, `failed`, `total`, `pass_rate`, `issue_fix`, and `issue_fix_rate`.

Qualification and candidate scoring use `build_time_machine_evaluation_plan` and `evaluate_time_machine_candidate`. Per-contract admission is resolved through `oracle_blocking_reasons`. These entry points connect the frozen cases, baseline status, and evaluated candidate state to the scoring records described in Appendix H.1.2.

E.3.3 Containers and Dependencies

The container evaluator uses a digest-pinned base image and a per-pack layer containing pinned test dependencies. The formal container path accepts `docker` or `apptainer`, requires `trust_level = untrusted` and `network_enabled = False`, and validates an image reference of the form `sha256:[0-9a-f]{64}`. The execution platform is `linux/amd64`. The repository build script creates the image, and `ORG_EVALUATOR_CONTAINER_IMAGE` supplies its concrete identifier at the execution site. A `formal_digest_pinned_image_required` status records a failed image-policy check.

Network isolation combines process-level and test-wrapper controls. `HTTP_PROXY`, `HTTPS_PROXY`, and `ALL_PROXY` point to the closed loopback endpoint `http://127.0.0.1:9`; the hidden-test wrapper restricts `socket.socket.connect` to loopback. The search system operates on its internal and frozen corpora through the separate interface in Appendix D.2.1.

The host development path evaluates an exported candidate tree in a separate process. Execution-policy and evaluator receipts identify the configuration used by each run, together with the candidate, test, and evaluator digests. Container execution and host-process execution therefore have explicit, recorded configurations for reproducing the corresponding result.

E.3.4 Versioning and Integrity Receipts

The manifest schema `org_env_oss_time_machine_pack_v2` records `starter_repo_digest`, `reference_repo_digest`, `hidden_suite_hash`, `evaluator_environment_hash`,

qualification_hash, qualification_result_hash, result_hash, runtime_test_status, agent_filesystem_isolation_status, oracle_reachability, and component_map, alongside the entry-point and test declarations. Run records carry the artifact-identity fields that associate an evaluated outcome with its frozen pack.

evaluator_environment_hash is computed from the evaluator source. A scoring-code change produces a new evaluator identity, while the recorded results retain the identity of the evaluator that produced them. The verification chain joins the dataset manifest, starter, reference, hidden suite, evaluator, tool surface, condition, event graph, and run manifest. Checkpoints additionally carry payload and sidecar digests. The recorded digest values and their frozen manifests support artifact matching during qualification, aggregation, and resume.

E.4 Task and Evaluator Construction

E.4.1 Authoring Provenance and Freeze Order

For W01–W05, authoring proceeds from the human-selected functional brief to technical documentation and feature specification, a new reference implementation, and then feature tests and verification logic. The original authoring trajectories are retained with the constructed artifacts and endpoint-qualification records. These records document how the specification, reference, and retained scoring cases were produced.

For W06–W10, the frozen initial/final repository pair anchors the feature and issue requirements and their acceptance checks. Both construction paths qualify each retained case on the starter and reference before B0–B3 rollouts. The resulting tests, references, and scoring configuration remain fixed during the evaluated organizational runs.

E.4.2 Agent-Visible Task Briefs

Briefs are rendered from frozen pack manifests. Repository-based briefs contain the product summary, public bug-fix and feature requirements, acceptance hints, entry points, and public-test command. Construction briefs contain the specification, progressive steps, skeleton layout, and public contract tests. Members also receive the artifacts and results shared during their run.

E.4.3 Exposed and Held-Out Behavioral Cases

An *exposed* unit corresponds to a requirement the organization can see: a seeded public issue, or a step of a published progressive contract. A *held-out* unit is a requirement in issues/heldout/ that is never shown to members. Its contracts are scored offline on final mainline states and saved checkpoints across all three models. In both cases the *content* of the scoring test is evaluator-only; what an exposed unit gives the organization is the requirement, plus a public acceptance check derived from it, not the hidden assertion.

The denominators depend on the scoring granularity declared by each pack, which is separate from its repair or construction label. In the frozen manifest records, contract-level scoring gives 16 units for W07, 13 for W08, 4 for W04, and 9 for W03. Other packs decompose hidden contracts into behavioural cases: W01 has 5 contracts scored in 35 cases and W02 has 9 contracts scored in 40 cases. Appendix H defines the rate aggregation for these scoring units.

E.4.4 Progressive and Complete Contracts

A complete contract requires every constituent behavioral case to pass; partial credit remains at case level. In Mini Blobstore (W01), the five progressive contracts contain seven cases each. Pre-run qualification gives 0/7 passing cases for the starter and 7/7 for the reference within every contract, yielding 0/5 and 5/5 complete contracts, respectively.

Agent outcomes are evaluated on *mainline*. Workspace results are reported separately (Appendix I.3.2).

E.5 Quality Control and Preflight

E.5.1 Starter and Reference Sanity

The preflight entry points are:

```
PYTHONPATH=. python tools/preflight_pack_environment.py --sco
  re <pack>
PYTHONPATH=. python tools/check_pack_health.py <pack>
PYTHONPATH=. python tools/print_evaluator_hashes.py <pack>
```

The first command executes the three qualification gates in Appendix E.2.2, including actual scoring of the starter and reference. The second checks issue-to-file routing, artifact resolution, oracle qualification, held-out visibility, public-suite size, and editable-file size. The third prints the evaluator-environment and qualification-plan hashes used as required batch-runner arguments.

Qualification receipts are written to `manifest.yaml` through `runtime_test_status`, `qualification_hash`, and `qualification_result_hash`, and to `provenance/authoring_report.json`. The launch preflight validates these receipts against the selected frozen workload.

E.5.2 Evaluator Determinism

Evaluator-source hashing supplies `evaluator_environment_hash`; the qualification payload supplies the plan hash. Scoring hashes the candidate tree before and after execution and raises an error on a detected mutation. The recorded tree, evaluator, and plan identities link each score to the exact evaluated state.

Malformed or erroring qualification contracts receive contract-level error statuses and are excluded by the qualification gate. CI and public-test infrastructure faults have the explicit statuses `ci_infrastructure_error` and `public_tests_infrastructure_error`. Test outcomes and execution faults consequently retain their own status fields through analysis.

E.5.3 Lower and Upper Bounds

Every retained scored case in W01–W10 is qualified against both the untouched starter and the fixed reference. The starter fails each retained case and the reference passes it, yielding 0% and 100% qualification scores on each retained acceptance suite. W01–W05 use the newly authored reference programs; W06–W10 use the frozen final upstream versions.

Qualification failure blocks admission with an explicit reason, including `baseline_not_failed:<test_id>:<status>` and `reference_not_passed:<test_id>:<status>`. Test source, references, and scoring rules are frozen before organizational rollout and versioned independently of release packaging.

E.5.4 Leakage and Identity Checks

Four automatic checks validate the member-facing interfaces: artifact inspection checks hidden and reference content, search tests inspect retrievable results, perception tests inspect rendered packets, and snapshot tests verify the aggregate-count representation. The formal world also validates evaluator-vault bindings at initialization.

A post-run future-recall check compares identifiers written by agents with identifiers found only in evaluator-side future versions. Together with the runtime access checks, these tests connect the information invariants in Table 13 to concrete artifact, search, perception, snapshot, and output records.

E.6 Completed Main Study and Evaluation Coverage

E.6.1 Main-Study Matrix

The main study contains 360 runs: ten workloads, three seeds, and B0–B3 under each of three models. GPT-5.6 Terra, Claude Opus 4.6, and DeepSeek-V4-Flash each contribute 120 runs. Endpoint evaluation and 24-step checkpoint curves cover the three-model study.

E.6.2 Analysis Populations

Analysis	Runs	Design
Main endpoints and checkpoint curves	360	Three models; ten workloads; three seeds; B0–B3.
Terra+Opus auxiliary analyses	240	Two models; ten workloads; three seeds; B0–B3.
Autonomous-protocol census	60	Terra+Opus B3 runs.

Table 17: **Analysis populations.**

E.6.3 Workload Manifest Table

Table 18: Scoring-unit inventory in the frozen W01–W10 order.

Frozen scoring-unit inventory							
Pack	**Family**	**Seeded**	**Exposed**	**Held-out**	**Contracts**	**Cases**	**Status**
W01	construction	5	35	0	5	35	main study + case
W02	construction	7	7	2	9	40	main study
W03	construction	7	7	2	9	9	main study
W04	construction	3	3	1	4	4	main study
W05	construction	3	3	1	4	4	main study
W06	repair	12	12	4	16	16	main study
W07	repair	12	12	4	16	16	main study
W08	repair	9	9	4	13	13	main study
W09	repair	12	12	4	16	16	main study
W10	repair	12	12	4	16	16	main study

Scoring-unit identity check. For W03, the manifest declares seven public issue units and two held-out issue units. Scored B2/B3 endpoint rows carry `exposed_total = 7` and `heldout_total = 2`, matching the nine contract-level behavioral cases in Table 18. This field-level join ties the exposure split to the frozen task definition.

F Experimental Conditions and Run Protocol

F.1 Exact B0–B3 Definitions

Conditions are declared as a frozen dataclass with eleven fields and resolved by condition identifier. The reported configurations are `b0_single_agent_founder`, `b1_persistent_role_org`, `b2_policy_conditioned_org`, and `b3_full_relic`. Each run retains the resolved condition and its configuration fingerprint.

F.1.1 B0: Reflective Individual

B0 uses one member with `action_selection_mode = llm_direct`. The member is constructed from the canonical eight-member roster: skill coverage is the per-domain maximum, while decision priors, communication style, and work rhythm use roster means. It has a private workspace, sandbox, and memory with the shared salience threshold and capacity limit, plus the common registered actions, tools, and task-visible information.

Operational tasks, artifacts, budget records, and repository state persist throughout the run. `institutionalization_enabled` and `capability_learning_enabled` are false. Appraised observations enter private memory under the shared salience and capacity rules; inspected or retrieved material enters the bounded context for subsequent LLM-direct decisions.

F.1.2 B1: Long-Horizon Role-Based Multi-Agent System

B1 uses eight persistent members with fixed functional roles, a shared company workspace, channels, meetings, and private member memory. The roster and member-local state persist from $t = 0$ through $t = 336$; `temporary_team = false` retains them across episode and sprint boundaries. The

model selects from the feasible candidate pool through `action_selection_mode = llm_direct`.

`profile_conditioning_enabled`, `capability_learning_enabled`, and `institutionalization_enabled` are false. Role-scoped affordances, shared artifacts, communication, and repository delivery remain available through the common execution substrate. Shared workspace and private member memory retain their respective visibility and storage interfaces.

F.1.3 B2: SDL-Based Long-Horizon Multi-Agent System

B2 uses the same eight-member team and work environment as B1, with SDL action selection (`action_selection_mode = profile_policy`), profile conditioning, and member-local skill, reputation, and authority updates.

Protocol proposal, adoption, compilation, and revision are disabled. The registered institutional actions are `propose_protocol`, `amend_protocol`, `support_protocol`, and `follow_protocol`. They may enter the candidate menu; with institutionalization disabled, their handlers return `mechanism_ablation:institutionalization`. The wish-to-proposal cadence, adoption, compilation, and registration path are skipped, while member-local learning, private memory, and shared artifacts continue to accumulate.

B1–B2 compares the combined addition of SDL, profile conditioning, and member learning. B3–B2 enables the protocol lifecycle on that shared backbone.

F.1.4 B3: Persistent Protocol-Governed Organization

B3 shares the B2 roster, model, tools, candidate construction, features, SDL parameterization, member learning, and execution handlers. Enabling `institutionalization_enabled` activates reflection-to-wish, proposal, validation, governance, and protocol compilation. Adopted protocols enter R_t and Q_t, supplying decision features, responsibility signals, and post-handler protocol checks and records. Members can amend and retire adopted rules through the same lifecycle.

F.2 Roster, Functions, and Decision Priors

F.2.1 Roster Size and Functional Specialties

The canonical roster is eight members, used in full by B1, B2 and B3, and aggregated into one member for B0. The functional specialties are a leadership and direction function, two implementation functions differing in speed-versus-care emphasis, a reliability and verification function, and four further functions covering review, documentation, customer signal and coordination.

F.2.2 Roles, Responsibilities, and Permissions

The board starts with unassigned tasks, and members allocate ownership during execution. Role-scoped affordances determine which functions receive edit and review candidates and which may approve releases; these affordances are identical across B1, B2, and B3. Release approval requires a lead role and the reliability function, capped by roster size for multi-member conditions. Object permissions follow Appendix D.2, and escalation combines the registered escalation action with the governance deadlock rule.

The declared-factor conformance test checks that all multi-member conditions use the same governance role topology. The comparison therefore retains the same functional approval structure while varying the declared mechanism switches.

F.2.3 Functional Decision Priors

A prior $p_i(t)$ is a member-specific map of named, real-valued profile and skill dimensions initialized from the canonical roster. It enters the 32-dimensional SDL representation in Appendix C.3.1. The reported assignment is `aligned`, with functional conditioning fixed before the runs and shared by B2 and B3.

Profile-backed inputs remain fixed, while skill-backed inputs evolve through the common member-learning path. The `profile_conditioning_enabled` switch activates these prior inputs in B2 and B3. The same roster initialization and profile assignments are used for the two configurations.

F.2.4 MEMBER INITIALIZATION

Every member starts with: empty private memory; an empty personal workspace and an empty sandbox; no owned tasks; default continuous condition (`attention` 1.0, `morale` 0.6, `trust_in_company` 0.7, `perceived_recognition` 0.5, `role_clarity` 0.4, the remaining variables 0); the roster's literal prior and skill maps; and eight-domain reputation and authority at their configured baselines. Task knowledge is whatever the brief and the starter tree contain (Appendix E.4.2); nothing about the hidden suite, the reference or the held-out requirements is present in any member's initial state.

Randomization at initialization is limited to availability registration, which is seeded. Under fresh-member turnover, the same initialization procedure reconstructs member-local state for the target organization while preserving the functional role slots. A fresh member therefore starts with the same kind of local state as a member at $t = 0$ of an ordinary run (Appendix G.2.1).

F.3 PROMPTS AND INFORMATION CONTROLS

Relic separates selection of *what to do* from model calls used to carry out or reflect on a selected activity. B0 and B1 ask an LLM to choose one action from a constrained candidate menu. B2, B3, B3-text, Transfer Text, and Transfer Exec use SDL for what-action selection and issue no LLM action-selection prompt. Other model calls remain module-specific, including code/document editing, reflection, proposal drafting, and proposal evaluation.

Prompt stage	B0	B1	B2	B3	B3-text	Text	Exec
Shared grounding / identity	cond.	cond.	cond.	cond.	cond.	cond.	cond.
What-action selection	LLM	LLM	SDL	SDL	SDL	SDL	SDL
Code/document editing	cond.	cond.	cond.	cond.	cond.	cond.	cond.
Reflection	cond.	cond.	cond.	cond.	cond.	cond.	cond.
Separate LLM wish extraction	—	—	—	—	—	—	—
Wish → proposal	—	—	—	cond.	cond.	—	—
Proposal evaluation	—	—	—	cond.	cond.	—	—
Approval-specific LLM call	—	—	—	—	—	—	—
Readable rules in code-editor context	—	—	—	avail.	avail.	6	6

Table 19: Prompt routing across conditions. "cond." denotes a module call when that activity occurs.

The wish stage maps and filters reflection-generated `improvement_ideas`. Proposal evaluation supplies scores and revisions; governance actions supply approval and adoption.

F.3.1 SHARED SYSTEM AND GROUNDING TEMPLATE

Cognitive modules compose the system message from a shared workload/company brief, global grounding rules, the module name, and the module-specific instruction. The member identity block—role mandate, enabled functional conditioning, and current memory/context—is prepended to the user message. The grounding rules require the model to use supplied objects and identifiers, select only available actions and valid targets where applicable, avoid inventing outcomes, and return the requested structured output.

F.3.2 B0/B1 ACTION-SELECTION PROMPT

For B0/B1, the user message presents the member-visible action context and asks for one exact candidate:

```
Choose the next intentional action for {agent_id} at tick
{tick}.
{rendered_action_context}
Return ONLY JSON matching the action schema;
```

```
candidate_action and target must identify one exact entry
from candidate_options.
```

The returned choice is validated against the constrained candidate pool before execution. B2/B3 and all text/transfer variants do not use this prompt because SDL performs the corresponding selection.

F.3.3 Code and Document Editing

After an edit action has been selected, the code editor receives the target file, current content, available public evidence, the edit goal, and any readable organizational rules applicable to the treatment. The editor returns anchored replacements rather than a whole-file rewrite. In B3/B3-text and both transfer treatments, readable rules enter the model through a dedicated block:

```
HOW THIS ORGANIZATION WORKS:
- {readable rule 1}
- {readable rule 2}
...
```

F.3.4 Formation Prompts: Reflection, Wishes, and Proposals

Reflection receives recent work, failures, relevant episodes, product context, memory, and existing protocol context and asks for structured self/team assessment, blockers, and concrete `improvement_ideas`. A `protocol_need` is used when recurring avoidable failure suggests a standing, checkable rule. The main formation path then maps these ideas into wishes; no separate LLM wish-extraction call is used.

A qualifying wish is drafted into a proposal with the visible wish record, available registered actions, and product context. The proposal instruction requires a concrete checkable requirement, names required approvals, and does not allow the model to assume adoption. Proposal evaluation separately returns feasibility, usefulness, risk, adoption score, blocking issues, and suggested revision. Recurring-pattern synthesis can also produce a candidate protocol; it remains a proposal until the governance path adopts it.

The exact prompt text, readable transfer rules, and wish-to-protocol measurement criteria are given in Appendix K.7.1.

F.3.5 Context Rendering

Rendering is section-ordered and budget-aware. Candidate menus, task boards, inboxes, pull requests, test results, and product context retain the information needed to act; narrative history/reflection sections use bounded rendering. Inbox triage and private-memory capacity provide additional shared limits across conditions.

F.4 Matched Experimental Controls

F.4.1 Model and Decoding Configuration

The main study uses GPT-5.6 Terra, Claude Opus 4.6, and DeepSeek-V4-Flash, with 120 runs per model. The DeepSeek runs use the provider model identifier `deepseek-ai/DeepSeek-V4-Flash-0731`. GPT-5.6 Terra response identifiers include `gpt-5.6-terra-2026-07-09` and the unversioned family name. Internal transfer uses GPT-5.6 Terra; ProgramBench uses GPT-5.6 with `xhigh` reasoning effort. The in-situ binding ablation uses Claude Opus 4.6. The same-model CooperBench comparison uses Claude Opus 4.6 with high reasoning effort.

Client transport documentation specifies a `chat_completions` route, a 180-second request deadline, six retries, and exponential backoff starting at three seconds. Connect, read, write, and pool timeouts have separate configuration fields. Each run retains its provider/endpoint routing, wire API, output limits, decoding fields, and request policy in the client configuration record. Reasoning effort and temperature are separately identified fields.

A model-call failure propagates as a failed decision or enters the module-specific fallback path. `llm_runtime_identity` records `observed_response_models` as a counter of returned model

strings, an endpoint hash, a routing-context fingerprint, and a chained response-ID digest. `llm_runtime_fingerprint` hashes that identity block. Requested model identifiers, observed response identifiers, and routing metadata thus remain directly inspectable for each recorded configuration.

F.4.2 Tools and Registered Actions

One action registry and wiring function install the same candidate mapper, feature extractor, policy implementation, and execution adapters across the conditions. Condition switches choose LLM-direct or SDL action selection and activate member learning and institutionalization. The run record's `tool_surface_fingerprint` identifies the offered tool and action surface used by that configuration.

Protocol and tool creation pass through B3 governance. The four institutional actions are named in Appendix F.1.3; their registry presence and condition-dependent execution are checked independently of the common repository-delivery actions.

F.4.3 Task-Visible Information

The pack and seed fix the starter tree, public requirements or specification, public-test command, entry points, and issue timing. Public-test outcomes are computed on each condition's candidate state. The analysis joins paired conditions through the manifest, starter, hidden-suite, and evaluator identities retained in the run and evaluator records.

The aggregation pipeline validates starter digests before pooling paired conditions. A revised public gate or evaluator produces a versioned asset identity, and the analysis maps each outcome to its designated frozen task configuration. This join preserves the correspondence among the task, candidate artifact, scoring procedure, and reported comparison.

F.4.4 Execution Horizon and Resource Ceilings

The main-study matrix uses a 336-step horizon, checkpoints every 24 steps, 168-step sprints, and disabled work rhythm. Token expenditure is measured per run, with no matched hard token, model-call, or primary-action ceiling. Company-balance accounting is soft: charging clamps the remaining balance and records violation flags, while action execution remains governed by the action and repository validity checks.

The runner also supports a company-wide resource-ledger configuration with `max_llm_calls`, `max_llm_requested_tokens`, `max_llm_prompt_characters`, `max_primary_actions`, and `max_ticks`. When enabled, its append-only usage and exhaustion signal operate at organization scope. The run configuration records the active horizon and resource policy; the reported main-study results use the scheduling-horizon setting above.

F.4.5 Seeds and Randomness

Each model–workload–condition combination uses seeds 1401, 2711, and 4013. The mechanism case uses seed 1401.

Randomness is namespaced from the run seed rather than drawn from one global stream: policy jitter and softmax sampling use `Random(f"{seed}:{tick}:{agent}")`; availability registration, sandbox timing, the simulated community and substrate controls each derive their own stream; and the shuffled-prior transform uses a prefixed string seed. Model calls use the configured provider sampling. Within each step, members are served in roster insertion order.

F.5 Run Protocol

F.5.1 Initialization and Preflight

Before launch, the pack passes the three qualification gates with `--score`. The evaluator and qualification-plan hashes are printed and supplied as required runner arguments. The code tree is copied into a run-specific frozen directory, and the public suite is executed on both starter and reference to verify red/green acceptance behavior. The runner then initializes organizational state from the selected manifest.

Evaluator assets are loaded into the token-keyed vault, with the binding stored in world state. When prompt auditing is enabled, hashing starts at the first intercepted call and the audit receipt records its call coverage. The initialized run therefore retains the code, task, evaluator, and information-interface identities used by its subsequent execution.

F.5.2 Online Execution

Each step: advance the clock; at a day boundary run the daily passes (growth where enabled, budget, structural detectors); process meetings and background jobs; then for each member in roster order, triage the inbox, build perception, construct and guard candidates, select, execute, appraise into memory, and write graph edges. Governance passes run on their own cadences (Algorithm A2). State is written to a checkpoint every 24 steps together with token totals, recorded at the same checkpoint.

F.5.3 Checkpoint and Endpoint Grading

Checkpoint scoring evaluates saved states offline with a 120-second per-contract timeout. Endpoint grading evaluates final mainline state. The reported endpoint and 24-step checkpoint series cover the three-model, 360-run study, with the tree type, scoring units, and metric eligibility attached to each evaluated series.

Each checkpoint couples the saved organization and candidate state with its token totals and evaluated outcomes. Figure 4 uses these recorded checkpoint steps; the cost sensitivity selects saved checkpoints according to Appendix I.7.3.

F.5.4 Stopping, Submission, and Resume

Runs end at the scheduling horizon. Interrupted execution resumes from the last saved checkpoint under the same run identity. Appendix K.6.2 specifies provider retries and recovery.

F.5.5 Run Matrix and Inventory

Table 20: Experimental accounting.

Experimental accounting			
Evidence line	**Runs**	**Design / coverage**	**Configuration**
GPT-5.6 Terra main study	120	$10 \times 3 \times 4$	Ten workloads, three seeds, B0–B3.
Claude Opus 4.6 main study	120	$10 \times 3 \times 4$	Ten workloads, three seeds, B0–B3.
DeepSeek-V4-Flash main study	120	$10 \times 3 \times 4$	Ten workloads, three seeds, B0–B3; included in the pooled endpoint table.
Internal transfer	60	30 Text + 30 Exec	Ten workloads $\times$ three seeds per arm; 30 GPT-5.6 Terra B2 main-study references reused.
In-situ binding ablation	30 new	B3-text; Claude Opus 4.6, seeds 1401/2711/4013	Thirty matched B3 references reused from the Claude Opus 4.6 main study.
Selected case	included	One matched GPT-5.6 Terra main-study unit	W01; GPT-5.6 Terra; seed 1401.
ProgramBench	separate	Same 25 tasks, two systems	External protocol plug-in; no SDL.
CooperBench	separate	Full 652-pair benchmark + fixed 48-pair subset	Two-member Relic; full released-reference comparison plus same-model Solo/Peer control.
Human–agent extension	walk-through study	P2/P3 walkthroughs	Human-facing interface evaluation.

F.6 Condition-Conformance Tests

F.6.1 Mechanism-Isolation Tests

The declared-factor test enumerates the fields allowed to change at each B0–B3 step and compares them with the resolved condition definitions. For B2–B3, the allowed mechanism switch is `institutionalization_enabled`. The shared wiring installs the candidate generator, feature extractor, decision layer, execution operator, registry, model, and member-information interfaces for each condition.

An action-reachability test checks the offered candidates for the delivery chain: edit, commit, open, review, CI, merge, and release. It checks the chain in both B0 and B3. When the candidate-pool capacity limit binds, delivery actions are retained first. The conformance tests inspect this pool-priority rule and the shared governance role topology alongside the condition switches.

F.6.2 Configuration, Horizon, and Information Matching

The four conditions for a workload and seed are launched from one plan that fixes workload, horizon, seed, and model. Run records retain `dataset_manifest_hash`, `starter_repo_digest`, `hidden_suite_hash`, `evaluator_environment_hash`, `tool_surface_fingerprint`, `ablation_fingerprint`, `llm_runtime_fingerprint`, `seed`, and `action_selection_mode`. These fields bind results to the resolved configuration and support the workload/version/eligibility joins used by paired aggregation.

F.6.3 Member-Information and Access Tests

Access tests cover evaluator-private assets, future repository history, private member workspaces, and live-network access. Artifact, search, perception, and snapshot checks are specified in Appendix E.5.4. The optional prompt-visibility interceptor records the intercepted calls, and the future-recall checker inspects generated identifiers. The execution-layer proxy and socket controls in Appendix E.3.3 complement these member-facing checks.

G Internal Protocol Transfer and Executable Binding

The internal-transfer experiment uses 30 Text and 30 Exec GPT-5.6 Terra target runs over ten workloads and three seeds. Fresh uses the corresponding 30 B2 main-study runs. Text and Exec share a source-derived frozen rule package and vary its executable binding.

G.1 Protocol Source and Bundle Freezing

G.1.1 Source-Run Eligibility

The reported transfer package is constructed from B3-formed protocols using recurring capability families observed across source runs as the selection frame rather than target performance. From these families we selected functionally distinct representative concerns and normalized them into four source concerns and six closed-set typed runtime guards. This selection did not use target-evaluation outcomes. Normalization was limited to making rule semantics mechanically executable in the target runtime: timed, multi-head, or otherwise unobservable clauses were removed or rewritten; triggers, required fields/steps, affected actions, violation conditions, reason codes, and machine bindings were made explicit. Compatibility checks verified trigger observability and action reachability. Table 21 shows the public functional mapping.

The final package uses bundle schema `org_capability_bundle_v2` and binding schema `org_protocol_bindings_v2`. It was frozen before the reported target evaluation. Text and Exec use exactly the same frozen readable content; the treatment contrast is the presence or absence of executable bindings.

Table 21: Public functional mapping of four source-protocol concerns to six compiled runtime guards.

Transferred protocol-to-guard mapping

Source concern	Compiled guard	Affected actions	Functional role
S1	issue owner assigned	`open_pr`	customer triage / ownership
S1	branch owned by actor	`open_pr`	ownership map
S2	unchanged failed CI not retried	`run_ci`, `ci_test`	evidence workflow
S2 + S3	current CI attested	`merge_pr`	workflow integration
S3	independent review	`review_pr`, `approve_pr`, `formal_pr_review`, `merge_pr`	review gate
S4	release gate covered	`publish_product_release`	release governance

Notes. S1–S4 identify four source concerns; the CI-attestation guard combines S2 and S3.

G.1.2 Package Provenance and Freeze Record

The transfer record identifies the package version, content digest, selected source items, normalization steps, and freeze time. A source-checkpoint contribution is linked to the corresponding item. The final freeze fixes the six compiled guards and their readable summaries before target evaluation, and Text and Exec reference the same package identity.

Source-selection metadata, source IDs, and normalization records are retained with the frozen package, connecting each compiled guard to the source concern in Table 21.

G.1.3 Target-Workload Selection

Targets use the W01–W10 starters, public requirements, and evaluators. Text and Exec are matched through their target/workload/seed/configuration mapping, yielding 30 matched units. Fresh uses the corresponding 30-run GPT-5.6 Terra B2 stratum. Seed rates are averaged within workloads, and applicable workloads receive equal weight.

Curation includes a reverse-reachability check: required verification steps are mapped back to agent-reachable artifacts and public issues before task admission. This check validates that the target action surface supplies the work needed to satisfy the protocol's verification obligations. Source-to-target relations and scoring identities remain linked to the package and the target record.

G.2 Fresh-Member Initialization

G.2.1 Fresh Roster

The target organization is freshly initialized from the canonical eight-role roster. Member-local state is reset: private memory, personal workspace contents, sandbox cache, message read/acknowledgement state, commitments, reflections, wishes, and accumulated member-local experience are not inherited from the source run. Functional roles are reinstantiated for the target task, while the transferred organizational package is applied according to treatment.

G.3 Organizational-State Arms

G.3.1 Exec: Executable Protocol Binding

Exec initializes fresh target members and supplies the frozen package as readable rules with six live executable bindings. The target runtime receives the triggers, scopes, affected actions, required steps and evidence, responsibility functions, exceptions, and retirement conditions. Registration and identifier remapping associate subsequent runtime events with the imported rule and package.

Readable content is identical to Text. During target execution, existing bindings produce use and consequence records while the package remains frozen under the treatment in Appendix G.3.4.

G.3.2 TEXT: CONTENT-MATCHED PROTOCOL PRESENTATION

Text initializes fresh target members and presents the same six ordered rule summaries to model-mediated work, including the code editor, through `HOW THIS ORGANIZATION WORKS`. The prose preserves each rule's trigger, scope, required steps and evidence, responsible functions, exceptions, and consequences. The representation uses `capability_as_prose` and the `TEXT_ONLY_DOC_TYPE` carrier.

Text retains SDL and member learning with zero compiled bindings for the transferred package. The readable summaries and their order are checked against the same frozen package as Exec; the verbatim content is reproduced in Appendix K.7.1.

G.3.3 FRESH REFERENCE: REUSED B2 MAIN-STUDY RUNS

Fresh uses the 30 GPT-5.6 Terra B2 main-study runs with ten workloads and three seeds, initialized without an inherited protocol package.

G.3.4 FROZEN FORMATION AND THE B2 REFERENCE

The treatment freezes the mechanism-producing path for the entire target window: no new protocol proposals, adoption of additional protocols, or learned rule revisions. Ordinary code edits, tests, communication, local memory, and the B2 member-learning paths continue. Imported executable bindings and their use/consequence records remain active while formation is frozen.

Table 22: Why the main-study GPT-5.6 Terra B2 condition is the fresh-start reference.

Transfer treatment freeze matrix			
Target component	**Fresh reference**	**Text**	**Exec**
B2 backbone: SDL and member learning	same	same	same
New protocol proposal / adoption / revision	off	off	off
Fixed transferred protocol content	absent	present	present
Transferred executable bindings	absent	absent	present
Origin of counted runs	main study	target run	target run

G.4 TRANSFER PAYLOAD

G.4.1 PERMITTED PAYLOAD

Only organizational state may cross: executable rules with their `trigger_condition`, `scope`, `affected_actions`, `required_steps`, `required_fields`, `enforcement_rule`, `enforcement_action`, `exception_rule` and `sunset_rule`; their responsibility mappings over functions; and the carrier documents those rules reference. The bundle is schema-versioned and validated on both export and import.

G.4.2 EXCLUDED PAYLOAD

Excluded by construction: source code and patches from the source workload; any target patch or solution; private member memory; hidden-test outcomes and any evaluator record; task-specific answers; and future repository history.

G.4.3 PAYLOAD AND RESET RECORDS

Each transfer record contains a `capability_transfer` block with `roster_origin`, `capability_form`, `source_repository_id`, and `protocols_injected`. `experiment_phase` is `capability_transfer`, and `pack` identifies the source-to-target relation. Receipt validation checks the required fields, package contents, and source/target relation against the selected evaluation plan.

The record stores treatment identity, package-item and carrier-document counts, package digest, and reset state. A source-to-target role map binds obligations to target functions. Fresh initialization

reconstructs member memory, workspaces, sandboxes, messages, commitments, reflections, and wishes while retaining the designated external protocol package. Runtime events refer to the injected package and rule identifiers, providing the join from inherited organizational state to later target actions.

G.5 Content Matching and Binding Control

G.5.1 Content-Matched Prose Construction

Text and Exec use the same `canonical_v2` package. The six ordered summaries render to an identical 3,148-character code-editor block in both treatments. Exec activates six compiled bindings; Text activates none. Appendix K.7.1 provides the shared readable text and package identifiers.

G.5.2 Execution-Hook Removal

Exec registers the transferred package's `affected_actions` matchers, `responsible_roles` routing hooks, and execution checks. Text retains the corresponding prose in shared work state with zero compiled bindings. Both treatments keep the target-stage proposal, adoption, and revision path disabled.

Conformance checks compare package identities, readable fields, and rendering order across Text and Exec and inspect the presence of compiled bindings in Exec and their absence in Text. Existing-binding use, violation, and execution events are recorded separately from protocol creation and revision, preserving the frozen target treatment throughout the observation window.

G.6 Transfer Estimands and Evaluation Scope

G.6.1 Primary Controlled Comparisons

Table 23: **Internal-transfer comparisons.**

Internal transfer: controlled contrasts			
Contrast	**Reference**	**Pass-rate effect**	**Interpretation**
Exec–Text	Actual target treatments	**+6.5 pp**	Executable binding of content-matched protocols.
Exec–Fresh	Reused GPT-5.6 Terra B2 equivalent	**+15.8 pp**	Comparison with the designated no-package reference.

G.6.2 Interpretation

Exec–Text estimates the gain from executable binding of the content-matched source-derived package. Exec–Fresh measures the gain over fresh B2 initialization.

H Metrics and Statistical Analysis

H.1 Canonical Data Sources

H.1.1 Repository State and Action Log

Product and delivery outcomes use the repository objects in `world.repo_system.repo`, the product artifacts, and evaluator outputs for the corresponding candidate state. The action log records explicit action occurrences. Protocol lifecycle measures use the protocol ledger and proposal histories. These source interfaces retain the object identifiers needed to join a reported count to its underlying records.

The selected case labels repository and action counters separately and specifies the runtime paths connecting them in Table 48.

H.1.2 Evaluator Outputs

The final-evaluation artifact uses schema `orgenv_oss_final_evaluation_v1`. Each contract or check records `task_id`, `command`, `owner`, `kind`, `base_status`, `candidate_status`, `required`, `baseline_repo_digest`, `candidate_repo_digest`, `execution_policy_hash`, `exit_code`, `stdout_hash`, and `stderr_hash`.

Passing cases satisfy `candidate_status == passed`; a confirmed fix additionally satisfies `base_status == failed`. Checkpoint evaluation uses the same scoring fields at each saved step across the current 360-run study. Infrastructure and malformed-evaluation errors retain explicit statuses and enter the metric-eligibility rules in Appendix H.3.3.

H.1.3 Organizational-State and Protocol Records

Protocol measures join proposal-manager objects and their status histories with the protocol event ledger. Adoption, use, violation, enforcement, amendment, and retirement each have their own event type. Per-spec scalar counters provide a cross-check against the corresponding ledger totals.

Distinct affected targets are reconstructed from typed governed-object references in enforcement events. The references identify PRs, patches, commits, and other work objects, and link the object-level consequences to the protocol lineage that generated them.

H.1.4 Token and Compute Receipts

Provider receipts accumulate per call in `usage_totals` and are written to the run-level `llm_usage` record, including retry calls. Input, output, and cached-token counters are retained where supplied by the provider. Checkpoint token totals are stored with the corresponding saved state for the matched-spend analysis.

Run duration is recorded in seconds. Provider-token accounting and elapsed time are separate measurements; tool and container compute follows the resource-accounting definition in Appendix D.5.

H.2 Metric Dictionary

Table 24: Metric dictionary.

Metric dictionary

Metric	Numerator	Denominator	Better	Applicability
Verified production (evaluator, mainline)				
Evaluator-confirmed seeded issues	issues whose contract failed on starter and passes on candidate	seeded issues in pack	high	all packs
Exposed cases on mainline	exposed units passing	exposed units	high	packs separating exposed from held-out
Held-out cases on mainline	held-out units passing	held-out units	high	packs with a held-out set
Complete contracts on mainline	hidden contracts fully passing	hidden contracts	high	all packs
Workspace and delivery				
Workspace contracts	contracts passing on the workspace tree	hidden contracts	high	packs with a workspace series
Workspace exposed cases	exposed units passing on workspace tree	exposed units	high	idem
Patch acceptance	accepted patches	generated patches	high	all runs
Accepted work reaching mainline	accepted patches present in a merged commit	accepted patches	high	all runs with patches
PR merge rate	merged pull requests	opened pull requests	high	runs that opened a PR
Releases	release objects published	— (count)	n/a	all runs
Organizational mechanism				
Currently adopted protocols	registry lineages with terminal `adoption_status=adopted`	— (count)	n/a	all runs
Ever-adopted protocol lineages	distinct run–protocol lineages with an adoption record, including later obsolete rules	— (count)	n/a	runs with lifecycle records
Terminal protocol-registry objects	all registry objects, with adopted, proposed, and obsolete states counted separately	— (count)	n/a	runs with a terminal registry

Continued on next page.

Table 24 (continued)

Metric dictionary

Metric	Numerator	Denominator	Better	Applicability
Mechanism uses	recorded `use` events, evidence level labelled	— (count)	n/a	all runs
Runtime protocol enforcements	exported protocol-linked enforcement events; scope stated with the count	— (count)	n/a	runs with the reported counter
Distinct affected targets	distinct governed objects in enforcement events	— (count)	n/a	bundles with references
Efficiency				
Tokens per task	total tokens summed over tasks	tasks	low	all runs
Tokens per confirmed fix	block-level token total	block-level confirmed-fix count	low	positive-fix model–workload blocks; macro averaged
Actions per outcome	total actions	confirmed fixes	low	idem
Diagnostic				
Seeded completion (declared)	tasks the board marks complete	seeded issues	—	all packs
Completion credibility	declared issues also evaluator-confirmed	declared-complete issues	high	common eligible issue set; positive declarations

H.2.1 Verified Production Metrics

The four mainline production rates in Table 24 are computed from frozen evaluator records. Confirmed seeded issues pair a baseline-failing contract with a passing candidate status. Exposed and held-out rates use their own requirement sets and scoring units. A complete contract requires all constituent behavioral cases to pass, with partial credit retained at case level.

Each numerator is computed on the evaluated mainline artifact and paired with the corresponding frozen denominator. This procedure ties the reported production endpoints to both the starter behavior and the final integrated code.

H.2.2 Workspace and Delivery Metrics

Workspace metrics score local candidate trees; mainline metrics score the integrated shared repository. Delivery accounting follows generated patches, accepted patches, accepted patches contained in merged commits, opened pull requests, and merged pull requests. Repository objects supply these stages, while explicit action occurrences remain separate log counters.

Workspace/mainline comparisons align run identity, checkpoint or endpoint, and scoring units. Latency measures use recorded event or checkpoint times. The selected case additionally joins PR identities to their opening, verification, blocking, and merge records.

H.2.3 Organizational-Mechanism Metrics

Protocol measures include proposal, adoption, registration, activation, action matching, reading, use, enforcement, amendment, and retirement. Each event retains its protocol identity and time. The generic `use` event credits a successful action matching the protocol's `affected_actions`; decision and execution effects have their own linked records.

The census identifies autonomous, seeded, and imported rules by origin. Revisions are grouped into the parent within-run lineage. Persistence is measured from a named lifecycle origin, such as adoption or activation, to the last qualifying use or recorded active step. Formation applies the unit, step, use-count, and 48-step criterion in Appendix C.4.3; the strong evidence grade joins third-party execution, governed-object changes, and evaluator outcomes.

H.2.4 Efficiency Metrics

Mean tokens per run averages provider-token totals over contributing runs. For cost per confirmed issue, tokens and confirmed fixes are pooled across the contributing seeds within each model–workload block. The block cost is its token total divided by its confirmed-fix total. Zero-success runs retain their expenditure within the block, and blocks with positive confirmed-fix counts supply the eligible block ratios.

Arm estimates macro-average their eligible block ratios: B0–B3 use 18, 21, 21, and 22 blocks, respectively. The paired B3–B2 contrast computes both conditions on the same 20 common blocks before aggregation. Table 38 retains the Terra+Opus auxiliary calculation on its 14 common blocks. Internal-transfer costs use the separate transfer analysis associated with Table 2.

H.2.5 Diagnostic Metrics

Completion credibility is the fraction of declared-complete issues confirmed by the evaluator.

H.3 Aggregation, Coverage, and Missingness

H.3.1 Metric-Specific Pooling

For each run r and metric k, the analysis records numerator n_{rk}, denominator d_{rk}, applicability, and evaluation status. It first computes the eligible run-level rate n_{rk}/d_{rk}, averages the three seeds within each model–workload–condition block, and then equally weights applicable model–workload blocks across the three-model study. The Terra+Opus auxiliary analyses use the same block-macro procedure on their two-model population.

The numerator and denominator travel together with the metric's scoring unit and contributing-run set. Measured zero outcomes retain their zero value; inapplicable or undefined rates retain their NA status. Paired contrasts apply the same metric-eligibility and workload/version mapping to both conditions before recomputing the aggregate.

H.3.2 Checkpoint and Endpoint Coverage

The reported checkpoint curves and main endpoints cover the 360-run three-model study. Curves use the 24-step grid, with tree type and metric-specific scoring units fixed for each series.

H.3.3 Missing and Inapplicable Metrics

The analysis retains metric applicability, denominator validity, evaluator status, run completion, and version eligibility as separate fields. NA denotes an inapplicable or undefined rate, including held-out metrics for workloads without a held-out set and outcome-normalized ratios with a zero denominator. A measured failure contributes zero to the corresponding pass numerator while retaining its valid denominator.

Infrastructure and malformed-evaluation errors retain explicit error statuses. The recorded eligibility rule determines which metric rows enter the paired calculation, and both conditions use the same rule. Run and version records preserve the reason associated with an excluded or superseded observation.

H.3.4 Exact Count Records

Each reported metric is associated with its raw numerator, denominator, scoring unit, contributing-run count, and analysis-population identifiers. The aggregation records retain the workload and model weights used to produce the displayed rates. These fields provide the numerical inputs for recomputing the table entries and matching an aggregate back to its constituent run and evaluator records.

H.4 Estimands

H.4.1 Primary Organization Contrast

The primary contrast is B3–B2 within matched model–workload–seed units for verified production and resource outcomes. The design contains 30 matched units per model, or 90 across the three models.

H.4.2 Secondary Organization Contrasts

B2 minus B1 estimates the SDL-based configuration bundle, including selection mode, profile conditioning, and member learning (Appendix F.1.3); B1 minus B0 estimates having long-horizon

multi-agent collaboration under the documented condition definitions. All three models cover the same ten-workload, three-seed, four-condition design. Both are reported as secondary ladder diagnostics.

H.4.3 Transfer Estimands

Exec–Text uses 30 matched GPT-5.6 Terra workload–seed units. Exec–Fresh uses the corresponding B2 main-study runs. Seed rates are averaged within workloads, then applicable workloads receive equal weight. Held-out outcomes use W02–W10, or 27 observations per treatment.

H.4.4 Heterogeneity Estimands

Heterogeneity is summarized by workload, workload family, seed, and model. Model-stratified endpoints cover all three models. Workload/family and leave-one-workload-out summaries use Terra+Opus, with run-level rates macro-averaged over the corresponding blocks.

H.5 Statistical Analysis and Supplementary Checks

H.5.1 Matched Effects and Clustering

The paired main-study unit is model–workload–seed. Bootstrap resampling holds the model–workload blocks fixed and samples seeds within each block. For a treatment contrast, corresponding seeds are resampled as pairs, and each draw recomputes the same block-macro estimate used by the endpoint table. Arm intervals recompute their respective arm estimates.

This procedure preserves the shared starter, requirement set, and evaluator associated with each workload while retaining the paired treatment structure. Population-specific resampling settings are given below.

H.5.2 Confidence Intervals and Bootstrap

The production endpoints in Table 1 use 50,000 bootstrap draws. Each arm interval recomputes that arm's block-macro estimate; the B3–B2 interval resamples matched seeds within each fixed model–workload block.

The three-model outcome-normalized-cost analysis uses 20,000 draws. Arm intervals use the respective eligible blocks, and the cost contrast resamples matched seeds within the 20 common B2–B3 blocks. Terra+Opus workload/family, leave-one-workload-out, and budget-capped analyses use 10,000 draws with random seed 1729.

Internal-transfer intervals come from the transfer analysis associated with Table 2. The in-situ complete-contract interval uses 200,000 matched-seed draws within its ten workloads; Table 43 reports paired 95% intervals for all four in-situ endpoints.

H.5.3 Multiplicity

The complete-contract B3–B2 contrast is primary; secondary endpoints are reported with pointwise 95% intervals.

I Full Results and Robustness Checks

I.1 Run Inventory

I.1.1 Main-Study Coverage

The main study contains 120 runs per model and 30 runs per condition within each model. Table 20 summarizes the experiment inventory.

I.1.2 Transfer Runs

Table 25 summarizes the transfer treatments and reused Fresh reference.

Table 25: Internal-transfer treatment accounting.

Internal-transfer treatment accounting		
Condition	**Run accounting**	**State and origin**
Text	30 new runs	Ten workloads × three seeds; fresh members and readable protocol content.
Exec	30 new runs	The same ten-workload, three-seed design; fresh members and content-matched executable package.
Fresh reference	30 reused runs	GPT-5.6 Terra B2 main-study reference.

I.1.3 Case-Diagnostic Runs

The mechanism case uses W01, GPT-5.6 Terra, seed 1401, and the 336-step horizon. Appendix J gives the protocol and delivery timeline.

I.2 Workload, Seed, and Family Outcomes

I.2.1 Per-Workload Outcomes

Table 26 reports Terra+Opus workload-level B3–B2 effects using the block-macro aggregation in Appendix H.3.1.

Table 26: **Per-workload mainline B3–B2 point estimates.**

Per-workload verified production: B3–B2			
Workload	**Complete contracts**	**Confirmed seeded issues**	**Exposed cases**
W01	−6.67 pp	−6.67 pp	−6.19 pp
W02	+5.56 pp	+4.76 pp	+4.76 pp
W03	+14.81 pp	+19.05 pp	+16.67 pp
W04	+16.67 pp	+16.67 pp	+16.67 pp
W05	0.00 pp	0.00 pp	0.00 pp
W06	+6.25 pp	+8.33 pp	+8.33 pp
W07	+13.54 pp	+18.06 pp	+18.06 pp
W08	+7.69 pp	+9.26 pp	+9.26 pp
W09	+1.04 pp	+1.39 pp	+1.39 pp
W10	+6.25 pp	+8.33 pp	+8.33 pp

I.2.2 Repair and Construction Split

Table 27: **Construction and repository-based workloads reported separately.**

Verified production by workload family					
Metric	**B0**	**B1**	**B2**	**B3**	**B3–B2**
Construction					
Complete contracts on mainline	6.92%	8.10%	9.25%	**15.32%**	**+6.07 pp**
Evaluator-confirmed seeded issues	8.37%	7.70%	10.30%	**17.06%**	**+6.76 pp**
Repair					
Complete contracts on mainline	13.69%	28.32%	23.37%	**30.32%**	**+6.95 pp**
Evaluator-confirmed seeded issues	18.33%	38.24%	31.39%	**40.46%**	**+9.07 pp**

Notes. Repository-based workloads W06–W10 include bug fixes and feature development.

I.3 Full Endpoint and Time-Series Results

I.3.1 Checkpoint Curves

Figure 4 pools checkpoint trajectories and endpoints across the three-model, 360-run study. Series are evaluated on the 24-step grid and rendered as step-held curves between checkpoints.

I.3.2 Workspace and Mainline Outcomes

Figure 4 presents workspace and mainline outcomes. Appendix J.3 traces repository delivery through seven 48-step windows in the mechanism case.

I.3.3 Declared Completion and Evaluator Confirmation

Figure 4a compares declared completion and evaluator confirmation across all three models. Dashed and solid curves use the same checkpoint grid and eligible issue set.

For one eligible run/checkpoint let D be the set of seeded issues declared complete, E the set confirmed by the evaluator, and I the common eligible issue set. The two marginal rates are $|D|/|I|$ and $|E|/|I|$. The object-level summary reports declaration confirmation $|D \cap E|/|D|$, unconfirmed declarations $|D \setminus E|/|D|$, and confirmed-but-undeclared issues $|E \setminus D|$.

Table 28: **Object-matched declaration diagnostics.**

Matched declaration diagnostics				
Matched diagnostic	**B0**	**B1**	**B2**	**B3**
Declared issues confirmed, $\|D \cap E\|/\|D\|$	**56.68%**	56.41%	54.01%	48.37%
Declared issues unconfirmed, $\|D \setminus E\|/\|D\|$	**43.32%**	43.59%	45.99%	51.63%

Notes. Terra+Opus endpoint declaration records.

I.4 Action Allocation and Delivery Conversion

I.4.1 Action Totals and Category Coverage

Table 29 reports mapped and unmapped action totals for the Terra+Opus runs. Action shares use mapped actions within each condition.

Table 29: **Terra+Opus action-log totals.**

Terra+Opus action-log accounting				
Action accounting	**B0**	**B1**	**B2**	**B3**
Mapped actions	9,459	86,288	82,111	88,289
Unmapped actions	0	1	0	0

I.4.2 Delivery Funnel

Table 30: **Raw delivery-object counts across 60 runs per condition.**

Delivery funnel: repository-object counts				
Stage (60-run raw totals)	**B0**	**B1**	**B2**	**B3**
Generated patches	1,868	9,048	8,610	8,079
Accepted patches	1,508	6,684	6,473	6,381
Accepted patches reaching mainline	510	1,435	1,971	**2,234**
Opened PRs	498	1,695	2,361	2,908

Continued on next page.

Table 30 (continued)

Delivery funnel: repository-object counts				
Stage (60-run raw totals)	**B0**	**B1**	**B2**	**B3**
Merged PRs	352	1,168	1,743	**2,090**

Notes. Delivery rates use model–workload macro averaging.

I.4.3 No-Visible-Progress and Rework Diagnostics

No-visible-progress actions record effort or status without touching an artifact, patch, branch, review, or message. The action taxonomy assigns `work_on_task`, `rest`, `defer`, `overtime`, and `idle` to this category.

I.5 Organizational-Mechanism Inventory

I.5.1 Proposals and Adoption

Table 31 summarizes 617 terminal protocol objects across the 60 Terra+Opus B3 runs: 513 adopted, 92 proposed, and 12 obsolete. Adopted objects comprise 393 autonomous and 120 environment-seeded rules. The 12 obsolete lineages each retain a nonempty `adoption_tick`, which joins their earlier adoption to the terminal lifecycle state. Runtime `protocol_enforcements` counts accumulate over the recorded trajectory, including events preceding retirement.

Table 31: **B3 protocol-state inventory and recorded runtime activity.**

B3 protocol inventory and runtime activity		
A. Terminal registry status across the Terra+Opus 60 B3 runs		
Terminal status	**Autonomous**	**Environment-seeded**
Currently adopted	393	120
Proposed, not adopted	92	0
Obsolete after adoption	12	0
Terminal objects by origin	**497**	**120**
All terminal protocol-registry objects		**617**
All currently adopted protocols		**513**
All ever-adopted lineages, including later obsolete rules		**525**
B. Recorded runtime activity by workload		
Workload	**Recorded uses**	**Runtime enforcements**
W01	6,450	4,199
W02	4,453	1,270
W03	4,848	4,297
W04	4,648	582
W05	4,231	3,997
W06	4,155	2,452
W07	4,235	4,373
W08	6,087	5,144
W09	5,230	1,970
W10	5,253	5,159
All B3 runs (all protocols)	**49,590**	**33,443**
B0/B1/B2, each condition	0	0

Notes. The environment-seeded rules are `proto_review_before_merge` and `proto_experiment_logging`, each present in all 60 Terra+Opus B3 runs. Runtime activity totals include both autonomous and seeded rules.

I.5.2 Use and Enforcement

Table 31 reports protocol-use and enforcement events. The mechanism case links 211 enforcements to 12 pull requests (Appendix J.8.1).

I.5.3 Amendment and Lifecycle

Of the 280 formed lineages, 182 have revisions, totaling 1,231 revision entries and 1,231 folded or superseded proposal identifiers. Retirement is recorded as `obsolete`; repair proposals use `policy_repair_proposal`.

I.5.4 Capability Families

Recurring protocol themes include commit-bound verification, interface-to-contract checking, review assignment, experiment logging, release readiness, and handoff readiness.

I.5.5 Corpus, Counting Units, and Formation Coverage

The autonomous-protocol census covers the 60 Terra+Opus B3 runs. Autonomous lineages exclude `proto_experiment_logging` and `proto_review_before_merge`.

The census counts within-run protocol lineages identified by `(run, protocol_id)`, with revisions grouped into their parent lineage. Formation uses the criterion in Appendix C.4.3; the strong grade adds object-state and evaluator-linked evidence.

Coverage. The census links terminal protocol states, structured rule content, formation records, family labels, roles, and revisions across all 60 Terra+Opus B3 runs. Each formed registry ID resolves to its rich `ProtocolSpec`; the `(run, protocol_id)` key joins its proposal, adoption, use, and revision records while grouping versions into one lineage.

Table 32: **Autonomous-protocol census over the Terra+Opus B3 population.**

Autonomous formation: Terra+Opus 60-run census		
Stage or coverage	**All 60 B3 runs**	**Interpretation**
Autonomous proposal lineages	497	Excludes both seeded rules.
Currently adopted lineages	393	Terminal `adoption_status=adopted`.
Unadopted proposal lineages	92	Terminal `adoption_status=proposed`.
Subsequently obsolete lineages	12	Terminal `obsolete`; all have an adoption step.
Ever-adopted lineages	405	393 currently adopted plus 12 subsequently obsolete.
Weak-or-strong formed	280	Exact formation count.
weak-only	248	Strong is excluded from this row.
strong subset	32	Object-level and evaluator-linked evidence grade.

Across the complete 60-run census, 393 of 497 autonomous proposal lineages remain adopted at the endpoint (79.1%). Another 12 were adopted and later became obsolete, giving 405 ever-adopted lineages (81.5% of proposals); the remaining 92 are unadopted proposals. The 280 weak-or-strong formed lineages constitute 69.1% of the ever-adopted set and 56.3% of all autonomous proposal lineages. Of the formed lineages, 248 are weak-only (88.6%) and 32 strong (11.4%). Every formed registry ID maps to its rich `ProtocolSpec`, allowing versions to remain in one lineage.

I.5.6 Semantic Clustering and Functional Categories

We distinguish raw naming from functional grouping. The 280 formed lineages use 243 distinct raw `protocol_type` strings. These strings capture free naming and task-specific wording. All 280 map to a rich spec with an existing, runtime-recorded `family`. The present grouping uses these recorded labels, with content inspection organized around triggers, concrete obligations, roles, affected objects, and intended effects. Eight of the ten registered families occur; `docs` and `other` do not occur as primary family labels.

Table 33: **Recorded functional families of formed autonomous protocols.**

Formed capabilities by functional family							
Recorded family	**Formed lineages**	**Share**	**Runs /60**	**Tasks /10**	**GPT-5.6 Terra**	**Claude Opus 4.6**	**Strong**
Review and merge	108	38.6%	48	10	38	70	14
Release engineering	80	28.6%	46	10	39	41	8
Evidence governance	56	20.0%	38	10	26	30	6
Debugging	17	6.1%	12	9	3	14	4
Customer	10	3.6%	7	4	0	10	0
Ownership	7	2.5%	6	6	2	5	0
Experiment	1	0.4%	1	1	0	1	0
Budget governance	1	0.4%	1	1	0	1	0
Total	**280**	**100%**	—	**10**	**108**	**172**	**32**

Notes. Family counts use one primary label per lineage. Run and workload coverage may overlap across families.

Review and merge. The largest family typically responds to an opened or updated PR, or a claim that work is merge-ready or release-ready. Obligations include mapping issue acceptance criteria to tests, rerunning applicable CI against current mainline, recording a non-author review decision, and refusing readiness claims or merges when evidence, ownership, CI, or review is incomplete. The 108 lineages cover all ten workloads.

Release engineering. These 80 lineages center on release candidates and release gates. A recurring sequence assigns an owner, locates the failing module, applies a patch, runs targeted and smoke/readiness tests, obtains reviewer sign-off, and reruns the release gate before publishing or declaring readiness. This family and review/merge both use CI and evidence, but differ in their principal object: a release candidate versus a PR and mainline promotion.

Evidence governance. The 56 lineages regulate claims about fixes, compatibility, resolution, experiment results, or release status. They require applicable acceptance behavior, preserved compatibility, current verification status, and traceable evidence; insufficiently verified work remains pending, blocked, or under review rather than resolved or shipped. Coverage across all ten workloads shows recurrence of this protocol theme.

Debugging and recovery. Seventeen lineages join failure signatures, suspected modules, patches, targeted tests, smoke results, and review sign-off into a closure chain. Four have strong evidence, a larger within-family fraction than the overall strong fraction.

Customer, ownership, experiment, and budget. The ten customer lineages, all in the Claude Opus 4.6 records, connect customer issues to implementation or PRs, triage, owners, and response dates. Seven ownership lineages specify responsibility and handoff for tasks or artifacts. One experiment lineage requires shared tracker entries with run content, reproducibility state, seed/configuration, raw trace, and cost. The single budget-governance lineage targets *institutional cost*: when a protocol blocks too many requests in a week, its benefit and time cost must be reviewed and revision or retirement may be proposed.

Structural completeness, revisions, and role templates. All 280 formed lineages have nonempty triggers, responsibility maps, success metrics, and enforcement-action fields. Problem evidence is present in 264 (94.3%), required steps in 274 (97.9%), and required fields in 270 (96.4%). These fields describe the structured content of formed protocols.

Table 34: **Field completeness of the 280 formed rich protocol specifications.**

Structured-field coverage of formed protocols		
Structured field	**Nonempty /280**	**Coverage**
Trigger condition	280	100%
Responsible roles	280	100%
Success metric	280	100%
Enforcement action	280	100%
Problem evidence	264	94.3%
Required steps	274	97.9%
Required fields	270	96.4%

Executor-role assignments are founder 179, cofounder 27, reliability 18, editorial 16, fast engineer 13, artifact design 12, community 8, and external voice 7. Every spec's reviewer and approver fields contain the same `cofounder + founder` pair. The generation template fixes this reviewer/approver pair; executor-role assignments vary by protocol.

I.5.7 Diversity, Concentration, and Cross-Workload Reuse

The three largest families account for 244/280 formed lineages (87.1%). With family shares p_k, the observed Herfindahl index is $\sum_k p_k^2 = 0.276$. Shannon entropy is $H = -\sum_k p_k \log p_k = 1.469$ nats, giving $H/\log 8 = 0.706$ across the eight observed families and an effective family count $\exp(H) = 4.34$. The distribution has a long tail but is concentrated around review, release, and evidence obligations. These statistics summarize lineage-weighted diversity over the recorded functional families.

Table 35: **Model strata of the Terra+Opus B3 protocol census.**

Model strata of the capability census		
Complete model stratum	**GPT-5.6 Terra**	**Claude Opus 4.6**
B3 runs with content and evidence	30	30
Formed lineages	108	172
Strong lineages	14	18
Mean formed per run	3.60	5.73

The GPT-5.6 Terra stratum contains 108 formed lineages across five recorded families, while the Claude Opus 4.6 stratum contains 172 across eight. These strata characterize how the observed capability census is distributed across the two model settings.

Review/merge, release engineering, and evidence governance each occur across 10/10 workloads; debugging occurs across nine, ownership six, customer four, and experiment and budget one each. Within a recurring family, triggers, fields, and affected artifacts remain workload-specific.

Episode-window reconstruction shows post-adoption use beyond the originating episode window for all 280 formed lineages. This complements the cross-workload family recurrence and the independent-organizational-unit criterion used in the formation classifier.

I.5.8 Capability Coverage

The formed-protocol census is concentrated in review/merge, release engineering, and evidence governance, with smaller families covering debugging, customer handling, ownership, experiment

tracking, and institutional cost. A complementary formation-funnel view follows visible friction through recognition, proposal, adoption, and sustained use, while ordinary repairs remain separate from newly formed organizational mechanisms.

I.6 Model-Conditioned Friction and Institutional Responses

We examine recorded work episodes and episode-to-protocol links under GPT-5.6 Terra, Claude Opus 4.6, and DeepSeek-V4-Flash, abbreviated as Terra, Opus 4.6, and DeepSeek.

Overlapping recorded pressures. All three strata encounter debugging, launch pressure, experimentation, feedback, and claim disputes (Table 36). Typical debugging and launch-pressure counts are close, although other categories have different spreads. One Opus 4.6 PDF-workload run records 4,009 experiment episodes; retaining it raises that stratum's experiment mean to 209.86, while its median is 26.5. We therefore describe typical activity using medians and interquartile ranges.

Table 36: **Recorded friction and pressure episodes.** Entries are the reported median [first quartile, third quartile] per run.

Episode category	Terra	Opus 4.6	DeepSeek
Debugging	33.0 [27.5, 45.8]	31.5 [26.0, 39.8]	29.5 [24.0, 39.3]
Launch pressure	34.0 [33.0, 36.8]	35.0 [33.0, 37.8]	33.0 [32.0, 35.0]
Experiment activity	27.0 [24.3, 39.0]	26.5 [22.3, 33.0]	21.0 [18.0, 25.0]
Feedback ingestion	12.0 [10.3, 73.0]	13.0 [11.0, 37.5]	13.0 [11.0, 14.8]
Claim disputes	1.0 [0.0, 1.0]	1.0 [1.0, 1.0]	1.0 [0.0, 1.0]

Episode-to-protocol links. Formed protocol specifications link source-episode categories to protocol families through `source_episode_ids`. Table 37 reports distinct linked lineages for selected source–family combinations. For example, Terra contributes 24 launch-pressure-to-release-engineering links; Opus 4.6 contributes 17 debugging-to-release-engineering links and 13 launch-pressure-to-review/merge links. DeepSeek contributes seven debugging-to-release-engineering links and five experiment-to-review/merge links. These records connect concrete work episodes to protocol formation.

Table 37: **Recorded links from source episodes to formed protocols.** Counts pool the three analysis strata.

Source episode	Protocol family	Distinct linked lineages
Launch pressure	Release engineering	39
Debugging	Release engineering	34
Launch pressure	Review / merge	25
Experiment	Review / merge	18
Feedback ingestion	Review / merge	14
Debugging	Review / merge	11
Feedback ingestion	Evidence governance	5
Experiment	Evidence governance	4
Launch pressure	Evidence governance	4
Claim dispute	Evidence governance	3
Debugging	Debugging	3

Notes. A lineage may link to multiple source-episode categories.

Friction volume and institutional response. Workload–seed-block-centered correlations vary by episode category. Debugging with debugging/release institutions has $r = 0.20$ (95% CI $[-0.07, 0.46]$); claim disputes with evidence governance, $r = 0.19$ $[-0.15, 0.47]$; and launch pressure with review/release, $r = 0.20$ $[-0.06, 0.46]$. Experiment activity with experiment/evidence institutions is

positively associated ($r = 0.24$ [0.08, 0.44]), while feedback with customer/ownership institutions is negatively associated ($r = -0.24$ [-0.33, -0.14]).

The episode links connect recorded work experiences to the protocol-formation history.

I.7 Cost and Efficiency Results

I.7.1 240-Run Token Accounting

Table 38: **Terra+Opus token accounting over 240 runs.**

Main-study token expenditure and outcome-normalized cost					
Token metric	**B0**	**B1**	**B2**	**B3**	**B3–B2 paired**
Terminal total (M)	**211.309**	2,634.198	225.058	391.372	–
Mean / run (M)	**3.522** [3.350,3.695]	43.903 [40.745,47.262]	3.751 [3.439,4.054]	6.523 [5.119,8.379]	**+2.772** [1.373,4.665]
Per confirmed issue (M)	4.947 [3.396,5.963]	31.760 [25.994,46.592]	**2.907** [2.402,4.632]	3.651 [2.359,6.660]	**−0.174** [−1.686,0.633]

Notes. Brackets give 95% intervals. Mean-token/run uses all 60 runs per condition. Per-confirmed-issue arm estimates average each arm's eligible nonzero model–workload blocks; the paired B3–B2 contrast is recomputed on 14 common nonzero blocks (42 paired runs), so it need not equal the arithmetic difference of the two arm estimates.

I.7.2 Outcome-Normalized Cost

Table 38 reports Terra+Opus cost estimates. Table 1 reports the pooled three-model estimates. Both use the block-macro cost definition in Appendix H.2.4.

I.7.3 Budget-Capped Checkpoint Sensitivity

For each (model, workload, seed) pair, the budget-capped sensitivity takes B2's final token expenditure as the cap and selects the last saved B3 checkpoint at or below that cap. The score is the evaluator output at the selected checkpoint. All 60 Terra+Opus matched units have a qualifying checkpoint with an evaluator output.

Token receipts provide recoverable monotone checkpoint curves for the 60 B3 runs, and terminal totals match the corresponding frozen token records. The analysis joins each checkpoint's recorded cumulative spend, candidate state, and evaluator output. It equally weights model–workload blocks and uses 10,000 paired-seed bootstrap draws with seed 1729.

Table 39: **Budget-capped checkpoint sensitivity.**

Budget-capped checkpoint sensitivity					
Endpoint	**Pairs**	**Blocks**	**B2**	**B3 under cap**	**B3–B2 [95% CI]**
Complete contracts	60	20	16.310%	**23.760%**	**+7.450 pp** [3.491, 10.927]
Held-out cases	54	18	13.43%	**18.51%**	**+5.08 pp** [0.926, 11.111]
Exposed cases	60	20	22.570%	**34.290%**	**+11.720 pp** [6.781, 19.617]
Evaluator-confirmed seeded issues	60	20	20.840%	**27.052%**	**+6.212 pp** [4.528, 11.719]

Notes. Held-out outcomes use W02–W10; all other rows use W01–W10.

All four verified production contrasts remain positive at the selected budget-capped B3 checkpoints.

I.8 Complete Transfer Results

I.8.1 All Intervention Arms

Table 40: **Complete internal-transfer summary.**

Internal protocol transfer: complete outcomes and costs			
Metric	**Fresh**	**Text**	**Exec**
Behavioral-case pass rate	25.4%	34.6%	**41.2%**
Exposed-case pass rate	23.434%	39.7%	**44.4%**
Held-out-case pass rate	11.111%	11.1%	**25.9%**
Complete-contract rate	18.574%	31.5%	**32.3%**
Evaluator-confirmed issue rate	21.720%	41.4%	**45.1%**
Pull requests opened, mean / run	47.1	52.1	48.8
Pull requests merged, mean / run	38.2	42.2	39.7
Releases, mean / run	24.1	26.7	26.2
Patches generated / accepted, mean / run	141.8 / 110.0	124.7 / 122.4	118.0 / 116.7
File-edit actions, mean / run	141.9	121.0	118.0
Total actions, mean / run	1,389.3	1,377.8	1,385.9
Imported mechanisms, mean / run	0	6.0 (text)	6.0 (executable)
New target mechanisms, mean / run	0	0	0
Recorded uses, mean / run	0	0	588.0
Recorded binding enforcements, mean / run	0	0	197.0
Amendments, mean / run	0	0	0
Mean tokens / run (M)	**2.381**	2.615	2.448
Tokens / evaluated case (k)	176	154	**144**
Tokens / verified pass (k)	691	444	**350**

Notes. Count-valued rows are means per run. Confidence intervals for behavioral-case pass rate and cost metrics appear in Table 2.

I.8.2 Primary Controlled Contrasts

Exec improves behavioral-case pass rate by 6.5 pp [0.7, 15.6] over Text and 15.8 pp [9.2, 31.8] over Fresh. Exec also exceeds Text on each production endpoint in Table 40.

I.9 Robustness Checks

I.9.1 Equal-Task and Matched Subsets

Terra+Opus auxiliary analyses use ten workloads, three seeds, and B0–B3. Complete, exposed, and confirmed-issue contrasts have 60 pairs; held-out contrasts have 54 pairs over W02–W10.

I.9.2 Model-Stratified B3–B2 Effects

Table 41: **Model-stratified B3–B2 effects.**

Model-stratified B3–B2 effects				
Endpoint	**B2**	**B3**	**B3–B2 [95% CI]**	**Applicable pairs**
GPT-5.6 Terra				
Complete contracts	18.574%	**23.440%**	**+4.866 pp** [−1.713, 11.359]	30/30

Continued on next page.

Table 41 (continued)

Model-stratified B3–B2 effects				
Endpoint	**B2**	**B3**	**B3–B2 [95% CI]**	**Applicable pairs**
Held-out cases	11.111%	**12.037%**	**+0.926 pp** [−4.630, 7.407]	27/30
Exposed cases	23.434%	**28.082%**	**+4.648 pp** [−0.011, 12.505]	30/30
Confirmed seeded issues	21.720%	**28.082%**	**+6.362 pp** [1.296, 14.220]	30/30
Claude Opus 4.6				
Complete contracts	14.046%	**22.200%**	**+8.154 pp** [4.509, 13.271]	30/30
Held-out cases	15.741%	**35.185%**	**+19.444 pp** [0.167, 49.333]	27/30
Exposed cases	21.706%	**32.518%**	**+10.812 pp** [3.911, 17.613]	30/30
Confirmed seeded issues	19.960%	**29.438%**	**+9.478 pp** [5.113, 18.018]	30/30
DeepSeek-V4-Flash				
Complete contracts	9.54%	**13.64%**	**+4.10 pp** [1.31, 7.13]	30/30
Held-out cases	1.85%	**4.63%**	**+2.78 pp** [0.93, 5.56]	27/30
Exposed cases	12.79%	**17.08%**	**+4.29 pp** [0.77, 7.92]	30/30
Confirmed seeded issues	12.05%	**17.08%**	**+5.03 pp** [1.73, 8.45]	30/30

Notes. Each model has 30 matched pairs for complete, exposed, and confirmed-issue outcomes and 27 for held-out outcomes. DeepSeek intervals use 50,000 same-workload, matched-seed bootstrap draws.

All four B3–B2 point estimates are positive for GPT-5.6 Terra, Claude Opus 4.6, and DeepSeek-V4-Flash, with no endpoint direction reversal across models. For DeepSeek-V4-Flash, the paired 95% intervals for all four verified endpoints exclude zero. For held-out cases specifically, B0/B1 are 0.00% [0.00, 0.00], B2 is 1.85% [0.00, 3.70], and B3 is 4.63% [0.00, 8.33], with B3–B2 +2.78 pp [0.93, 5.56]. Its additional behavioral diagnostics are positive on mainline (B0/B1/B2/B3: 4.80 [2.78, 7.06] / 11.50 [9.42, 13.59] / 10.28 [7.09, 13.49] / 13.64 [11.55, 15.90]%; B3–B2 +3.35 pp [0.28, 6.59]) and on workspace (9.88 [6.98, 12.46] / 17.54 [15.20, 20.11] / 15.21 [11.50, 18.63] / 17.57 [14.76, 20.55]%; +2.37 pp [−2.13, 7.15]). Under the same block-macro outcome-normalized cost definition used for the main-study paired estimand, the DeepSeek B3–B2 effect is −1.087M tokens per confirmed issue.

I.9.3 Leave-One-Workload-Out Analysis

Table 42: **Leave-one-workload-out robustness.**

Leave-one-workload-out robustness			
Endpoint	**Full B3–B2 [95% CI]**	**LOO > 0**	**LOO CI > 0**
Complete contracts	**+6.51 pp** [2.812, 10.517]	10/10	10/10
Exposed cases	**+7.73 pp** [2.183, 13.016]	10/10	8/10
Confirmed seeded issues	**+7.92 pp** [2.908, 14.409]	10/10	10/10

Continued on next page.

Table 42 (continued)

Leave-one-workload-out robustness			
Sensitivity range across workload omissions			
Endpoint	**Minimum LOO**	**Maximum LOO**	**Max. abs. deviation**
Complete contracts	+5.39 pp omit W04	+7.98 pp omit W01	1.46 pp
Exposed cases	+6.58 pp omit W07	+9.27 pp omit W01	1.55 pp
Confirmed seeded issues	+6.68 pp omit W03	+9.54 pp omit W01	1.62 pp

Notes. Each refit omits one workload from both models and paired conditions. "LOO CI > 0" counts intervals wholly above zero.

The three reported endpoints remain positive under every W01–W10 omission, so the aggregate direction is not attributable to a single workload. Complete-contract effects range from +5.39 to +7.98 percentage points, exposed effects from +6.58 to +9.27 points, and confirmed-issue effects from +6.68 to +9.54 points.

I.10 In-Situ Prose-Only Institutionalization Ablation

I.10.1 Design and Intervention

The internal Text/Exec experiment fixes a transferred package and disables new target-stage formation. Here we test executable binding inside the original organizational setting while proposal generation, governance, approval, adoption, and revision remain enabled. The completed ablation uses Claude Opus 4.6, all ten workloads W01–W10, and three seeds (1401, 2711, and 4013), yielding 30 new B3-text runs matched to the corresponding 30 executable B3 runs from the main study by workload and seed.

B3-text retains the B3 roster, member-local learning, reflection, wishes, protocol proposals, governance, adoption, revision, the core SDL scoring parameterization, and ordinary task and repository validity checks. The intervention changes what adoption produces operationally: an adopted `ProtocolSpec` remains a readable shared organizational rule and is rendered through the text-only decision-context interface, but it does not install an executable protocol binding. Learned-rule effects on selection and routing, automatic protocol-use recording, and protocol-specific enforcement are disabled. Environment-level CI, review, and transaction-validity checks continue to apply.

Both conditions begin at the initial state and independently develop their work, proposals, and adopted rules throughout the run.

I.10.2 Three-Seed Outcome Comparison

The primary in-situ endpoint is complete contracts on mainline. Complete, exposed, and confirmed-issue outcomes use 30 matched workload–seed units; held-out outcomes use 27 matched units because W01 has no held-out set. For each metric, run-level rates are first averaged over the three seeds within each applicable workload, then equally weighted across applicable workloads: ten for complete, exposed, and confirmed-issue outcomes, and nine for held-out outcomes. Table 43 reports the aggregate comparison.

Table 43: **Three-seed in-situ B3-text versus executable B3.**

In-situ binding ablation: verified mainline outcomes				
Endpoint	**B3-text**	**B3**	**B3−B3-text**	**Paired 95% CI**
Complete contracts on mainline	15.03%	**22.20%**	**+7.18 pp**	**[+3.54, +10.91] pp**
Held-out cases on mainline	12.96%	**35.19%**	**+22.22 pp**	**[+10.85, +29.71] pp**

Continued on next page.

Table 43 (continued)

In-situ binding ablation: verified mainline outcomes				
Endpoint	**B3-text**	**B3**	**B3−B3-text**	**Paired 95% CI**
Exposed cases on mainline	18.47%	**32.52%**	**+14.04 pp**	**[+8.99, +20.23] pp**
Evaluator-confirmed seeded issues	17.72%	**29.44%**	**+11.71 pp**	**[+6.27, +15.91] pp**

Notes. Rates macro-average the three seeds within each workload, then equally weight applicable workloads. Intervals are paired 95% CIs; held-out outcomes use W02–W10.

Executable B3 exceeds B3-text by 7.18 percentage points on complete contracts (95% CI [+3.54, +10.91]). The same direction appears on held-out (+22.22 pp), exposed (+14.04 pp), and evaluator-confirmed seeded-issue (+11.71 pp) endpoints. All four paired 95% intervals are above zero.

I.10.3 Matched Complete-Contract Endpoints

Table 44 reports complete-contract counts for the 30 matched B3-text / B3 pairs.

Table 44: **Matched complete-contract endpoints for B3-text / B3.**

Matched complete-contract endpoints: B3-text / B3			
Workload	**Seed 1401**	**Seed 2711**	**Seed 4013**
W01 Mini Blobstore	0/5 / 0/5	0/5 / 0/5	0/5 / 0/5
W02 Traffic Watch	2/9 / 2/9	0/9 / 2/9	1/9 / 1/9
W03 TG Automation	1/9 / 5/9	1/9 / 1/9	1/9 / 3/9
W04 PDF Reformatter	0/4 / 1/4	0/4 / 2/4	0/4 / 1/4
W05 FastAPI Dashboard	0/4 / 0/4	0/4 / 0/4	0/4 / 0/4
W06 Boltons	10/16 / 11/16	10/16 / 12/16	12/16 / 10/16
W07 Celery	1/16 / 3/16	1/16 / 1/16	2/16 / 2/16
W08 Soup Sieve	1/13 / 1/13	0/13 / 1/13	1/13 / 1/13
W09 cattrs	4/16 / 5/16	2/16 / 4/16	4/16 / 2/16
W10 Tenacity	6/16 / 4/16	2/16 / 5/16	5/16 / 3/16
10-workload macro	**17.23% / 25.42%**	**10.49% / 22.85%**	**17.37% / 18.34%**

Notes. Each cell gives B3-text / executable B3. The three-seed aggregate is 15.03% / 22.20%, yielding the +7.18 pp contrast in Table 43.

I.10.4 Seed-Stratified Endpoints

Table 45 reports seed-stratified endpoint rates.

Table 45: **Seed-stratified endpoint macros for the in-situ ablation.**

Seed-stratified endpoint macros				
Metric	**Arm**	**Seed 1401**	**Seed 2711**	**Seed 4013**
Complete contracts	B3-text	17.23%	10.49%	17.37%
Complete contracts	B3	25.42%	22.85%	18.34%
Held-out cases	B3-text	13.89%	2.78%	22.22%
Exposed cases	B3-text	20.96%	13.10%	21.37%
Evaluator-confirmed seeded issues	B3-text	19.21%	13.10%	20.87%

I.10.5 Binding Manipulation Check

Table 46 reports proposal, adoption, readable-rule, and executable-binding state across the 30 matched runs.

Table 46: **Three-seed B3-text manipulation check.**

B3-text manipulation check					
Check	**Seed 1401**	**Seed 2711**	**Seed 4013**	**B3-text total**	**B3 total**
Protocol proposals	106	115	103	**324**	347
Adopted protocols	70	81	76	**227**	321
Endpoint-renderable adopted rules	66	80	75	**221**	249
Cells with readable-rules block	10/10	10/10	10/10	**30/30**	—
Cells with executable binding enabled	0/10	0/10	0/10	**0/30**	30/30
Recorded protocol-use events	0	0	0	**0**	25,079
Protocol-specific enforcement events	0	0	0	**0**	18,594

Notes. Endpoint inventories and whole-run event totals across the matched 30 runs; seed columns each cover ten workloads.

J Case-Study Evidence: Commit-Bound Verification

This case traces protocol formation, enforcement, revision, and delivery in the W01 Mini Blobstore run with GPT-5.6 Terra and seed 1401. Displayed member, protocol, and event identifiers use consistent aliases.

J.1 Case Configuration

J.1.1 Matched Diagnostic Configuration

Four runs, one per arm, on Mini Blobstore (`W01`) at seed 1401 with a 336-step horizon and model `gpt-5.6-terra` with low reasoning effort. All four share pack, starter tree, hidden suite, evaluator, seed, horizon, model, tool surface and information; circadian rhythm is ablated in all four. The primary endpoint:

Table 47: **Matched case diagnostic, endpoint on mainline.**

Selected case: verified mainline endpoints				
Endpoint	**B0**	**B1**	**B2**	**B3**
Behavioural cases passing	0/35	0/35	0/35	**28/35**
Complete contracts	0/5	0/5	0/5	**2/5**

J.2 Repository and Action Counts

J.2.1 Repository-State Counters

Table 48: **Repository objects and action occurrences in the matched case.**

Selected case: repository and action counters					
Source	**Counter**	**B0**	**B1**	**B2**	**B3**
repo state	code patches accepted	74	168	113	118
repo state	pull requests opened	1	8	16	80
repo state	pull requests merged	0	2	4	**76**
action log	`edit_repo_file`	75	169	131	112
action log	`commit_patch`	1	12	50	93
action log	`open_pr`	0	0	3	20
action log	`merge_pr`	0	2	1	28
action log	`run_ci`	33	427	142	126
CI records	runs recorded	73	651	589	339

Notes. Some runtime paths create or merge pull requests without emitting a separate `open_pr` or `merge_pr` action-log entry.

J.3 Work Without Delivery

J.3.1 Local Production

The baseline arms continued to perform editing and verification actions. Actions taken per 48-step window are flat to the horizon: at $t \in [0, 48)$ the four arms take 23, 206, 194 and 208 actions; at $[96, 144)$ they take 22, 198, 184 and 231; at $[192, 240)$ they take 18, 204, 183 and 200; at $[288, 336)$ they take 17, 185, 174 and 234. B1 is as busy at $t = 330$ as at $t = 10$. B0 is a single member, so its twenty per window is comparable effort per head.

For local editing, B3 made 112 `edit_repo_file` actions and had 118 code patches accepted, against B1's 169 and 168. **B3 has the fewest recorded file-edit actions of the three multi-member arms**. The selected B1 mainline endpoint remains zero despite its larger number of these local operations.

J.3.2 Shared Delivery

Where they differ is handover. B0 committed once. B1 merged two pull requests, both inside the first 48 steps, opened its last at $t = 96$, and spent the remaining 240 steps writing and checking without handing anything over: its final 96 steps contain 114 CI runs, 63 edits, 62 internal searches, 51 public-test runs, and no pull request opened or merged. B2 opened sixteen and merged four, all four inside the first 48 steps; its final 96 steps went to `work_on_task` (114) and `update_task_status` (77), which is task bookkeeping around work that never left the desk. B3's final 96 steps are shaped differently: `work_on_task` 50, `run_ci` 36, `edit_repo_file` 27, `commit_patch` 26, `approve_proposal` 18.

J.3.3 Delivery Over Time

Table 49: **Pull requests opened / merged per 48-step window, from repository state differenced per window.**

Selected case: shared delivery over time				
Window	**B0**	**B1**	**B2**	**B3**
steps 0–48	1 / 0	6 / 2	12 / 4	13 / 9
steps 48–96	0 / 0	2 / 0	3 / 0	13 / 12
steps 96–144	0 / 0	0 / 0	1 / 0	13 / 14
steps 144–192	0 / 0	0 / 0	0 / 0	7 / 9
steps 192–240	0 / 0	0 / 0	0 / 0	13 / 9
steps 240–288	0 / 0	0 / 0	0 / 0	10 / 15
steps 288–336	0 / 0	0 / 0	0 / 0	11 / 8
Total	1 / 0	8 / 2	16 / 4	**80 / 76**

Early B3 adoptions occur at $t = 21$, 39, 58, 60, and 65. All recorded baseline merges occur in the first 48-step window; baseline PR openings extend into later windows, through $t = 96$–144 for B2. Adoption timing, PR creation, and merged delivery are shown as distinct series.

J.4 Capability Inventory and Governance

J.4.1 All Proposed Mechanisms

Table 50: **Every mechanism this run proposed, adopted or failed to adopt.**

Selected case: complete proposal and adoption inventory					
Proposal	**Proposed**	**Adopted**	**Uses**	**Enf.**	**Amend.**
P01	step 12	step 21	419	211	2
P02	step 13	step 39	58	0	0
P03	step 7	step 58	18	0	0
P04	step 48	step 60	247	0	4
P05	step 48	step 65	242	0	7
P06	step 264	step 276	39	0	0
P07	step 44	never	—	—	—
P08 (1st)	step 267	never	—	—	—
P09 (2nd)	step 278	step 291	—	—	—
P10 (3rd)	step 321	never	—	—	—

Notes. Event-ledger proposal and adoption records. A dash denotes an unavailable count.

J.4.2 Adopted and Rejected Mechanisms

The run proposes ten mechanisms, adopts seven, and leaves three unadopted. A merge-readiness handoff is adopted on its second attempt. P02 uses a generic review-before-merge template; P03 uses a generic experiment-logging template.

J.4.3 Uses, Enforcements, and Amendments

Counts are in Table 50. At the $t = 336$ endpoint, elapsed time since adoption is $336 - 21 = 315$ steps for the principal rule and $336 - 60 = 276$ steps for the current-commit integration-evidence rule. Lifecycle records separately track activation, revision, retirement, and last-active state. Amendments total 13 across three protocols, the last at $t = 317$, so revision continued to within twenty steps of the horizon. The case's 211 reported enforcements are linked to P01 and 12 distinct pull requests (Appendix J.8.1). P02–P06 have positive reported use counts and zero reported enforcements.

J.5 Related Rules for Verification and Evidence Freshness

J.5.1 Five Convergent Rules

Five related mechanisms address evidence completeness, current-commit verification, and readiness as repository state changes. Four members proposed them (M02, M04, M01, M05). The following clauses retain the source requirements, with task-identifying names and object identifiers anonymized; the clauses are:

- P01 (M02, proposed step 12, adopted step 21, 315 steps from adoption to the endpoint): a pull request affecting public callables or cross-module behaviour is merge-eligible only when its review record identifies every changed public callable signature, identifies every added or modified cross-module invocation, maps each identified item to the applicable written contract, and *records successful execution of the applicable public smoke tests through the highest stacking step affected by the change*; anything unmappable, or any required smoke failure, must be resolved or explicitly rejected before merge approval.
- P04 (M04, proposed step 48, adopted step 60, 276 steps from adoption to the endpoint, revised four times): a pull request changing a stacking-step implementation or a shared-state boundary must not be merged unless its recorded CI result and relevant public smoke result *both identify the exact current PR head commit as the tested commit* and both pass; *if mainline changes after either result is recorded, the PR must be synchronized and both*

rerun on the resulting current head; and a reviewer or merger *must be able to compare the recorded tested commit with the current head and determine pass or fail without relying on an author's statement*.

- `P05` (M01, adopted step 65, revised seven times): no release, launch announcement or readiness claim may proceed until the checklist is complete, *CI passes on current mainline*, required reviews are complete, and the external claim is limited to the verified dependency chain.
- `P06` (adopted step 276): not merge-ready until the review record contains a complete trace for each changed completion entry point and *a linked passing affected integration or public-contract smoke result for the PR head*.
- `P09` (adopted step 291 on the second attempt): the owner *records current mainline revision and passing CI immediately before merge; any mainline movement invalidates that*.

J.5.2 Shared Principle

The following formulation summarizes their shared verification concern:

> *A verification result belongs to a commit, not to a pull request — and when mainline moves, the result expires.*

The integration-evidence rule explicitly binds both CI and public smoke results to the current PR head and requires reruns after mainline movement. The readiness rule refers to current-mainline evidence, while the contract-boundary rule emphasizes checkable interface-to-contract mappings. The trace and handoff rules add related obligations.

The learned rules organize responsibility, review evidence, and readiness obligations around the shared commit-bound CI workflow.

J.5.3 Relation to the Observed Failure Mode

The CI ledger measures the absence of that discipline in the other arms (Table 51): B0 made one commit and ran CI against it 73 times, with none passing; B1 made twelve commits, recorded 651 CI runs, one single commit accounting for 162 of them, and passed two. The baseline arms repeatedly tested a small set of commits while delivering few changes. This repeated-verification pattern coexists with the environment-level commit-binding checks shared across conditions and motivates the additional organizational evidence discipline learned in Relic.

J.6 CI Ledger

J.6.1 Runs, Commits, Passes, and Merges

Table 51: **CI ledger for the matched diagnostic.**

Selected case: commit-bound verification and delivery					
Condition	**CI runs**	**Commits**	**Runs/ commit**	**CI passes**	**Merged**
B0	73	1	73.0	0	0
B1	651	12	54.2	2	2
B2	589	50	11.8	5	4
B3	339	93	3.6	**126**	**76**

J.6.2 Repeated-Verification Pathology

B0 made one commit in 336 steps and ran CI against it 73 times; none passed. B1 made twelve commits and recorded 651 CI runs, of which its most-tested object, `C-A`, accounts for 162 by itself; two passed. B2 sits between, at 11.8 runs per commit and five passes. B3 tested each of 93 commits about three and a half times.

The distinction that matters is between testing a lot and testing something that has moved. A rule requiring current-head evidence and freshness after mainline movement expresses a discipline relevant to this pathology.

J.7 One Rule's Governed Lifecycle

J.7.1 Originating Friction and Challenge

The initiating episode is `EP-A`, recorded as a speed-versus-quality tension in which the timeline did not establish that the implementation work had been validated end to end. The first challenge is a formal review in which M01 refuses pull request PR-C and states the requirement in the refusal: add the missing tests, provide the relevant test file and evidence, and request a review before it can proceed. Recurrence follows as repeated `ci_contract_break` events against different pull requests inside the same window. All three are events with identifiers, actors and steps, and all three were visible to the proposer at proposal time.

J.7.2 Proposal, Support, and Adoption

The proposal is made at $t = 12$, naming signature drift across a module boundary as the friction it is written against. It is supported and adopted at $t = 21$, with M02 as supporter. On adoption the compiled spec is registered and enters the action context.

J.7.3 First Use and First Enforcement

The first use event occurs at step 22, one step after adoption. At step 26, the ledger records a protocol-linked refusal of the requested merge and retains paired before/after snapshots of the governed PR. The event identity, step, protocol reference, and object snapshots connect the adopted rule to this concrete execution consequence. Subsequent PR records link the same object to repair, verification, and its final delivery state.

J.7.4 Amendments While in Force

The rule is amended twice while in force, at $t = 33$ by M01 and at $t = 59$ by M02. Governance applies to the amendments as well as to the rule: the source of the second, `AM-A`, was itself edited by M02 at $t = 39$ and by M01 at $t = 43$ before it was adopted. Enforcement per step rose from 0.50 to 0.68 across the first amendment and from 0.55 to 0.69 across the second. These rates summarize the rule's recorded enforcement activity across the two amendment intervals.

J.7.5 Later Reuse

The principal rule records 419 uses and 211 enforcements through $t = 336$, spanning 315 steps from adoption to the endpoint. It satisfies the formation criterion in Appendix C.4.3. P09 is adopted at $t = 291$ and has a 45-step post-adoption observation window.

J.8 Pull-Request-Level Consequences

J.8.1 Distinct Pull Requests Affected

The 211 enforcement events involve **12 distinct pull requests**.

J.8.2 Delayed and Later-Merged Work

Eleven of the twelve blocked requests subsequently merged, 15 to 112 steps after their first recorded block, usually with additional patches. These block-to-merge intervals include the subsequent implementation, verification, and coordination needed before delivery.

J.8.3 Work Still Blocked at the Observation Horizon

One request, `PR-B`, remained blocked and unmerged at the recorded horizon, without an observed override, despite approval by an agent occupying the reviewer role. Its head, `C-B`, was verified 87

times. The organization responded by building a tool named after the repair rather than by overriding the gate, preserving the rule while pursuing a repair.

J.8.4 Unaffected Merges

Of the 76 merges, **65 were never blocked by this rule**. Eleven previously blocked requests later merged and one remained unmerged at the horizon.

J.9 Case Interpretation

The case traces recurring integration friction into a proposed and adopted rule, followed by repeated use, pull-request enforcement, amendments, and sustained shared delivery.

K Integrity and Reproducibility

This section connects the execution interfaces, frozen artifacts, run records, and recovery procedures used to reproduce and inspect the reported experiments.

K.1 Evaluator Isolation

K.1.1 Filesystem and Process Interfaces

The member-facing substrate exposes the project identifier, product name, dataset directory, and starter directory. Evaluator assets occupy the private hidden-test, held-out-requirement, and reference directories. Scoring creates a one-shot candidate workspace outside the simulation and hashes its tree before and after execution. The recorded hashes bind the score to the exported candidate state.

In the container configuration, evaluation runs in a separate untrusted, network-disabled container built from a digest-pinned image. In the host configuration, it runs in a separate process against the exported tree. The execution-policy receipt identifies the mode, dependencies, and artifact identities used for the corresponding score (Appendix E.3.3).

K.1.2 Hidden-Test and Reference Isolation

The evaluator resolves hidden suites and reference content through the 32-byte token-keyed vault described in Appendix E.3.2. World state and checkpoints retain the versioned binding. Constant-time token comparison controls vault lookup. Artifact, search, perception, and snapshot tests inspect the member-facing representations, and initialization validates that private evaluator assets use the vault interface.

K.1.3 Output Interface

Endpoint evaluation runs after rollout and writes evaluator-side contract/check records. During rollout, members receive the outputs of public tests, CI, and ordinary readiness/release gates, including their associated failure reasons. Hidden-test source, assertions, held-out requirements, and reference implementation remain in the private evaluator interface. The run records retain both the member-visible execution feedback and the offline evaluation needed to reconstruct the sequence.

K.2 Temporal and Network Controls

K.2.1 Network Restrictions

Repository operations use the in-process repository model, and the search system's five domains use internal or frozen corpora. The frozen-web domain is a seeded document snapshot. Replay resolves searches through the saved corpus and cache; an unresolved lookup raises a cache-miss error. Product and evaluator execution use the proxy and loopback-only socket controls in Appendix E.3.3. Model-provider calls are the designated network egress. When enabled, the prompt-visibility interceptor records call hashes and coverage.

K.2.2 FUTURE-HISTORY RESTRICTIONS

The member-visible starter is the designated frozen task snapshot. Public issue text supplies the problem and acceptance requirements. Later repository versions, hidden requirements, reference code, and source-identifying metadata are retained in evaluator-side provenance. Workload names and upstream version pairs in the paper link the results to those frozen source records.

K.2.3 ACCESS VALIDATION

Runtime access tests, enabled prompt audits, and post-run future-recall checks record information-interface validity. A flagged run is handled under the recorded exclusion policy. The flag, run identity, and validity reason remain associated with its execution and analysis record.

K.3 BENCHMARK IDENTIFIERS AND ARTIFACT JOINS

W01–W10 consistently join workload manifests, result tables, transfer records, and case-study references. Table 16 gives their names, original pack IDs, and upstream version pairs in the frozen analysis order. Member, protocol, and event aliases retain consistent mappings within the case. These identifiers connect the paper's summaries to the corresponding task definitions and event records.

K.4 RUN-BUNDLE SCHEMA

K.4.1 CONFIGURATION AND IDENTITY

The versioned record `experiment_run_record.json` uses schema `orgenv_experiment_run_v2`. Its identity fields include `run_id`, `pack`, `condition`, `arm_id`, `seed`, `provider`, `model`, and `reasoning_effort`. Configuration fields retain `action_selection_mode`, `profile_assignment`, `experiment_phase`, `oss_control`, `evaluation_perturbation`, and `mechanism_ablations`.

The record schema associates these values with the `ablation_fingerprint`, `llm_runtime_identity`, `llm_runtime_fingerprint`, `resource_budget_fingerprint`, `tool_surface_fingerprint`, and `information_budget_fingerprint`. Artifact fields include `dataset_manifest_hash`, `starter_repo_digest`, `reference_repo_digest`, `candidate_repo_digest`, `hidden_suite_hash`, `evaluator_environment_hash`, `event_graph_hash`, and `run_manifest_hash`. `started_at`, `ended_at`, and `duration_seconds` retain execution timing.

K.4.2 EXECUTION RECORDS

The bundle contains the action log, typed world-event log, model-call records, and perception-derived decision context. `ActionDecision` rows retain validation status and rejection reasons. Checkpoints retain member state and private memory, repository and product state, the protocol ledger and proposal objects, scheduler and background-job state, and the deep per-step replay frames and final snapshot.

Typed identifiers join actions to the work objects they changed and to the governance or protocol records relevant to the transition. The same joins support event-ledger counts, distinct-object counts, and case-study reconstruction.

K.4.3 EVALUATION AND INTEGRITY RECORDS

The final-evaluation artifact retains per-contract status and digest fields, and the reported checkpoint series covers the three-model study. `llm_usage` supplies provider-token receipts. `organizational_capability_evidence` retains formation evidence and its hash, while transfer runs carry a `capability_transfer` receipt. Enabled prompt audits use `prompt_visibility_audit` with a chained hash.

A `completeness` block checks the record against the expected field profile for its run and phase, and `record_notes` preserves validation details. Required-field counts, availability statuses, and

artifact identities remain attached to the record. These checks provide an explicit input-validation step for replay and aggregation.

K.5 Checkpointing and Replay

K.5.1 Serialization

Whole-world serialization stores a versioned payload with a SHA-256, a separately hashed metadata sidecar, and the case-plan, source-provenance, model-binding, and resource-budget fingerprints. The live model client is detached before serialization and reattached after loading. The payload preserves the organization, member, scheduler, and background-job state needed to continue from the saved step.

K.5.2 Resume Equivalence

A zero-step resume loads the checkpoint and checks state equivalence before advancing the simulation. The resume path validates the case-plan, source-provenance, model-binding, and resource-budget fingerprints and rejects a mismatch. The supervised launcher uses this path for recovery after machine interruption.

K.5.3 Replay and Continued Execution

Recorded-event replay reconstructs the checkpoint world state, seeded environment randomness, saved search corpus and cache, and evaluator identity. Case-study timelines use the recorded events and checkpoints. Continued execution restores this state and then issues new provider calls under the saved configuration. The run records retain the restored checkpoint and subsequent calls as one execution lineage.

K.6 Run Validity and Recovery

K.6.1 Failure Taxonomy

Timeout, transport, empty/malformed response, and schema failures enter the provider-error handling path with bounded within-call retries. Returned-model counters preserve the observed model identity. Container and dependency faults are recorded as CI or public-test infrastructure errors. Qualification and evaluator failures retain contract-level statuses, and checkpoint scoring uses its per-contract timeout. Checkpoint hashes validate serialized artifacts; machine termination recovers from the last saved checkpoint.

K.6.2 Retry and Rerun Policy

Provider errors are retried within the call according to its deadline and retry policy. An interrupted run resumes from the last checkpoint under the same scenario–condition–seed identity, retaining earlier partial artifacts. A new statistical draw uses a new seed. A provider blackout activates the batch circuit breaker and pauses remaining launches. Superseded records retain their lineage and status while the designated completed record supplies the aggregate.

K.6.3 Run Inclusion and Version Mapping

The main endpoint matrix contains 360 condition runs. Each paired analysis applies its declared population, metric eligibility, and frozen asset/version mapping. Run identity, completion status, scoring eligibility, selected checkpoint lineage, and replacement reason are retained as separate analysis fields.

Starter or evaluator changes produce a new frozen asset identity. The aggregation record joins each retained outcome to its designated task, source package when applicable, manifest, evaluator, and completion status. Superseded or invalidated records remain linked through the provenance record, supporting reconstruction of the final included set.

K.7 Prompts, Schemas, and Configuration Artifacts

K.7.1 Exact Condition Prompts

Relic separates what-action selection from the model calls used to perform, reflect on, or formalize a selected activity. B0 and B1 use an LLM to choose from a constrained candidate menu. B2, B3, B3-text, Transfer Text, and Transfer Exec use SDL for what-action selection; their model calls are module-specific.

Shared cognitive-module guard. For reflection, proposal drafting, proposal evaluation, and protocol synthesis, the system message combines the rendered workload/company brief, global grounding rules, the module name, and the module instruction. The common guard is:

```
You are the cognition of one agent in a simulated startup. You PROPOSE, REASON,
COMPOSE, EVALUATE and VERBALIZE. You do NOT change the world: the system
validates and executes. Never invent facts, objects, or actions that are not in
the provided context. Return ONLY valid JSON matching the schema.
```

The composition pattern is:

```
SYSTEM =
  COMPANY_BRIEF_TEMPLATE.format(company_name, product_name,
                                product_stage, product_purpose)
  + "\n\nGlobal rules:\n"
  + GLOBAL_GROUNDING_RULES
  + "\n\nCurrent module: " + module_name
  + "\n\n" + module_instruction

USER =
  agent_identity_for(agent, world)
  + "\n\n"
  + module_user(rendered_context)
```

B0/B1 action choice.

```
Choose the next intentional action for {agent_id} at tick {tick}.

{rendered_action_context}

Return ONLY JSON matching the action schema
(candidate_action and target_object_id MUST identify
one exact entry from candidate_options).
```

The model receives the available action menu, candidate options, valid targets, work state, recent events, task board, inbox, pull requests, test results, and other member-visible context. The returned action is validated against the candidate pool before execution.

Code editing. After an edit action has been selected, the code editor receives the complete target file, public evidence available to the member, the edit goal, and readable organizational rules when present:

```
Target file:
{target_object_id} -- {title}
{summary}

CURRENT FILE CONTENT:
{complete_file_content}

{current_build_error, if present}
{published_contract, if present}
```

```
{imported_public_API, if present}
{readable_protocol_rules, if present}

Edit goal:
{edit_goal}

Reason:
{rationale}

Known gaps:
{known_gaps}

Return the patch JSON with `edits` = the list of anchored
replacements that make this change. Quote each `search`
exactly from the file above; do not return the whole file.

{revision_brief, if an earlier patch was refused}
{anchor_correction, if a replacement did not apply}
```

Reflection and wish creation. The reflection user message is:

```
Reflect on the situation.
CONTEXT:
{JSON(reflection_context)}
```

Its full formation instruction is:

```
Reflect on the agent's own and the team's situation after recent events/episodes.
Output an honest self_assessment, team_assessment, blockers, and concrete
improvement_ideas (each with need_type/description/urgency/risk_if_unaddressed).
`need_type` says what kind of thing is missing, and the schema lists the kinds.
Choose protocol_need when the same avoidable mistake keeps happening and what is
missing is a standing rule everyone follows -- a checkable requirement on work
before it lands. Say in `description` what the rule should require and which
recurring failure it prevents; the organization writes and votes on the rule
from that. Choose policy_repair_need when an existing rule is the problem and
should be relaxed or repealed. The rules you are shown carry what each has been
holding up lately: a rule turning away most of the work while nothing ships is a
candidate however sound it reads, and one turning away nothing is not. Then say
in `description` how it should read instead -- write the replacement requirement
in full, keeping the part that was earning its keep and dropping the part
nothing can satisfy. A repair that only says the rule is too strict cannot be
applied: the wording you give is the wording work will be judged against
afterwards. When `what_has_been_failing` is present it is what the gates
actually said, in their words, with how the recent failures divide by kind.
Read it before you name a need: a failure that keeps arriving in the same words
is the clearest thing you have to reason from, and the need you name should
answer what those verdicts say, not what the situation feels like.
```

The main-study path maps and filters the structured `improvement_ideas` from this reflection output into wishes; there is no separate LLM wish-extraction call.

Wish → proposal. The module instruction is:

```
Turn a wish into a concrete, feasible proposal composed from EXISTING
actions/capabilities/artifacts. Do not assume adoption -- name
approval_required_from. A wish is worded as a suggestion -- "propose a
checklist", "create a matrix". proposed_solution is not: it becomes the rule
the organization is held to, so write what must be true of a piece of work,
checkable by someone who was not there. Never write it as a plan to make a rule.
```

The user message is:

```
Draft a proposal for this wish.
CONTEXT:
{
  "wish": {visible wish record},
  "available_actions": [registered action identifiers],
  "product_context": {visible product context}
}
```

Recurrent-wish clustering. The clustering system prompt is:

```
You read what the members of one organization have been asking for and name the
few themes that recur. A theme is worth a standing rule when the same avoidable
failure keeps costing the organization and a checkable requirement on work would
have caught it. Group the wishes: cite in `wish_numbers` the ones each theme
rests on, and name no theme that rests on fewer than two. Do not restate a rule
the organization has already adopted, and do not return two themes that a single
rule would cover -- each must stand on its own. `enforcement_rule` must be
checkable against a piece of work by someone who was not there: name what must be
true, not what people should care about. Return nothing rather than something
vague. Reply JSON only.
```

The user message provides already adopted rules and numbered wishes and asks for at most the configured number of themes worth a standing rule.

Pattern → candidate protocol.

```
Synthesize a candidate protocol from a repeated problem pattern. It is only a
proposal -- never adopt it yourself.
```

The corresponding user message provides the repeated pattern, occurrence count, candidate seed, and available actions.

Proposal evaluation.

```
Evaluate the proposal's feasibility/usefulness/risk/adoption. You only score and
suggest revisions; you cannot approve or reject.
```

with user message:

```
Evaluate this proposal.
PROPOSAL:
{JSON(proposal)}
```

The evaluator returns feasibility, usefulness, risk, adoption score, blocking issues, and suggested revision. Approval/adoption itself follows the governed review and approval path.

Stage	Recorded criterion
Reflection → wish	Each idea records need type, description, urgency, and risk; wishes retain source reflection, related objects/episodes, supporters, urgency, and status. Similar open wishes are merged.
Wish → proposal	Eligible when urgency is at least 0.85, it has at least two supporters, spans at least two episodes, or has founder endorsement; proposal provenance retains source wish/reflection/episode links.
Proposal → adoption	A protocol proposal with available provenance represents recurrence across at least two episodes or two wishes. Adoption requires at least three steps of review and two distinct approvers in the reported semi-automatic governance configuration.
Weak formation	Valid proposal/adoption, at least two post-adoption uses on distinct use units and steps, and at least 48 steps from adoption to the last qualifying use.
Strong evidence	Weak formation plus validated third-party enforcement, governed-state change, and independently evaluated outcome evidence.

Table 52: Wish-to-protocol formation and measurement criteria.

Wish-to-protocol formation criteria.

Formation-record fields. The per-run formation interface includes `protocol_count`, `weak_protocol_count`, `strong_protocol_count`, `time_to_emergence`, `repeated_protocol_use_rate`, `cross_context_protocol_reuse_rate`, `protocol_persistence_rate`, `valid_third_party_enforcement_rate`, `measurable_impact`, and `protocol_proposal_rate`. Post-adoption persistence, amendment/repair, and episode-transfer records provide lifecycle detail. The autonomous census joins these records by within-run lineage and applies the formation criteria in Table 52.

Readable protocol exposure in Transfer Text and Exec. Both treatments use the frozen `canonical_v2` package. Its semantic SHA-256 is `a627adfcf6c299d0fabb185345bbc01106ea6901380aa0fbfee37c830ad8cdb6`. The shared 3,148-character code-editor rule block has UTF-8 SHA-256 `9c505e567afab8853436c1e7cbc609e51928de2e29e64be299d4721bef09ab49`.

Transfer component	Text	Exec
Frozen package	`canonical_v2`	`canonical_v2`
Six readable summaries and code-editor rendering order	identical	identical
Compiled machine bindings	0	6
New target protocol formation/revision	disabled	disabled

Table 53: Content matching for the Text/Exec transfer comparison.

The six verbatim readable summaries are:

1. **Issue ownership before PR opening.** "Before `open_pr`, resolve the source branch exactly as the action handler does. If its commits link work, every explicit linked task must exist and have a non-empty `owner_id`. Every native Issue must have `owner_id` set, and every Task associated with that issue must also exist and be owned. A ProductArtifact issue has no inferred owner: it must resolve to at least one associated Task, and every associated Task must exist and be owned. A branch with no issue or task linkage is outside this guard. Assign every missing owner and retry; editing and committing remain allowed."
2. **Branch ownership.** "Before `open_pr`, resolve the source branch exactly as the action handler does and require `actor.id` to equal `branch.owner_id`. A non-owner is blocked without changing the branch; the recorded owner can open the request, so editing, committing, review, and merge repair lanes remain available."
3. **CI retry on changed state.** "Before `run_ci` or `ci_test`, resolve the pull request exactly as the action handler does. Block only when its latest CI result is failed and already attests the current

source-branch head against the current mainline commit list. A new branch head or changed mainline permits a retry; a `not_run` infrastructure result follows the native retry path."

4. **Current CI attestation at merge.** "Before `merge_pr` for a pull request carrying commits or patches, require approved status and no merge conflict; require `pr.commit_ids` to exactly equal the source branch commit list and `pr.patch_ids` to exactly equal the ordered flattening of those commits' `patch_ids`; require both `pr.ci_passed` and a latest passed CI result for the current branch head and require `ci_base_main_commit_ids` to equal the current `repo.main_commit_ids`. When the active profile exposes a deterministic candidate-tree digest, `ci_tree_hash` must exactly equal that digest; ordinary RepoLite uses the commit head and mainline base as its exact identity and does not require an unavailable digest. Missing, failed, conflicted, unsynchronized, or stale evidence blocks only merge; commit and `run_ci` remain available to repair it."
5. **Independent review.** "When the roster contains an agent other than the pull-request author, `review_pr`, `approve_pr`, and `formal_pr_review` by that author are blocked, and `merge_pr` requires at least one `approved_by` identifier that is both a current roster member and different from `author_id`. Unknown actor identifiers cannot review or approve. A one-agent roster may self-review so the workflow does not deadlock; `request_changes` and all repair actions remain allowed."
6. **Release-gate coverage.** "Before `publish_product_release`, resolve the release candidate exactly as the action handler does. Require approved status, the roster-aware `release_approval_met` check, no blockers, a non-empty `included_pr_ids` list whose requests all exist, are merged, and have `ci_passed`. Each candidate must carry the exact non-empty, duplicate-free gate profile selected by `release_gates_for(world)`. Each profile gate must have exactly one recorded passed result or its exact identifier must already be listed in `waived_gates`; unknown gates, waivers, and results are rejected. Run readiness, collect the required approvals, and repair blockers before retrying publish."

K.7.2 Configuration Artifacts

Frozen workload manifests and brief templates define the public task inputs, entry points, progressive contract steps, and public-test commands. Machine-readable schemas define the member and organizational state, proposal and protocol fields, and runtime bindings. The exact model prompts and transferred rule summaries above identify the text consumed by the corresponding cognitive modules.

Transfer metadata links the source package, content digest, freeze record, carrier documents, role mapping, and target reset state to the treatment configuration. Appendix G.4.3 specifies the `capability_transfer` fields used to inspect the injected package and its later execution records.

L Human–Organization Interfaces

Relic primarily studies whether autonomous agent organizations can accumulate persistent organizational capabilities. Once such an organization can maintain its own workstreams, responsibilities, review processes, protocols, evidence, and institutional state, however, a second problem appears: *how should a human participate in an organization whose internal operating representation and execution tempo are increasingly machine-native?*

We explore three interface designs for human participation in a continuously operating agent organization.

L.1 Why Direct Human Participation Becomes Difficult

Our initial design treated the human as another organizational member. This preserves an appealing symmetry: human and agent members communicate through the same channels, observe organizational objects available to their roles, and participate in the same workflows. In practice, this design exposes a representation mismatch.

Relic maintains structured organizational state including task ownership, review and CI status, active protocols, approvals, episodes, evidence, commitments, requests, and institutional memory. These representations are useful for agent decision making and organizational governance, but they are not necessarily the representation in which a human wants to understand ongoing work. A person entering the organization may first have to determine which workstreams matter, what is blocked,

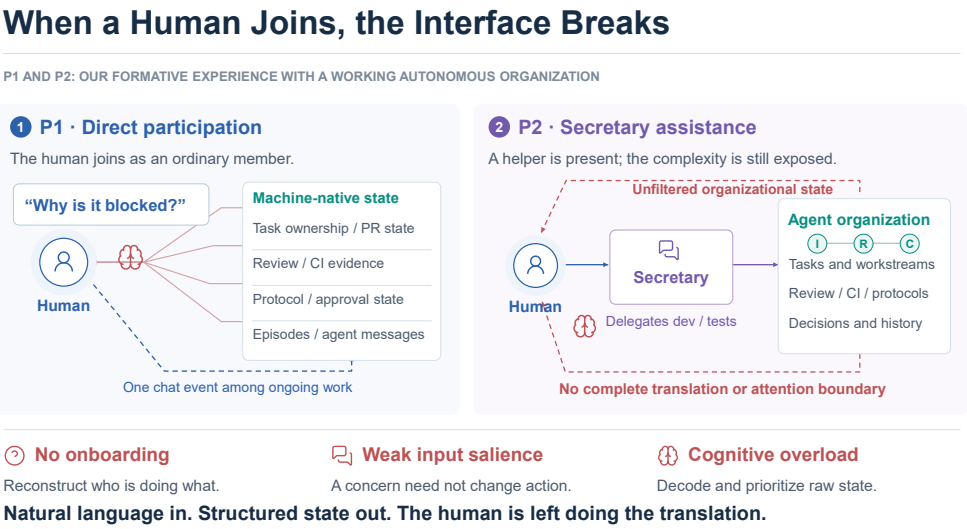

Figure 7: **Interface progression before P3.** In P1, the human joins the autonomous organization directly and must interpret machine-native organizational state. In P2, a secretary assists with delegated work, but substantial organizational complexity remains directly exposed. These experiences motivate a liaison-first abstraction boundary in P3.

which evidence is current, who owns a decision, and whether an apparent problem has already been handled elsewhere.

Conversely, a natural-language message from a human is only one event in a continuously operating organization. Asking "Why is this blocked?" does not itself guarantee that the organization will stop, that the relevant agent will immediately answer, or that no other authorized member will act before the human finishes considering the issue.

A second prototype introduced a secretary agent that could help the human delegate implementation and verification work. This reduced some execution burden, but much of the underlying organizational state remained directly exposed. The human still had to interpret the machine-native organization and decide which internal objects deserved attention.

Figure 7 summarizes the interface progression that motivated a stronger human–organization abstraction boundary.

L.2 Moving the Human–Organization Abstraction Boundary

We organize the design space into three interface levels. The progression changes how organizational complexity is presented to the human.

P1: Direct organization. The human directly navigates organizational state and communicates with organizational members. This maximizes direct visibility into the organization but places the burden of orientation, interpretation, and attention allocation on the human.

P2: Transparent liaison. A secretary assists with communication and delegated work while the underlying agents, workstreams, and organizational structures remain directly visible. The secretary reduces some execution burden, but the human still operates close to the machine-native representation and remains responsible for understanding a substantial fraction of organizational state.

P3: Liaison-first interaction. The secretary becomes the primary human-facing interface. The organization is not simplified internally. Instead, its state is summarized, translated, and selectively surfaced to the human. Detailed organizational evidence and provenance remain available on demand.

Hybrid human–organization interface levels		
	Primary interface	**Main interaction property**
P1	Organization directly	Human interprets organizational state and communicates with members directly.
P2	Secretary + visible organization	Secretary assists, but substantial organizational complexity remains exposed.
P3	Secretary first; organization on demand	Secretary translates state and intent while preserving access to evidence and provenance.

Table 54: Three human–organization interface levels.

L.3 P3: A Liaison-First Hybrid Human–Agent Organization

P3 places the secretary at the human-facing boundary while preserving the autonomous organization. It combines three concurrently operating components: a full autonomous Relic organization, a human member with a human-facing office, and the shared organizational workflow.

Autonomous Relic organization. The upper-right component of Figure 8 is the autonomous Relic organization. It continues to maintain its own workstreams, tasks, responsibilities, review processes, protocols, evidence, and institutional memory. Other organizational members continue working independently of the human-facing interaction.

The organization may discover problems, aggregate evidence, form recommendations, execute unrelated work, and update its institutional state without waiting for continuous human direction.

Human office. The human is represented as a member of the larger organization and retains their own execution capacity. The human office contains the human, the secretary, and delegated execution agents that can carry out work belonging to the human member's responsibilities. For example, a human may specify a desired behavior and constraints, after which the secretary can coordinate implementation or verification work on the human's behalf.

The secretary consequently has two distinct responsibilities:

1. **organizational liaison:** translate organizational state into human-readable summaries and route human intent back into organizational objects; and
2. **human-work coordinator:** decompose and delegate work that belongs to the human member's own organizational responsibilities.

The secretary coordinates the human-facing interface and delegated work while shared governance and the organization's persistent decision processes continue to govern organization-wide work.

Shared organizational workflow. Artifacts produced through the human office re-enter the same organizational workflow as work produced by other members. Code, documents, tests, and other artifacts remain subject to applicable review, CI, evidence requirements, and active protocols.

Together, these components form a *hybrid human–agent organization*: the agent organization continues operating autonomously, while the human participates through a human-facing abstraction layer and retains delegated execution capacity.

L.4 Bidirectional Translation

The secretary acts as a bidirectional abstraction layer between two different representations of work.

Organization to human. The autonomous organization may contain many simultaneously active tasks, reviews, failures, protocols, and commitments. The secretary compresses this state into human-facing updates. We distinguish three forms of communication:

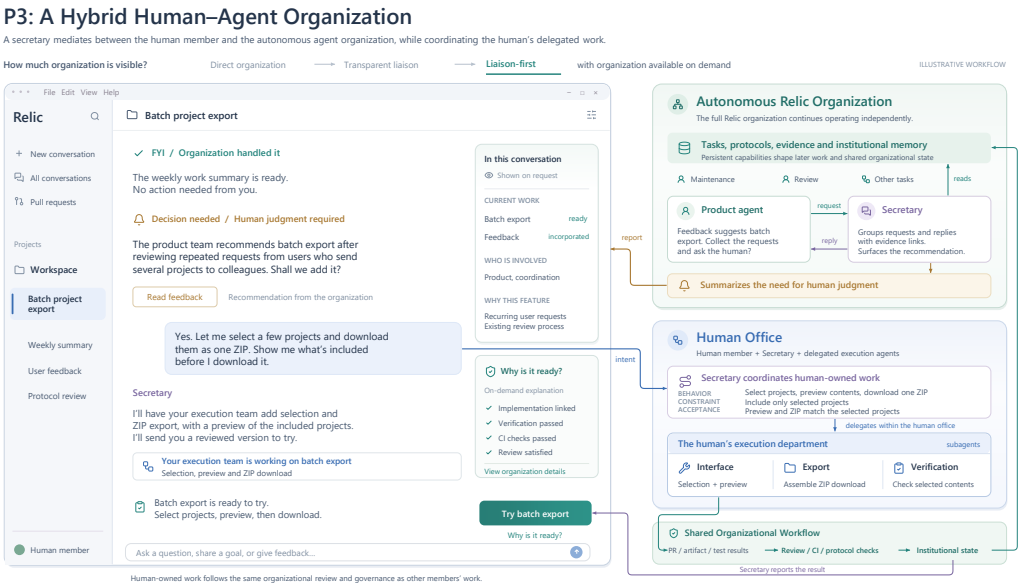

Figure 8: **P3: a liaison-first hybrid human–agent organization.** The autonomous Relic organization continues operating independently (upper right). The human interacts primarily through a secretary, which summarizes organizational state, surfaces items requiring human judgment, and translates human intent into organizational work. The human also has delegated execution agents inside a human office. Human-owned artifacts subsequently enter the same shared review, CI, protocol, and institutional workflow as other organizational work.

- **informational updates**, for work that the organization has handled without requiring human action;
- **decision requests**, for cases in which human judgment, preference, authorization, or unavailable information is required; and
- **on-demand explanation**, through which the human can request the evidence, provenance, review history, or organizational trace behind a summary.

P3 therefore uses progressive disclosure. The default representation is concise, while the underlying organizational state remains inspectable when the human needs additional evidence.

Human to organization. Human input is expressed in ordinary language but may imply different organizational actions. For example, "I do not trust this result" may indicate a request for additional evidence, whereas "always review these changes" may express a persistent policy-level intention rather than a one-time task.

The secretary can translate such input into structured organizational objects such as tasks, concerns, review requests, constraints, acceptance criteria, or protocol proposals. The resulting object then follows the organization's ordinary validation, approval, and execution path.

L.5 Human Attention as an Organizational Resource

A highly autonomous organization should not require the human to inspect every internal action. At the same time, aggressive filtering can conceal important disagreement or make consequential decisions difficult to inspect.

This creates an organizational problem of deciding *what* deserves human attention. Routine and reversible issues may be handled internally; recurring but non-urgent issues may be summarized periodically; high-impact, irreversible, or unresolved decisions may require direct escalation. Escalation policy therefore becomes part of the human–organization interface rather than only a notification preference.

The completed walkthroughs expose a second dimension of the same problem: *when* human attention can arrive. An organization must decide not only which events deserve human judgment, but also whether its own execution should wait, slow down, continue autonomously, or defer irreversible actions while that judgment is pending.

L.6 Formative Paired Walkthrough Protocol

Four members of the author team completed formative paired walkthroughs of P2 and P3 after implementation. None had participated in the design or implementation of Relic or the two interfaces. Each explored ongoing work, inspected blockers and evidence, communicated a requested change, and responded to decisions requiring human judgment.

L.7 Paired Internal Walkthrough Results

P3 reduced organizational interpretation burden. Across the four paired walkthroughs, the liaison-first P3 interface was consistently easier to follow than P2. In P2, the secretary could assist with delegated work, but the evaluator still had to inspect and interpret substantial machine-native organizational state before deciding how to intervene. Active workstreams, agent activity, pending reviews, protocol state, evidence, and other internal objects remained directly salient to the human interaction.

In P3, the secretary absorbed much of this interpretation burden. It summarized ongoing organizational state, surfaced the subset of issues that required attention, and allowed the evaluator to request underlying evidence or provenance when needed. The organization itself remained unchanged internally; what changed was the human-facing representation. The interaction therefore shifted from navigating the organization toward judging the decisions and exceptions that the organization surfaced.

Human deliberation became the remaining bottleneck. A recurring observation was that improved legibility did not remove the difference in operating speed between humans and autonomous agents. While an evaluator was reading a summary, inspecting evidence, or considering whether to approve, reject, or redirect work, other agents could continue operating elsewhere in the organization.

In some interactions, useful work advanced while the human was still deliberating. When another agent had sufficient authority over a pending item, that item could also be resolved before the human responded. The resulting friction was therefore no longer primarily about understanding the state of the organization. It was about whether human judgment could arrive on the same timescale as the organization that was waiting for, or acting around, that judgment.

This makes the timing of intervention a separate interface variable. A human may understand the issue perfectly and still be too slow to participate in every decision at machine execution speed.

Pausing provides control but changes the workflow. The interface can pause the organization while the human deliberates. This is a practical control mechanism: it prevents the relevant organizational state from continuing to evolve while the human reads evidence and forms a judgment.

However, pausing also changes the interaction regime. A continuously operating autonomous organization becomes temporarily synchronized to human decision speed. If every consequential decision requires such a pause, the resulting system behaves less like an autonomous organization with intermittent human oversight and more like a workflow that repeatedly waits for a human operator.

Representation mismatch and temporal mismatch are distinct. The P2/P3 comparison separates two problems that can otherwise be conflated.

The first is *representation mismatch*: directly exposing machine-native organizational state makes it costly for a human to understand what is happening. P3 substantially reduces this burden through liaison-first summarization and progressive disclosure.

The second is *temporal mismatch*: even when the relevant state is presented clearly, human deliberation may remain slower than the organization that is acting around that decision. Better summarization

reduces the time needed to understand a situation, but it does not make human reasoning operate at agent execution speed.

A human–organization interface must therefore decide both *what* should be escalated and *how the organization should behave while judgment is pending*. Possible policies include waiting only for irreversible actions, continuing reversible work, slowing selected workstreams, assigning decision deadlines, or allowing another authorized member to proceed after a timeout.

L.8 Authority and Interaction Records

Visibility, permissions, and decision rights follow the human and secretary roles. Human-owned artifacts re-enter the shared review, CI, and protocol workflow. Interaction records connect the human's original instruction, the secretary's structured interpretation, the resulting work or governance object, and subsequent execution.

The human can request the organizational evidence and provenance behind a summary. This record preserves the path from a human-facing decision to its source evidence and organizational consequences while the shared governance process remains active.

M External Software Benchmark: CooperBench

CooperBench evaluates joint implementation of interacting features in existing software repositories (Khatua et al., 2026). We evaluate the complete two-member Relic architecture on the full benchmark against released Solo, Peer/coop+git, and hierarchical Team trajectories. We additionally evaluate a fixed 48-pair same-model subset as a controlled test of coordination loss and recovery.

M.1 Full-Benchmark Scope and Success Criterion

The full CooperBench evaluation space contains 652 exact interacting-feature pairs. A pair succeeds when the delivered candidate passes the official tests for both features (`both_passed=true`); each exact pair is one scoring unit.

Benchmark-validity auditing ran alongside Relic execution. Once an issue family was established, all affected exact pairs were entered into the validity ledger. Relic completed all 477 pairs in the final validity set; the Raw and definite-defect matched analyses use the corresponding exact-pair identities with formal official verdicts.

We report three validity policies:

- **Raw.** No benchmark-validity exclusion is applied. The matched Relic analysis uses the 616 exact pairs with formal official verdicts.
- **Definite-defect exclusion.** We remove only pairs affected by a directly established specification–verification mismatch, unpublished mandatory interface or representation, explicit behavioral conflict, or established cross-feature incompatibility. This policy excludes **111 unique exact pairs from 21 issue families**, leaving 541 pairs.
- **Ambiguity-aware exclusion.** We additionally remove pairs for which the public contract does not uniquely determine a behavior that the evaluator treats as mandatory. The complete audit contains **175 unique exact pairs from 33 issue families**, leaving a **477-pair** validity-filtered benchmark.

For released systems, trajectories for all 652 pair identities are available. We therefore recompute their results on the same fixed 652-, 541-, and 477-pair universes. Only an official PASS contributes to the numerator; a non-passing or missing formal verdict does not further shrink the policy-defined denominator.

M.2 Two-Member Relic Adaptation

The adapter places two feature owners in one shared organization. Each starts from its assigned feature brief and works through a private workspace and branch. Requirements and local findings

reach the other member through explicit messages or shared artifacts. Pair-specific run identities keep workspaces, patches, messages, and organizational state separate across concurrent pairs.

The B3-2 configuration retains SDL, governed protocol formation, and executable bindings. SDL selects feasible work, communication, review, and governance actions; the model produces code and messages. Members can propose rules from visible coordination failures, submit them for validation and approval, and apply adopted rules to subsequent work.

Joint delivery follows owner-local editing and public testing, commits and pull requests bound to a code revision, peer review, integration replay, and synchronization to shared mainline. Both members attest to the same mainline digest and export the same joint patch to their submission slots. The official evaluator then applies the two native feature test suites to that delivered candidate.

Public-probe receipts retain the probe, associated public requirements, candidate and relevant baseline statuses, and bounded execution errors and output. Fixed patch guards and artifact-identity checks implement the adapter, while learned protocols retain their own proposal, adoption, and execution records. The final official evaluation is linked to the exact pair identity, model, execution settings, and adapter revision.

Initial adapter development used a separate task to check coordination, integration, and submission before the reported evaluations. The full-benchmark Relic runs use Claude Opus 4.6 with high reasoning effort.

M.3 Baseline Configurations

The full-benchmark comparison uses three released GPT-5.5-hao references (CooperBench, 2026b;a). **Solo** assigns both interacting features to one coding agent. **Peer / coop+git** assigns one feature to each of two peers without a permanent lead. **Team no-proto** uses the released lead–member hierarchy with shared task state and scratchpad.

Across the released runs, these structures exhibit a stable coordination hierarchy:

$$\text{Peer} < \text{Solo} < \text{Team}.$$

Splitting interacting features across plain peers reduces success relative to Solo, whereas the structured lead–member Team exceeds Solo. Relic retains the peer topology with no permanent lead and adds persistent organizational state and executable governance to that structure.

The separate 48-pair controlled comparison uses Claude Opus 4.6 with high reasoning effort for Relic, Official Peer, and Official Solo. Official Peer assigns one interacting feature to each peer; Official Solo assigns both features to one agent. Relic retains the same model and two-peer feature split.

M.4 Benchmark-Validity Audit

M.4.1 Hidden Tests Versus Hidden Requirements

We distinguish withheld *test instances* from withheld *requirements*. A benchmark may hide concrete test code, input instances, edge cases, reference implementations, and evaluator data. However, for a hidden evaluation to measure satisfaction of the stated task, the acceptance semantics exercised by those tests must be supported by the public task contract available to the evaluated system. In short, *test instances may be hidden, but task requirements should not be*.

For example, if a public specification requires correct handling of all valid JSON inputs, a previously unseen valid JSON instance is an appropriate hidden test. By contrast, if the public specification requires JSON and CSV support while the evaluator additionally requires YAML, the evaluator has introduced an unpublished requirement rather than merely an unseen test instance. Likewise, a requirement to produce an error log does not by itself justify an evaluator that accepts only one unpublished exact log string.

Our audit therefore asks whether behavior enforced by the official evaluator is supported by the public specification.

M.4.2 Audit Scope and Outcome Distribution

The audit identifies **33 shared specification–verification issue families** affecting **175 distinct exact feature pairs**. The sum of family-level pair incidences is 186 because 11 exact pairs instantiate two issue families; deduplication by exact pair identity yields 175 affected scoring units.

Once an issue family is admitted, it applies to every affected exact pair. In the complete 175-pair ambiguity-aware exclusion set, the raw Relic states comprise **21 PASS, 118 FAIL, and 36 pairs without an official verdict**. The 111-pair definite-defect subset contains **5 PASS, 76 FAIL, and 30 pairs without an official verdict**. The validity audit asks whether a scoring unit can distinguish failure to implement the public requirement from failure to satisfy an unpublished or materially under-specified evaluator assumption.

All original evaluator outcomes and experimental artifacts are retained. Validity exclusion changes only whether an exact pair contributes to the corresponding aggregate score; it does not rewrite its raw outcome.

M.4.3 Validity Evidence Levels

The six failure classes below describe *what kind* of contract problem is present. Independently, each issue family is assigned one of two evidence levels: a *definite defect* or a *specification ambiguity*.

Definite defects. A definite defect is directly established from the public specification and evaluator behavior: an explicit specification–verification mismatch, an unpublished mandatory interface or internal structure, an unpublished exact representation, an explicit behavioral conflict, or an established cross-feature incompatibility. The audit contains **21 definite-defect families covering 111 unique exact pairs**.

Specification ambiguities. A specification ambiguity occurs when more than one behavior remains consistent with the public contract, while the evaluator accepts only one such interpretation. The audit contains **12 ambiguity families covering 71 unique exact pairs**. Seven of those pairs are already affected by a definite-defect family, so the ambiguity families contribute 64 additional unique pairs:

$$111 + 71 - 7 = 175.$$

Thus, the definite-defect policy excludes 21 families / 111 pairs, whereas the ambiguity-aware policy excludes all 33 families / 175 pairs.

Class	Specification–verification failure	Families D/A	Primary pairs	D coverage	A coverage
C1	Public contract and evaluator acceptance are inconsistent	8 / 0	47	47	0
C2	Hidden interface, internal structure, or input-domain requirement	5 / 1	33	30	5
C3	Callback protocol is not defined by the public contract	0 / 3	18	0	20
C4	Behavioral scope or boundary semantics are not determined	1 / 8	45	3	48
C5	Exact output representation is not publicly specified	3 / 0	20	21	0
C6	Cross-feature contract or retained-test incompatibility	4 / 0	12	12	0
Across-class total / union		**21 / 12**	**175**	**111**	**71**

Table 55: **CooperBench validity taxonomy and evidence levels.** D denotes a definite defect and A denotes specification ambiguity. "Primary pairs" assigns each affected pair to one class in a fixed C1–C6 order so that the column sums to 175. D/A coverage reports unique pair coverage within each class and is not additive across classes because an exact pair may instantiate multiple issue families.

C1, C5, and C6 contain only definite defects. C3 contains only specification ambiguities. C2 contains five definite hidden-contract requirements and one private-interface ambiguity. C4 contains one definite unpublished behavior and eight boundary-semantics ambiguities.

M.4.4 Complete Issue-Family Inventory

Table 56 lists every admitted issue family. Because CooperBench scores exact feature pairs rather than isolated features, a defect associated with one repeatedly paired feature can propagate to several scoring units.

Table 56: **Complete CooperBench specification–verification issue-family audit.** D denotes a definite defect; A denotes specification ambiguity. "Pairs" is the number of exact feature pairs affected by the family and may overlap across rows.

ID	Class / tier	Repository / task / feature	Observed contract problem	Pairs
R01	C1 / D	`go_chi / 56 / f1`	The public example permits "GET, HEAD" as one Allow-header value, whereas the evaluator indexes two separate values from `Header.Values("Allow")`.	4
R02	C2 / D	`go_chi / 27 / f3`	The evaluator directly requires internal logging symbols that are not part of the publicly specified interface.	3
R03	C4 / A	`go_chi / 26 / f3`	The public contract does not fully determine the version-selection entry point or fallback behavior for an unknown selector or role; the evaluator selects one specific behavior.	3
R04	C4 / A	`outlines / 1371 / f2`	The public contract does not specify the behavior for unregistering a nonexistent filter; the evaluator requires `unregister_filter("non_existent")` to succeed without raising.	3
R05	C4 / D	`outlines / 1371 / f4`	The public task specifies JSON/CSV behavior, while the evaluator additionally requires YAML support.	3
R06	C4 / A	`outlines / 1655 / f4`	The public contract permits colon or hyphen separators but does not determine whether they may be mixed; the evaluator rejects mixed forms.	9
R07	C6 / D	`outlines / 1706 / f1`	The new feature requires wording that includes both Transformers and MLXLM, while retained tests from interacting features still match the older Transformers-only wording.	7
R08	C1 / D	`tiktoken / 0 / f9`	The public contract names the parameter `compress`, whereas the evaluator calls `compression`.	9
R09	C2 / D	`tiktoken / 0 / f10`	The public task specifies LRU/`use_cache` behavior, while the evaluator additionally requires an assignable `enc.cache` attribute that is not published in the contract.	9
R10	C3 / A	`tiktoken / 0 / f7`	The validator protocol is underspecified: the contract does not determine whether validation returns a Boolean or raises directly, while the evaluator fixes a False-returning predicate protocol.	9

Continued on next page

ID	Class / tier	Repository / task / feature	Observed contract problem	Pairs
R11	C5 / D	click / 2068 / f10	The public task requires error handling, but the evaluator additionally requires the unpublished exact text "Editing failed".	11
R12	C1 / D	click / 2956 / f7	The public example/model input uses an underscore-form parameter name, whereas the hidden acceptance condition requires the hyphenated CLI spelling.	7
R13	C4 / A	click / 2956 / f3	The public contract specifies ValueError for negative batch size but does not determine the exception class for non-multiple input; the evaluator requires TypeError.	7
R14	C6 / D	click / 2068 / paired features	One feature explicitly requires wait(timeout=timeout), while retained mocks in interacting-feature tests expose a narrower wait signature.	3
R15	C6 / D	click / 2068 / f1--f8	The new feature moves to edit_files, while the interacting feature's retained tests still depend on the older internal edit_file API.	1
R16	C1 / D	jinja / 1559 / f3	The public task requires retries beginning at priority 0, while the evaluator also requires the priority-0 case to invoke the operation only once.	9
R17	C4 / A	jinja / 1465 / f5	The public task discusses None, empty-string, and falsy values but does not define grouping semantics for a completely missing attribute; the evaluator requires one particular treatment.	9
R18	C3 / A	HF datasets / 3997 / f2,f3	The custom_criteria callback arguments are not specified. The evaluator accepts a single feature argument but rejects the alternative (name, feature) protocol.	7
R19	C4 / A	HF datasets / 3997 / f4	The public contract does not specify whether a JSON-formatted string supplied as an update value should be automatically deserialized into a dictionary; the evaluator requires that behavior.	4
R20	C3 / A	pillow / 68 / f4	The public task specifies an error_handler callable and high-level strategy but not its exact arguments or return protocol; the evaluator fixes (error,key,value) and specific fallback/skip semantics.	4
R21	C5 / D	pillow / 68 / f5	The public task requires relevant logging, whereas the evaluator accepts a particular unpublished sentence form such as "Orientation value: 3".	4
R22	C1 / D	pillow / 290 / f4	The public contract permits fallback to the maximum color count when the MSE threshold cannot be met, whereas the evaluator imposes strict error ordering. The reference image remains above the stated threshold even at 256 colors.	4
R23	C5 / D	llama_index / 17244 / f4	The public force_mimetype contract does not require the exact representation, while the evaluator does.	6
R24	C4 / A	llama_index / 18813 / f5	The public task mentions trimming but does not define it as removal of zero bytes from both ends of the raw binary payload; the evaluator fixes that interpretation.	5
R25	C2 / D	llama_index / 18813 / f6	For format=mp3 without an explicit mimetype, the evaluator requires unpublished BytesIO.mimetype and BytesIO.as_base64 attributes.	5
R26	C2 / D	dirty_equals / 43 / f6	The public task defines an issuer parameter, while test collection directly requires an unpublished class-subscript API such as IsCreditCard["Visa"].	8
R27	C4 / A	dirty_equals / 43 / f7	The public contract does not determine whether an invalid hash algorithm should fail at construction or evaluate to false during comparison; the evaluator requires a constructor-time ValueError.	8
R28	C2 / D	react_hook_form / 153 / f6	The public contract does not make the otherwise required onValid argument optional, while the evaluator supplies undefined and requires successful handling.	5
R29	C1 / D	react_hook_form / 153 / f5	The public task explicitly permits either a configuration object or an additional parameter, while the evaluator accepts only a third-argument options object.	5
R30	C1 / D	react_hook_form / 85 / f2	The public contract permits either formState.isLoading or a dedicated state representation, while the evaluator accepts only the former.	4
R31	C2 / A	dspy / 8635 / f4	The public task permits None to be intercepted at the call site, while the evaluator directly invokes a private helper with None; the private-helper input contract is not specified.	5
R32	C1 / D	dspy / 8587 / f3	The public task describes timestamps as optional, while the evaluator requires specific private _t0/_t_last mappings and fixed output keys.	5
R33	C6 / D	pillow / 25 / f1--f2	Feature 1 requires readonly state to be preserved when saving to a different file, whereas Feature 2 requires the default preserve_readonly=False behavior to ensure that the destination is writable before saving. The same default save-as operation therefore cannot simultaneously satisfy both public feature contracts.	1

Why 33 issues affect 175 scoring units. CooperBench scores exact feature pairs rather than isolated features. Consequently, a contract defect associated with one repeatedly paired feature propagates to every scoring unit containing that feature. For example, the unpublished exact-output requirement in click/2068/f10 affects 11 pairs; the tiktoken parameter-name and hidden-cache issues each affect nine; the jinja priority conflict affects nine; and the outlines separator ambiguity affects nine.

The 175 excluded pairs arise from **33 shared feature- or pair-level contract issues propagated through CooperBench's pairwise composition**.

M.5 Released-Baseline Sensitivity

The definite-defect policy leaves $652 - 111 = 541$ exact pairs, while the ambiguity-aware policy leaves $652 - 175 = 477$. We apply the identical pair sets to all released trajectories.

Validity policy	N	GPT-5.5 Solo	Peer / coop+git	Team no-proto
Raw	652	362/652 (55.5%)	329/652 (50.5%)	**403/652 (61.8%)**
Definite-defect	541	339/541 (62.7%)	305/541 (56.4%)	**380/541 (70.2%)**
Ambiguity-aware	477	316/477 (66.2%)	282/477 (59.1%)	**354/477 (74.2%)**

Table 57: **Released GPT-5.5-hao CooperBench trajectories under three benchmark-validity policies.** Only official PASS outcomes enter the numerator; the denominator is fixed by the validity policy.

The validity policy changes absolute pass rates but not the structural ordering:

$$\text{Peer / coop+git} < \text{Solo} < \text{Team no-proto}.$$

In the released CooperBench settings, plain peer cooperation is therefore the weakest of the three structures: splitting the interacting features across peers reduces performance relative to Solo, whereas the structured lead–member Team exceeds Solo.

M.6 Relic Results and Matched-Pair Sensitivity

Within each validity policy, Relic, Solo, Peer, and Team are compared on exactly the same exact-pair identities with formal Relic verdicts.

After completion of all 477 ambiguity-aware-valid pairs, the matched comparison contains 616 verdict-bearing identities under the Raw policy and 535 under the definite-defect policy.

Validity policy	Relic	Peer	Solo	Team no-proto
Raw ($n = 616$)	388/616 (63.0%)	317/616 (51.5%)	350/616 (56.8%)	**391/616 (63.5%)**
Definite-defect ($n = 535$)	**383/535 (71.6%)**	303/535 (56.6%)	337/535 (63.0%)	378/535 (70.7%)
Ambiguity-aware ($n = 477$)	**367/477 (76.9%)**	282/477 (59.1%)	316/477 (66.2%)	354/477 (74.2%)

Table 58: **Matched-pair CooperBench sensitivity.** Within each row, all systems are scored on exactly the same pair identities with formal Relic verdicts under that policy.

Raw comparison: lifting peer coordination to the Team regime. On the unfiltered matched set, Relic reaches 63.0% on 616 identities, compared with 51.5% for the released Peer reference and 56.8% for Solo on the same identities. This is a +11.5 percentage-point gain over Peer and a +6.2-point gain over Solo, with Relic 0.5 percentage points below the hierarchical Team reference on the same pairs (63.5%).

The released CooperBench baseline ordering is

$$\text{Peer} < \text{Solo} < \text{Team}.$$

Peer/coop+git is the weakest released coordination setting, whereas the lead–member Team is the strongest. Relic retains two peer feature owners with no permanent lead and reaches essentially the same performance regime as the specialized hierarchical Team on the unfiltered matched set.

Sensitivity to benchmark validity. Under the definite-defect policy, Relic reaches 71.6%, compared with 56.6% for Peer, 63.0% for Solo, and 70.7% for Team on the same 535 pair identities. The

corresponding differences are +15.0 points over Peer, +8.6 points over Solo, and +0.9 points over Team.

Under the ambiguity-aware policy, Relic reaches 76.9% on the complete 477-pair validity set, compared with 59.1% for Peer, 66.2% for Solo, and 74.2% for Team. The corresponding differences are +17.8 points over Peer, +10.7 points over Solo, and +2.7 points over the hierarchical Team reference.

Relic's advantage increases under stricter validity policies. On the raw matched set it exceeds Peer by 11.5 points and Solo by 6.2 points while remaining within 0.5 points of Team; under the definite-defect and ambiguity-aware policies it exceeds Team by 0.9 and 2.7 points, respectively.

From the weakest released topology to Team-level performance. The released CooperBench hierarchy shows a coordination penalty for plain peers: splitting interacting features across two peers performs worse than assigning both to one agent, while a specialized lead–member hierarchy is required to exceed Solo.

Relic keeps the peer feature-owner structure and adds persistent organizational state, governed protocol formation, and executable coordination rules. This moves the peer topology from the weakest released CooperBench setting to Team-level performance on the unfiltered matched set, above Team under the definite-defect policy, and further above the hierarchical reference on the ambiguity-aware validity set.

M.7 Frozen 48-Pair Same-Model Coordination Check

The same-model coordination check uses a fixed 48-pair subset spanning 30 task instances and 12 repositories, selected independently of Relic outcomes. Relic, Official Peer, and Official Solo use the same Claude Opus 4.6 model, high reasoning effort, and identical pair identities.

Table 59: **Frozen CooperBench same-model evaluation pairs.**

#	Repository	Task	Pair
1	dottxt_ai_outlines	1371	F1/F2
2	dottxt_ai_outlines	1655	F1/F3
3	dottxt_ai_outlines	1655	F6/F7
4	dottxt_ai_outlines	1655	F7/F10
5	dspy	8563	F1/F4
6	go_chi	27	F3/F4
7	go_chi	27	F2/F4
8	huggingface_datasets	7309	F1/F2
9	llama_index	17070	F1/F2
10	llama_index	17244	F5/F6
11	llama_index	17244	F2/F6
12	openai_tiktoken	0	F4/F8
13	openai_tiktoken	0	F1/F5
14	pallets_click	2800	F1/F4
15	pallets_click	2800	F1/F2
16	pallets_jinja	1621	F6/F10
17	pallets_jinja	1621	F1/F6
18	pillow	25	F1/F5
19	pillow	25	F1/F4
20	pillow	68	F1/F5
21	pillow	290	F3/F5
22	pillow	290	F2/F3
23	react_hook_form	153	F2/F6
24	react_hook_form	153	F1/F3
25	samuelcolvin_dirty_equals	43	F2/F3
26	samuelcolvin_dirty_equals	43	F2/F4
27	typst	6554	F2/F6
28	typst	6554	F1/F3
29	dottxt_ai_outlines	1706	F4/F6
30	dottxt_ai_outlines	1706	F5/F6
31	dspy	8394	F3/F4
32	dspy	8394	F3/F5
33	dspy	8587	F1/F4
34	dspy	8587	F2/F3
35	dspy	8635	F1/F4

Continued on next page

#	Repository	Task	Pair
36	dspy	8635	F4/F6
37	go_chi	26	F1/F2
38	go_chi	26	F2/F4
39	go_chi	56	F1/F5
40	go_chi	56	F2/F3
41	huggingface_datasets	3997	F1/F2
42	huggingface_datasets	6252	F4/F6
43	llama_index	18813	F1/F5
44	pallets_click	2068	F1/F4
45	pallets_click	2956	F1/F8
46	pallets_jinja	1465	F1/F7
47	pallets_jinja	1559	F1/F8
48	react_hook_form	85	F3/F4

M.7.1 Same-Model Results

System	Model	Structure	Pass / 48
■ **Relic (B3-2)**	Claude Opus 4.6 (high)	Two peer feature owners; no fixed lead	**29/48**
Official Solo	Claude Opus 4.6 (high)	One agent receives both interacting features	26/48
Official Peer	Claude Opus 4.6 (high)	Two peers split the interacting features	13/48

Table 60: **Frozen 48-pair same-model CooperBench coordination check.**

Official Peer drops from Solo's 26/48 to 13/48 when the interacting features are divided across peers. Relic retains the same model, reasoning setting, two-peer feature split, and absence of a permanent lead, yet reaches 29/48. Thus Relic not only recovers the coordination loss observed in Official Peer but exceeds the same-model Solo reference on this controlled subset.

Together with the full-benchmark comparison, the 48-pair experiment provides a direct same-model control over the peer coordination setting, while the full benchmark evaluates the same peer-organizational design at benchmark scale against the released Solo, Peer, and hierarchical Team references.

N External Software Benchmark: ProgramBench

ProgramBench (Yang et al., 2026) evaluates a frozen executable protocol layer attached to the native Bash-action interface of a single mini-SWE-agent coding agent.

N.1 System Configuration

The baseline is official mini-SWE-agent (Yang et al., 2024). The treatment adds a stateful protocol sidecar around the same single-agent Bash interface. The model chooses actions, edits the candidate code, maintains the conversation history, and decides when to submit. Before dispatch, the sidecar checks the proposed action and accumulated execution evidence. Its response allows the action, requests current evidence, or refuses a transition that violates an executable rule.

The treatment uses mini-SWE-agent 2.4.6, ProgramBench 1.2.4, and GPT-5.6 with `xhigh` reasoning effort and all-turn context. Protocol content remains frozen throughout the target runs. Binding rules and advisory rules are identified separately in Table 62.

N.2 Task Selection and Fixed Evaluation Set

Adapter integration was developed on a separate demonstration task. We randomly sampled 25 ProgramBench tasks, fixed their identities, and evaluated both systems on this paired set.

Fixed random sample of 25 ProgramBench tasks		
1. `zip-password-finder`	10. `tig`	19. `muffet`
2. `tui-journal`	11. `parqeye`	20. `elfcat`
3. `oranda`	12. `igrep`	21. `svd2rust`
4. `scc`	13. `loop`	22. `argc`
5. `rustowl`	14. `dropbear`	23. `code-minimap`
6. `datasurgeon`	15. `bartib`	24. `tty-clock`
7. `xh`	16. `gdal`	25. `nomino`
8. `rust-sloth`	17. `proj`	
9. `7zip`	18. `quinn`	

Table 61: **ProgramBench evaluation set.** The task identifiers are randomly sampled once and then held fixed for both systems.

The sample contains 17 Rust tasks, three C++ tasks, three C tasks, and two Go tasks.

N.3 Frozen Protocol Package and Runtime Binding

The treatment uses the fixed `programbench_pack_v0` package. It contains six executable binding rules and three advisory rules. Harness adaptation maps their triggers, evidence fields, and actions to the native mini-SWE-agent interface; the package remains fixed during evaluation.

Rule	Mode	Runtime intent
`PB0`	Binding	Require an original candidate implementation rather than copying, linking, or delegating execution to the reference executable.
`P0`	Binding	Protect public probes and other harness-scored surfaces from modification by the candidate.
`P3`	Binding	Invalidate stale verification after source changes; modified code must be rebuilt or rechecked in its current state.
`P4`	Binding	Preserve build/check failure semantics; an unsuccessful or unavailable check cannot be represented as successful evidence.
`P7`	Binding	Require `patch.txt`, when present, to correspond to the current candidate code state.
`P8`	Binding	Gate final submission on originality, non-empty source changes, current build evidence, absence of known failing current checks, and inspection of the current diff.
`P1`	Advisory	Encourage probing the available reference behavior on real inputs before repeated implementation changes.
`P5`	Advisory	Encourage obtaining new reference evidence when the same failure recurs without new information.
`P6`	Advisory	Encourage periodic re-inspection of the current source and diff during long trajectories.

Table 62: **ProgramBench protocol package.** Binding rules can affect execution; advisory rules are recorded as shadow guidance and do not themselves block an action.

N.4 Runtime Footprint

Across the 25 treatment trajectories, the coding agent executes 2,215 Bash actions. The protocol layer records 933 validation receipts and 815 inspections of source, diffs, or reference behavior, while issuing 40 hard refusals. Twenty-six hard refusals arise from `P3`, requiring current verification after the code state changes, and fourteen arise from `P4`, preserving a failed or unavailable build/check as a failure rather than accepting it as successful evidence.

The three advisory rules additionally record 293 shadow cases in which their recommended condition would have requested a different next step: 239 under `P6`, 37 under `P1`, and 17 under `P5`. These shadow events do not block execution.

The 933 validation receipts comprise 370 `PASS`, 48 `FAIL`, and 515 `INCONCLUSIVE` outcomes.

N.5 Scoring and Outcomes

We compute each task's behavioral pass percentage using ProgramBench's native evaluator, excluding tests marked ignored by that evaluator, and macro-average over the fixed 25 tasks. Both systems use the same scoring procedure.

Official mini-SWE-agent reaches 64.164%; the protocol treatment reaches 70.916%, an absolute gain of 6.752 percentage points (10.5% relative). All 25 treatment trajectories produce submissions and completed evaluations.