LOST BETWEEN AGENTS: HOW DO TASK INSTRUCTIONS DRIFT ACROSS LLM AGENTS?

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-agent LLM systems solve complex tasks through communication, and successful collaboration depends on whether important task information is preserved as messages pass between agents. Existing evaluations mainly measure final task performance or overall message similarity, but they do not show what changed during communication or whether those changes actually mattered. This gap raises a key question: *how can we identify information loss in agent communication and determine which changes affect downstream behavior?* Importantly, a change that occurs frequently is not necessarily the one that matters most to the receiving agent. To address this question, we study sequential task specification handoffs and introduce COMMAUDIT, a protocol for auditing how specifications change as they pass between agents. COMMAUDIT measures both how often task constraints change, which we call *exposure*, and how much those changes affect the executor, which we call *impact*, using matched repair and control interventions. Using this protocol, we find that repairing the changes identified by COMMAUDIT consistently improves executor accuracy over stylistic controls across communication agents powered by four different LLMs. Moreover, across three benchmarks and four communication agents, the most frequent change type differs from the one with the largest downstream effect in six of twelve settings, while the same change counts can still lead to different repair outcomes.

1 INTRODUCTION

Multi-agent LLM systems decompose complex tasks across multiple agents, enabling coordinated solutions that may be difficult for a single model to produce alone (Hong et al., 2024; Du et al., 2023; Wu et al., 2023; Xu et al., 2025). But effective coordination requires something more basic: each agent must understand what it is being asked to do. That understanding forms through communication (Qian et al., 2024; Li et al., 2023), yet what a message is meant to convey and what the receiving agent acts on can differ (Lin et al., 2026; Becker et al., 2026; Cemri et al., 2026). An agent that reasons correctly about a misunderstood task still gives the wrong answer. System performance therefore depends not only on individual capability but on whether task-relevant information survives each communication step (Zhang et al., 2025a; Shen et al., 2025; He et al., 2026); information loss can become a system-level bottleneck even when every participating agent is individually capable. How specifications drift across hops, and which of those changes matter, determine what the executor is ultimately asked to solve.

Existing evaluations offer only partial visibility into this bottleneck. End-to-end performance shows whether the system ultimately succeeded but cannot separate communication failures from reasoning or execution failures (Liu et al., 2024b; Cemri et al., 2026). Message-level similarity indicates how much a specification diverges from its source but not whether the divergence involves task-critical content. Change counts record which alterations occur most often; they cannot say whether frequent changes are the ones the executor responds to (Lin et al., 2026; Becker et al., 2026). Together, they leave the key question open: ***how can we identify information loss during agent communication and determine which changes affect downstream behavior?***

We study this question through sequential task-specification handoffs, a foundational form of agent delegation in which one agent must convey what should be done before another can act (Zhang et al., 2024b). Answering it requires separating what changes on the message side from whether those

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.

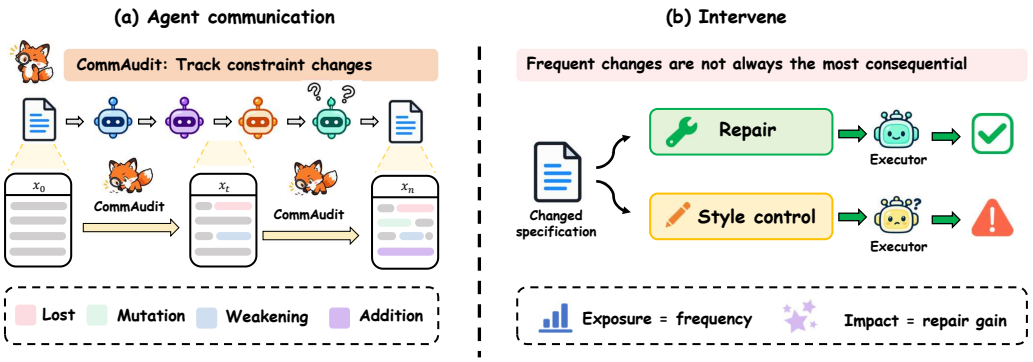

Figure 1: **Overview of CommAudit. (a)** Source-anchored constraint tracking labels each change as loss, mutation, weakening, or addition, yielding message-side exposure. **(b)** Matched repair–control interventions estimate the corresponding receiver-side impact. The most frequent changes are not necessarily the most consequential.

changes matter to the executor. *Exposure* captures how often task-relevant constraints change as a specification passes between agents. *Impact* measures the receiver-side consequence: how executor behavior changes when an identified alteration is repaired rather than left in place. Impact cannot be identified from the transmitted message alone; it must be measured relative to the receiving agent.

To measure both, we introduce CommAudit, an intervention-based protocol for auditing task-relevant information in agent communication (Figure 1). CommAudit decomposes the source specification into atomic constraints (Zhou et al., 2023; Jiang et al., 2024) and tracks each as *faithful*, *lost*, *mutated*, or *weakened* across hops, recording additions separately. Impact is estimated by pairing targeted repairs with matched stylistic controls that make equivalent surface edits without touching any constraint.

Repeated relay degrades executor accuracy. On BigCodeBench, repairing the changes CommAudit identifies yields substantially higher executor accuracy than matched stylistic controls across all four communication agents. Across all three benchmarks and four communication agents, the most exposed change type is not the most consequential in six of twelve settings, and tasks with identical change-count vectors yield heterogeneous repair outcomes. Formal analysis (Blackwell, 1953) shows that the invariance condition required for exposure to proxy impact fails in every setting we evaluate; communication quality is therefore a joint property of the sender, transmitted message, receiving agent, and downstream task.

We make three contributions:

- **Measurement of consequential information loss.** We formulate consequential information loss in agent communication as a measurement problem with two dimensions: *exposure*, which measures how often task-relevant constraints change during communication, and *impact*, which measures how these changes affect executor performance.
- **Auditing communication through interventions.** We introduce CommAudit, an intervention-based audit protocol that combines source-anchored constraint tracking with targeted repairs and matched stylistic controls. On BigCodeBench, these repairs improve executor accuracy over the controls across all four communication agents.
- **Establishing the exposure–impact gap.** Across three benchmarks and four communication agents, the most frequent change type differs from the most consequential in six of twelve settings, while identical change-count vectors yield heterogeneous repair outcomes. We characterize the invariance condition required for change counts to universally predict impact and show that the observed heterogeneity violates this condition.

2 Related Work

Inter-agent communication. LLM multi-agent systems span topologies from debate and voting to sequential pipelines. Sequential chains are widely used for staged refinement and long-context

decomposition (Zhang et al., 2024b); sharing intermediate reasoning can narrow the gap between sender intent and receiver interpretation (Zheng et al., 2026); latent interaction patterns shape information survival across hops (Zou et al., 2025); and agent diversity can raise aggregate accuracy (Yang et al., 2026). Other work improves communication by pruning redundant messages, tracing propagation across sparse and dense topologies, or augmenting text with state-transition information (Zhang et al., 2025a; Shen et al., 2025; Tang et al., 2025); information-theoretic accounts compare single- and multi-agent systems under bounded relays (He et al., 2026; Yu et al., 2026). We hold topology fixed and audit *what* happens to the task specification along a linear chain, tracking each atomic constraint's state across hops.

Iterated LLM generation. Repeated LLM transmission can amplify biases and drive text toward attractor-like states, and iterative rewriting distorts content even when individual outputs stay fluent (Perez et al., 2024; Mohamed et al., 2025); chains where each agent sees only the previous relay admit a Markov description (Geng et al., 2026). Closer to our setting, Shawn (2026) shows that strong communication agents can be nearly lossless while weak ones tend to preserve injected errors rather than correct them. Other work documents handoff failure modes including missing data, referential drift, safety-constraint drift, and error propagation (Lin et al., 2026; Becker et al., 2026; Li et al., 2026b; Singh & Pawar, 2026). These studies show that drift takes many forms but do not measure whether a given change matters to the executor; we pair constraint tracking with matched repair–control interventions to separate *exposure* from *impact*.

Multi-agent failure attribution. A complementary line diagnoses *which step* causes behavioral failure: surveys identify inter-agent miscommunication as a primary contributor (Cemri et al., 2026); attribution methods assign failures to specific agents via trace replay or intervention (Zhang et al., 2025b; Zhang et al.; Chen et al., 2026); and debuggers patch identified failure points at inference time (Ma et al., 2026). These methods ask which agent produced a wrong output; CommAudit tracks what changed in the specification at each step and measures the executor-side consequence through matched repair–control interventions.

Constraint-level instruction fidelity. Prior work represents tasks as testable requirements. IFEval and FollowBench evaluate verifiable constraints at multiple difficulty levels (Zhou et al., 2023; Jiang et al., 2024); CodeIF specializes this to code generation (Yan et al., 2025); InFoBench, CFBench, and DeCRIM decompose instructions or localize constraint failures (Qin et al., 2024; Zhang et al., 2025c; Ferraz et al., 2024); ComplexBench and related work test multi-constraint compositions and broader constraint types (Wen et al., 2024; Pyatkin et al., 2026); AGENTIF extends evaluation to agentic, tool-using trajectories (Qi et al., 2026). CommAudit applies the same decomposition to the communication step: it source-anchors each relay to x_0, labels each constraint's state as *faithful*, *lost*, *mutated*, or *weakened*, records additions separately, and pairs repairs with matched stylistic controls to isolate constraint-level changes from surface edits (which already shift executor behavior (Sclar et al., 2024)). A detailed survey of related work appears in App. D.

3 Problem Formulation

We model LLM-mediated task communication as a communication chain followed by an executor that solves the task from a relayed specification.

3.1 The Communication Chain

Let x_0 denote the original natural-language task specification and x_t its relay at hop t. A *communication agent* M, given a communication prompt π_R and the previous relay, samples

$$x_{t+1} \sim q_M(\cdot \mid x_t; \pi_R), \qquad t = 0, 1, \ldots, H-1. \tag{1}$$

This gives two properties:

(P1) *Markov structure.* $x_{t+1} \perp x_{<t} \mid x_t$.

(P2) *No oracle correction.* The chain contains no feedback path from the executor's outcome back to any communication agent.

At each hop t, a fresh *executor* E takes x_t (concatenated with the benchmark's output-format hint; see App. B.1) and produces an answer $y_t \sim E(\cdot \mid x_t)$. A programmatic evaluator then maps y_t to a correctness bit $\ell_t \in \{0, 1\}$. We write

$$A_t \;:=\; \Pr[\ell_t = 1] \tag{2}$$

for the executor's success probability at hop t: A_0 is the no-relay baseline and A_H the terminal relay accuracy.

3.2 The Relay as a Garbling of Task Evidence

Let J denote the ground-truth task drawn from the benchmark distribution and let X_t be the hop-t relay viewed as a random variable, with realization x_t. Under (P1)–(P2), the joint law satisfies the Markov structure

$$J \;\longrightarrow\; X_0 \;\longrightarrow\; X_1 \;\longrightarrow\; \cdots \;\longrightarrow\; X_H, \tag{3}$$

so each hop is a stochastic post-processing — a *garbling*, in the sense of Blackwell (1953) — of the previous relay with respect to J. The communication agent's generative prior P_M enters the transition kernel, but since P_M carries no task-specific information about J beyond X_t, the Markov structure is preserved.

Proposition 1 (Blackwell monotonicity of oracle value)**.** *For any bounded utility $u\colon \mathcal{J} \times \mathcal{A} \to \mathbb{R}$, define the oracle decision value at hop t as $V_t^\star(u) = \sup_\pi \mathbb{E}[u(J,\, a_t)]$ with $a_t \sim \pi(\cdot \mid X_t)$, the supremum ranging over Markov kernels $\pi\colon \mathcal{X} \to \Delta(\mathcal{A})$. Then under (P1)–(P2), $V_{t+1}^\star(u) \le V_t^\star(u)$ for every u.*

The proposition establishes a decoder-independent information ceiling: $V_t^\star$ may select a new decision rule at each hop, whereas our experiments hold the executor E fixed, so the executor-relative accuracy gap below that ceiling must be measured directly. Generic shared information is also insufficient: $I(X_0; X_t)/H(X_0)$ can approach 0 or 1 regardless of whether decisive task content survives (App. G.2).

3.3 Constraint State as a Measurement of the Relay

Relays may alter atomic constraints as well as surface properties such as wording, ordering, format, and length. CommAudit resolves the former into a constraint state and handles the latter through the matched controls of §4.

Decompose x_0 into a finite set of *atomic constraints* $\mathcal{C}(x_0) = \{c_1, \ldots, c_K\}$, each an assertion the executor's solution must respect (e.g., a function signature, a boundary condition, a return format, an example output). After t hops of relay, each constraint c_i has state

$$s_{i,t} \;\in\; \mathcal{S} \;:=\; \{F,\, L,\, M,\, W\}, \tag{4}$$

where F = *faithful* (present and unchanged), L = *lost* (absent from x_t), M = *mutated* (present but semantically different), and W = *weakened* (present but its strength or scope is reduced). Constraints added to x_t that were not present in x_0 are recorded separately as an *addition set* B_t. The tuple $\Phi(x_0, x_t) := (s_{1,t}, \ldots, s_{K,t}; B_t)$ is the *constraint state* of x_t.

The constraint state does not encode wording, ordering, format, or length, so $\Phi(x_0, x_t)$ is a *partial* measurement of x_t. $\mathcal{C}(x_0)$ also treats constraints as an unordered flat set; this approximation can break under conditional or chained dependencies between constraints (App. H).

4 CommAudit: Auditing Task-Relevant Information in Agent Communication

We develop CommAudit, an audit protocol that separates *what has changed* from *what matters to the executor*. CommAudit has three components: an exposure profile that summarizes which change types the communication agent produces; matched interventions that quantify executor-relative impact on affected tasks and across the full task population; and a boundary result characterizing when exposure counts can represent downstream executor consequences. Figure 1 gives an overview.

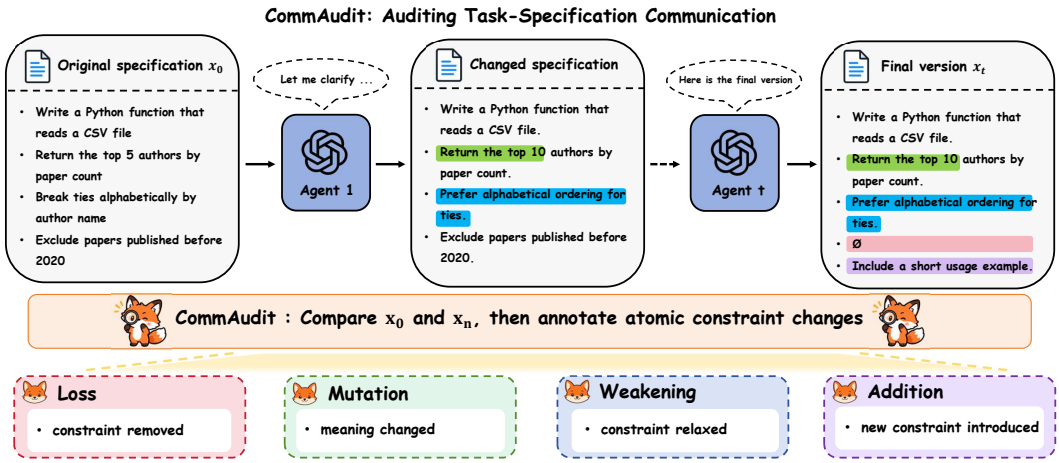

Figure 2: **Source-anchored constraint tracking in CommAudit.** Each relay x_t is compared against the original x_0; every atomic constraint is labeled as faithful, lost, mutated, or weakened, and unsupported additions are recorded separately. Anchoring to x_0 at each hop captures cumulative changes, not just differences between adjacent messages.

Throughout, $j \in \mathcal{T}$ indexes a task, m indexes a *communication agent* (the communication agent in the experiments; more generally, a full configuration of communication agent, prompt, format, and decoding), and $t \in \{0, 1, \ldots, H\}$ indexes the hop.

4.1 Exposure Profile

Given the constraint state $\Phi(x_{j,0}, x_{j,t}) = (s_{j,1,t}, \ldots, s_{j,K_j,t}; B_{j,t})$ from §3.3, write $N_{j,t,k} = \sum_i \mathbf{1}(s_{j,i,t} = k)$ for the number of type-k constraint changes in task j at hop t, and $N_{j,t,+} = |B_{j,t}|$ for the number of additions. The communication agent's *exposure profile* at hop t is the vector of mean change counts by type,

$$d_{m,t,k} \;=\; \mathbb{E}_{j\sim\mathcal{T}}\big[N_{j,t,k}\big], \qquad k \in \{L, M, W, +\}, \tag{5}$$

so $d_{m,t,k}$ reports the expected number of type-k constraint changes that communication agent m produces per task at hop t. In figures and tables of §5 where m and t are fixed by context, we write $\mathbb{E}[n_k]$ for the same quantity. $\mathbf{d}_{m,t}$ measures the *current* state at hop t rather than a cumulative history: a constraint mutated at hop 2 and returned to a faithful state at hop 4 contributes to $d_{m,2,M}$ but not to $d_{m,4,M}$. A per-communication agent change signature (e.g. addition-dominated, loss-dominated, mutation-dominated) is a property of $\mathbf{d}_{m,t}$.

4.2 Conditional and Population Impact

The exposure profile captures *what has happened* but not *what matters to the executor*. We address the latter with type-specific matched interventions on $x_{j,t}$.

For each change type $k \in \{L, M, W, +\}$, let $D_{j,t,k} := \mathbf{1}\{N_{j,t,k} > 0\}$ indicate that task j carries at least one type-k change at hop t. On tasks with $D_{j,t,k} = 1$, let $R_k(x_{j,t})$ be a targeted repair that reverts the annotated type-k changes back to x_0 (for $k \in \{L, M, W\}$) or removes unsupported additions (for $k = +$), and let $C_k(x_{j,t})$ be a matched control that applies the prescribed stylistic edits without touching any constraint. The task-level matched contrast is

$$\Delta_{E,k}(j,t) \;=\; Y_j\big(R_k(x_{j,t})\big) - Y_j\big(C_k(x_{j,t})\big), \tag{6}$$

where $Y_j(x) \in \{0, 1\}$ is the executor's correctness bit on task j when run on input x.

The *conditional impact* of type k is the signed matched repair–control effect on the affected subset,

$$\tau_{m,t,k} \;=\; \mathbb{E}_j[\Delta_{E,k}(j,t) \,|\, D_{j,t,k} = 1]\,. \tag{7}$$

It measures the expected change in executor accuracy from the type-k repair among tasks where that repair applies. In appendix tables where m and t are fixed by context, we suppress those indices and write τ_k.

For comparison with the unconditional exposure profile, we use the *population impact*

$$\pi_{m,t,k} \;=\; \Pr_j(D_{j,t,k}=1)\,\tau_{m,t,k} \;=\; \mathbb{E}_j[D_{j,t,k}\,\Delta_{E,k}(j,t)]\,, \tag{8}$$

where the product in the final expectation is defined as zero when $D_{j,t,k}=0$. The vector $\boldsymbol{\pi}_{m,t}$ is the *population-impact profile*: its type-k entry is the signed accuracy change induced by the specified type-k repair per task drawn from the full benchmark population. This puts $\mathbf{d}_{m,t}$ and $\boldsymbol{\pi}_{m,t}$ on the same task population, though not on the same numerical scale.

Because change types co-occur, a type-k-only repair can leave other failure-causing changes in place. Neither $\tau_{m,t,k}$ nor $\pi_{m,t,k}$ is therefore an isolated causal attribution, and the entries of $\boldsymbol{\pi}_{m,t}$ need not sum to the endpoint loss or to the all-repair effect. At the aggregate level, the all-repair matched effect is

$$\tau^{\mathrm{all}}_{m,t} \;=\; \mathbb{E}_j\left[Y_j\big(R_{\mathrm{all}}(x_{j,t})\big) - Y_j\big(C_{\mathrm{all}}(x_{j,t})\big)\right]. \tag{9}$$

4.3 When Can Exposure Represent Population Impact?

The exposure profile $\mathbf{d}_m$ and population-impact profile $\boldsymbol{\pi}_m$ are two measurements of the same relay over the same task population. The question is whether exposure counts alone could suffice: does some fixed function of the count summary track the executor-relative consequence across benchmarks and executors? We give the boundary condition under which such a reduction is possible.

Fix a communication agent m, a hop t, and an executor E. A single *task–relay state* is

$$\omega \;=\; (j,\, x_{j,0},\, x_{j,t}) \;\in\; \Omega, \tag{10}$$

recording the task, its original specification, and the hop-t relay; a distribution Q on Ω describes a population of such states. An *audit summary* is any mapping $S\colon \Omega \to \mathcal{Z}$ that coarsens ω to a statistic in some space $\mathcal{Z}$; the constraint state $\Phi(\omega)$ of §3.3 and count vector $N(\omega) = \big(N_L, N_M, N_W, N_+\big)(\omega)$ are both instances. Writing $\Delta_{E,k}(\omega)$ for the task-level contrast associated with state ω, the population-level executor consequence for type k is

$$r_{E,k}(\omega) \;=\; D_k(\omega)\,\Delta_{E,k}(\omega), \tag{11}$$

with the same zero convention as in Equation (8), so $\pi_{m,t,k} = \mathbb{E}_Q[r_{E,k}]$ for the empirical population induced by (m,t) on a benchmark. We ask whether a fixed summary S suffices: does a single distribution-independent function f predict $\mathbb{E}_Q[r_E]$ from S across all Q?

Proposition 2 (Outcome sufficiency under coarsening)**.** *Let $r_E\colon \Omega \to \mathbb{R}$ be bounded and measurable, and let $S\colon \Omega \to \mathcal{Z}$ be measurable, with $(\Omega, \mathcal{F})$ and $(\mathcal{Z}, \mathcal{G})$ both standard Borel spaces. The following are equivalent:*

(i) there exists a measurable function $f\colon \mathcal{Z} \to \mathbb{R}$ such that

$$\mathbb{E}_Q\big[r_E(\omega)\big] \;=\; \mathbb{E}_Q\big[f(S(\omega))\big]$$

for every probability distribution Q on Ω;

(ii) r_E is constant on every fiber of S: for all $\omega, \omega' \in \Omega$,

$$S(\omega) = S(\omega') \quad\Longrightarrow\quad r_E(\omega) = r_E(\omega').$$

Instantiating Proposition 2 with $S = N$ and $r_E = r_{E,k}$, the fiber condition requires that any two task–relay states with identical count vectors produce identical executor-relative consequences. This must hold regardless of which specific constraint changed, on which task, under which communication agent, or under which benchmark evaluator. Because D_k is determined by N_k, this reduces on the affected subset to requiring $\Delta_{E,k}$ to be constant within each count fiber. The empirical analysis in §5.4 shows that this condition fails on the environments we evaluate; exposure and population impact must therefore be reported separately (proof in App. G.3).

5 Experiments

We report three results: repeated relay degrades executor accuracy; CommAudit's annotated changes are consequential for the executor and its constraint profile carries information beyond text features; and exposure does not reliably localize population impact.

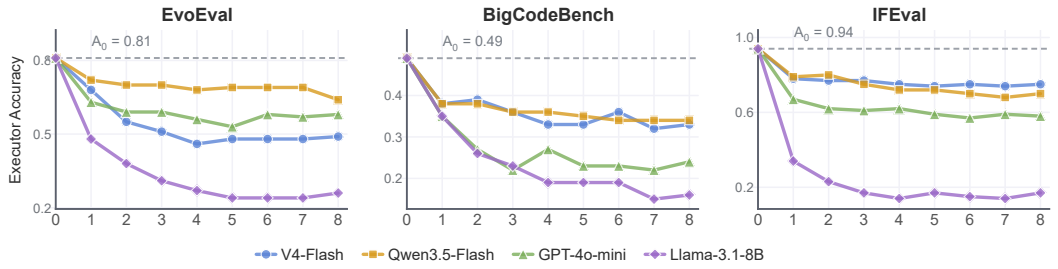

Figure 3: Executor accuracy per hop under the **paraphrase** prompt, across three benchmarks (columns) and four communication agents (curves). Most accuracy loss concentrates in the first two or three hops.

5.1 Setup

Communication chain and executor. We run an $H = 8$ hop chain and evaluate a fresh executor at every hop $t \in \{0, 1, \ldots, 8\}$. The executor is held fixed to DeepSeek V4-Flash; the communication agent at each hop sees only x_{t-1}. All decoding and budget settings are held constant across calls (App. B.1). Impact estimates in §5.3–§5.4 are executor-relative in the sense of Proposition 2; the exposure profile is executor-independent.

Benchmarks. Three programmatically graded benchmarks, each with $N = 100$ tasks: **EvoEval** (Xia et al., 2024) (five HumanEval paraphrases), **BigCodeBench** (Zhuo et al., 2025) (multi-library Python with unit tests), and **IFEval** (Zhou et al., 2023) (instruction following with programmatic checkers). Together they span code generation and instruction following.

Models. We instantiate four communication agents, each powered by a distinct LLM: DeepSeek V4-Flash, Qwen3.5-Flash, GPT-4o-mini, and Llama-3.1-8B-Instruct. Each drives an independent chain; the chains are evaluated independently under identical conditions. Communication prompts are defined where they first appear; full prompts and annotation protocol are in App. B–C.

5.2 Repeated Relays Degrade Executor Performance

To test whether degradation is confined to specific prompt regimes, we vary π_R across five conditions. **No-extra** passes the task without a communication prompt, capturing default completion behavior. **Verbatim** reproduces the input word-for-word and serves as a fidelity reference. **Summarize** and **elaborate** probe lossy-compression and unsupported-addition regimes respectively. **Paraphrase** asks the agent to rephrase freely without fidelity or length constraints, representing the default inter-agent communication mode in deployed multi-agent systems. Appendix Table 3 shows that degradation is not confined to any single regime: every non-**verbatim** cell has $\Delta < 0$, $|\Delta|$ is at least 0.10 on almost every row (maximum 0.78), and degradation persists in several **verbatim** conditions as well, though degradation is generally larger under the other four prompts across both code generation and instruction following.

Figure 3 shows the hop-by-hop trajectory under **paraphrase**, the running condition for the rest of §5. Across the twelve (communication agent, benchmark) curves, most of the loss concentrates in the first two or three hops rather than accumulating gradually, suggesting that early relays are the critical bottleneck for information preservation. The four communication agents also reach markedly different accuracy plateaus despite sharing the same executor and benchmarks, showing that agent identity is a key determinant of how much task-relevant content survives. The remainder of §5 examines which specification changes underlie these differences and which of them matter for the executor.

5.3 Does CommAudit Measure Consequential Changes?

Before using CommAudit's annotations to compare communication agents, we verify two properties: that the annotated changes are consequential for the executor, and that the constraint profile carries information beyond relay text features. We test both on hop-8 BigCodeBench relays using a matched-intervention repair experiment and a known-group classification; protocols are in App. E.

Reverting the annotated changes moves the executor. Figure 4 compares executor performance on three versions of the same hop-8 relay: Hop 8, a stylistic *Control* that re-edits the same textual locations without modifying any annotated constraint, and a *Repair* that restores every annotated constraint change to its source specification in x_0. Across all four communication agents, the Control condition stays within ± 0.02 of Hop 8, whereas Repair consistently recovers substantial accuracy. V4-Flash improves from 0.33 to 0.45, Qwen3.5-Flash from 0.34 to 0.44, GPT-4o-mini from 0.24 to 0.45, and Llama-3.1-8B from 0.15 to 0.46, with the two weaker communication agents recovering to within ± 0.04 of the no-relay baseline $A_0 = 0.49$. Because Repair and Control are matched in edit location and differ only in whether the annotated constraint changes are reverted, the gap $A_{\text{repair}} - A_{\text{control}}$ is attributable to the annotated changes. App. E.5 shows the gap persists when the executor is replaced by a different-family model.

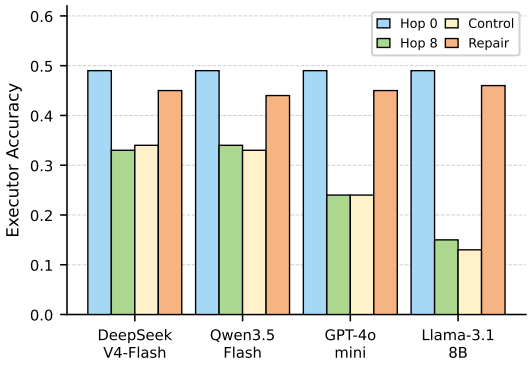

Figure 4: Repair vs. matched stylistic control on hop-8 BigCodeBench relays.

Table 1: Communication agent-identity classification on hop-8 BigCodeBench relays.

Feature set	Macro AUROC
Chance (macro OVR)	0.500
Length ratio $\lvert x_8\rvert/\lvert x_0\rvert$	0.668 ± 0.038
Word count delta	0.677 ± 0.047
Sentence-embedding cosine	0.776 ± 0.022
Constraint counts $N_{j,8}$	0.797 ± 0.041
Cosine + constraint profile	$\mathbf{0.888 \pm 0.017}$

The constraint profile adds information beyond coarse text features. Table 1 asks each candidate feature to identify the communication agent from a single hop-8 relay $x_{j,8}$, evaluated under 5-fold GroupKFold by task ID (no task appears in both training and evaluation). Among the listed text features, sentence-embedding cosine to x_0 is the strongest (0.776 ± 0.022), while the length-based features perform substantially worse (0.668 and 0.677). The four-dimensional per-task count vector $N_{j,8} = (N_{j,8,L}, N_{j,8,M}, N_{j,8,W}, N_{j,8,+})$ alone reaches 0.797 ± 0.041, slightly above the strongest listed text feature. Combining the count vector with cosine raises AUROC from 0.776 ± 0.022 to 0.888 ± 0.017, indicating that the constraint profile adds information beyond embedding cosine.

5.4 Exposure Does Not Reliably Localize Downstream Impact

We compare the exposure profile $\mathbf{d}_m$ (§4.1) and the population-impact profile $\boldsymbol{\pi}_m$ (§4.2) at hop 8, across all four communication agents on BigCodeBench, EvoEval, and IFEval. The question is whether exposure reliably identifies which change types matter most for executor outcomes, not merely whether the two profiles share a numerical scale.

Exposure does not admit a stable calibration to population impact. Figure 5 shows neither a uniformly positive nor a uniformly negative relationship. Loss exposure and population impact rise together across the BigCodeBench communication agents, but the ordering changes for Addition and Mutation and across benchmarks. In particular, high Addition exposure can accompany positive, negligible, or negative population impact.

The most exposed type need not have the largest population impact. Table 2 reports the complete three-type profiles. The exposure and population-impact maxima differ in six of the twelve settings. Four of these disagreements have bootstrap probability $p_{\text{mis}} \geq 0.89$; the other two are more moderate (0.62 and 0.69), so we treat the result as a profile-level pattern rather than twelve independent rank claims. For Qwen3.5-Flash on BigCodeBench, Addition is the largest exposure component (2.02) yet carries negative population impact (-0.02), while the less prevalent Loss component (1.18) yields $+0.02$. On IFEval, Qwen3.5-Flash again produces substantially more Additions than Losses (2.18 versus 0.62), yet Loss repair has more than four times the population impact ($+0.09$ versus $+0.02$).

Table 2: Exposure and population-impact profiles at hop 8. Bold marks the largest change type in each profile; p_{mis} is the bootstrap probability (B=5,000) that the two maxima differ.

		Exposure $\mathbb{E}[n_k]$			Population Impact π_k			
Benchmark	Model	Loss	Add.	Mut.	Loss	Add.	Mut.	p_{mis}
BigCodeBench	V4-Flash	0.47	**3.16**	1.07	**+0.02**	+0.01	−0.04	0.69
	Qwen3.5-Flash	1.18	**2.02**	1.77	**+0.02**	−0.02	+0.02	0.92
	GPT-4o-mini	**1.75**	0.45	0.77	**+0.07**	+0.03	+0.03	0.28
	Llama-3.1-8B	**3.56**	1.00	0.31	**+0.25**	+0.02	+0.04	0.00
EvoEval	V4-Flash	0.47	**1.44**	0.75	+0.01	**+0.08**	+0.06	0.31
	Qwen3.5-Flash	0.88	**1.23**	1.06	−0.05	+0.00	**+0.03**	0.62
	GPT-4o-mini	**1.99**	0.12	0.34	**+0.04**	−0.01	+0.04	0.46
	Llama-3.1-8B	**2.58**	0.10	0.50	**+0.28**	+0.00	+0.12	0.01
IFEval	V4-Flash	0.26	**1.39**	0.53	**+0.04**	+0.01	+0.01	0.89
	Qwen3.5-Flash	0.62	**2.18**	0.83	**+0.09**	+0.02	−0.01	0.98
	GPT-4o-mini	0.54	0.49	**0.70**	**+0.16**	+0.02	+0.09	0.90
	Llama-3.1-8B	**2.20**	1.11	0.76	**+0.62**	+0.02	+0.20	0.00

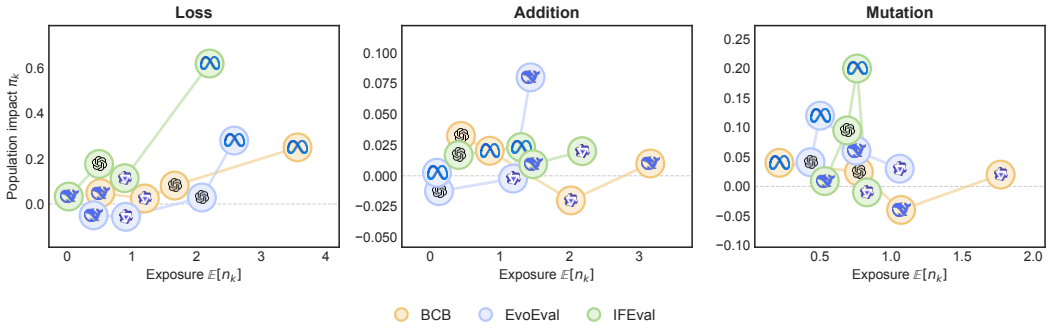

Figure 5: Exposure $\mathbb{E}[n_k]$ vs. population impact π_k at hop 8 for three change types (Loss, Addition, Mutation). Each point is one (communication agent, benchmark) chain; logos mark communication agents, color marks benchmarks. Weakening is omitted (all cells $n_k < 20$); bootstrap CIs for all four types appear in Table 4.

Aggregate agreement does not make counts outcome-sufficient. In the remaining six combinations the maxima agree, but this does not mean exposure counts are sufficient to predict which change types matter downstream. Proposition 2 requires any exposure-only surrogate to assign the same executor-relative consequence to task–relay states with identical count vectors. Direct within-fiber analysis contradicts this requirement: in 94% of non-sparse (benchmark, communication agent, change type) cells, within-fiber variance in the matched contrast $\Delta_{E,k}$ is at least half of its total variance (App. G.4). Exposure therefore identifies where a communication agent changes specifications, but does not reliably identify which change types matter most for executor outcomes. Finally, π_k is the population-level effect of the specified type-k repair, not an additive attribution of the endpoint loss: change types co-occur, and their repairs can interact.

6 Conclusion

We introduced CommAudit, an audit protocol that separates exposure (how often a communication agent changes task constraints) from executor-relative impact under matched repair–control interventions. Across three benchmarks and four communication agents, repeated relay produces a consistent early-drop pattern in executor accuracy. On BigCodeBench, reverting CommAudit's annotated changes recovers substantial accuracy, and the constraint profile adds information beyond embedding cosine. Exposure does not, however, stably locate population impact: the two profile maxima differ in six of twelve settings, and identical count vectors retain heterogeneous matched outcomes. Formal analysis shows that using exposure as a surrogate for impact requires an invariance condition that fails in every environment we evaluate; the two must be reported separately. We discuss limitations in App. H and design implications for multi-agent communication in App. I.

REFERENCES

Pratyay Banerjee and Ankit Chadha. Routed graph handoff: Adaptive format selection for multi-agent llm delegation. *arXiv preprint arXiv:2608.25277*, 2026.

Jonas Becker, Lars Benedikt Kaesberg, Andreas Stephan, Jan Philip Wahle, Terry Ruas, and Bela Gipp. Stay focused: Problem drift in multi-agent debate. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 5068–5102, 2026.

David Blackwell. Equivalent comparisons of experiments. *The annals of mathematical statistics*, pp. 265–272, 1953.

David Blackwell. On a class of probability spaces. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 2: Contributions to Probability Theory*, volume 3, pp. 1–7. University of California Press, 1956.

Mert Cemri, Melissa Z Pan, Shuyi Yang, Lakshya A Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, et al. Why do multi-agent llm systems fail? *Advances in Neural Information Processing Systems*, 38, 2026.

Mengzhuo Chen, Junjie Wang, Fangwen Mu, Yawen Wang, Zhe Liu, Huanxiang Feng, and Qing Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 19888–19905, 2026.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pp. 20094–20136, 2024a.

Weize Chen, Chenfei Yuan, Jiarui Yuan, Yusheng Su, Chen Qian, Cheng Yang, Ruobing Xie, Zhiyuan Liu, and Maosong Sun. Beyond natural language: Llms leveraging alternative formats for enhanced reasoning and communication. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10626–10641, 2024b.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.

Zhuoyun Du, Runze Wang, Huiyu Bai, Zouying Cao, Xiaoyong Zhu, Yu Cheng, Bo Zheng, Wei Chen, and Haochao Ying. Enabling agents to communicate entirely in latent space. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27106–27129, 2026.

Thomas Palmeira Ferraz, Kartik Mehta, Yu-Hsiang Lin, Haw-Shiuan Chang, Shereen Oraby, Sijia Liu, Vivek Subramanian, Tagyoung Chung, Mohit Bansal, and Nanyun Peng. Llm self-correction with decrim: Decompose, critique, and refine for enhanced following of instructions with multiple constraints. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7773–7812, 2024.

Mingmeng Geng, Amr Mohamed, Guokan Shang, Michalis Vazirgiannis, and Thierry Poibeau. Markovian generation chains in large language models. *arXiv preprint arXiv:2603.11228*, 2026.

Jiasheng Gu, Hongyu Zhao, Hanzi Xu, Liangyu Nie, Hongyuan Mei, and Wenpeng Yin. Robustness of learning from task instructions. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13935–13948, 2023.

Shashwat Gupta, Anson Bastos, Mayukh Das, Supriyo Ghosh, Nagarajan Natarajan, Chetan Bansal, and Saravan Rajmohan. Learning optimal message representations for agentic communication. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 28849–28879, 2026.

Shizhe He, Avanika Narayan, Ishan Khare, Scott Linderman, Christopher Ré, and Dan Biderman. An information theoretic perspective on agentic system design. In *International Conference on Learning Representations*, volume 2026, pp. 50920–50958, 2026.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Steven Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*, volume 2024, pp. 23247–23275, 2024.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Philippe Laban, Tobias Schnabel, Paul N Bennett, and Marti A Hearst. Summac: Re-visiting nli-based models for inconsistency detection in summarization. *Transactions of the Association for Computational Linguistics*, 10:163–177, 2022.

Philippe Laban, Tobias Schnabel, and Jennifer Neville. Llms corrupt your documents when you delegate. *arXiv preprint arXiv:2604.15597*, 2026.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Jiazheng Li, Emine Yilmaz, Bei Chen, and Thu Le. Towards self-improving error diagnosis in multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 2063–2077, 2026a.

Tianxiao Li, Yixing Ma, Haiquan Wen, Zhenglin Huang, Qianyu Zhou, Zeyu Fu, and Guangliang Cheng. Safe multi-agent behavior must be maintained, not merely asserted: Constraint drift in llm-based multi-agent systems. *arXiv preprint arXiv:2605.10481*, 2026b.

Yunxuan Li, Yibing Du, Jiageng Zhang, Le Hou, Peter Grabowski, Yeqing Li, and Eugene Ie. Improving multi-agent debate with sparse communication topology. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7281–7294, 2024.

Bohan Lin, Kuo Yang, Zelin Tan, Yingchuan Lai, Chen Zhang, Guibin Zhang, Xinlei Yu, Miao Yu, Xu Wang, Yudong Zhang, et al. Agentask: Multi-agent systems need to ask. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28055–28077, 2026.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173, 2024a.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, pp. 52989–53046, 2024b.

Ryan Lowe, Jakob Foerster, Y-Lan Boureau, Joelle Pineau, and Yann Dauphin. On the pitfalls of measuring emergent communication. *arXiv preprint arXiv:1903.05168*, 2019.

Ming Ma, Jue Zhang, Fangkai Yang, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Dover: Intervention-driven auto debugging for llm multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pp. 28059–28087, 2026.

Amr Mohamed, Mingmeng Geng, Michalis Vazirgiannis, and Guokan Shang. Llm as a broken telephone: Iterative generation distorts information. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7493–7509, 2025.

Artidoro Pagnoni, Vidhisha Balachandran, and Yulia Tsvetkov. Understanding factuality in abstractive summarization with frank: A benchmark for factuality metrics. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4812–4829, 2021.

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, 2024.

Jérémy Perez, Grgur Kovač, Corentin Léger, Cédric Colas, Gaia Molinaro, Maxime Derex, Pierre-Yves Oudeyer, and Clément Moulin-Frier. When llms play the telephone game: Cultural attractors as conceptual tools to evaluate llms in multi-turn settings. *arXiv preprint arXiv:2407.04503*, 2024.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hanna Hajishirzi. Generalizing verifiable instruction following. *Advances in Neural Information Processing Systems*, 38, 2026.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Amy Xin, Youfeng Liu, Bin Xu, Lei Hou, and Juanzi Li. Agentif: Benchmarking large language models instruction following ability in agentic scenarios. *Advances in Neural Information Processing Systems*, 38, 2026.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 15174–15186, 2024.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13025–13048, 2024.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. In *International Conference on Learning Representations*, volume 2024, pp. 25055–25083, 2024.

Zayx Shawn. Faithful, not corrective: Message-format effects in multi-hop agent relays are tier-dependent. *arXiv preprint arXiv:2607.09678*, 2026.

Xu Shen, Yixin Liu, Yiwei Dai, Yili Wang, Rui Miao, Yue Tan, Shirui Pan, and Xin Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 12358–12372, 2025.

Prabhjot Singh and Bhushan Pawar. The hallucination snowball: Modeling error propagation as state transitions in multi-agent llm pipelines. *arXiv preprint arXiv:2608.14588*, 2026.

Yichen Tang, Weihang Su, Yujia Zhou, Yiqun Liu, Min Zhang, Shaoping Ma, and Qingyao Ai. Augmenting multi-agent communication with state delta trajectory. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 10230–10251, 2025.

Zhexuan Wang, Yutong Wang, Xuebo Liu, Liang Ding, Miao Zhang, Jie Liu, and Min Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 24013–24035, 2025.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645, 2024.

Robert West, Ashton Anderson, Ece Kamar, and Eric Horvitz. Tandem training for language models. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8265–8278, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. Top leaderboard ranking= top coding proficiency, always? evoeval: Evolving coding benchmarks via llm. *arXiv preprint arXiv:2403.19114*, 2024.

Tianqi Xu, Linyao Chen, Dai-Jie Wu, Yanjun Chen, Zecheng Zhang, Xiang Yao, Zhiqiang Xie, Yongchao Chen, Shilong Liu, Bochen Qian, et al. Crab: Cross-environment agent benchmark for multimodal language model agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 21607–21647, 2025.

Zhuoning Xu, Xiucheng Zhang, Hanjun Luo, Yingbin Jin, Yinpeng Dong, and Hanan Salam. Masdrift: Benchmarking authorization preservation across multi-agent architectures. *arXiv preprint arXiv:2608.07556*, 2026.

Kaiwen Yan, Hongcheng Guo, Xuanqing Shi, Shaosheng Cao, Donglin Di, and Zhoujun Li. Codeif: Benchmarking the instruction-following capabilities of large language models for code generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1272–1286, 2025.

Lu Yan, Xuan Chen, and Xiangyu Zhang. When the specification emerges: Benchmarking faithfulness loss in long-horizon coding agents. *arXiv preprint arXiv:2603.17104*, 2026.

Yingxuan Yang, Chengrui Qu, Muning Wen, Laixi Shi, Ying Wen, Weinan Zhang, Adam Wierman, and Shangding Gu. Understanding agent scaling in llm-based multi-agent systems via diversity. *arXiv preprint arXiv:2602.03794*, 2026.

Wendi Yu, Lianhao Zhou, Xiangjue Dong, Sai Sudarshan Barath, Declan Staunton, Byung-Jun Yoon, Xiaoning Qian, James Caverlee, and Shuiwang Ji. When do multi-agent systems help? an information bottleneck perspective. *arXiv preprint arXiv:2607.16133*, 2026.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. Agentracer: Who is inducing failure in the llm agentic systems?, 2025a. *URL https://arxiv.org/abs/2509.03312*.

Guibin Zhang, Yanwei Yue, Xiangguo Sun, Guancheng Wan, Miao Yu, Junfeng Fang, Kun Wang, Tianlong Chen, and Dawei Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024a.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Yu, and Tianlong Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pp. 75389–75428, 2025a.

Huixiang Zhang and Mahzabeen Emu. Do latent channels actually communicate? a causal audit of latent multi-agent llm. *arXiv preprint arXiv:2607.26773*, 2026.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025b.

Tao Zhang, Chenglin Zhu, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, et al. Cfbench: A comprehensive constraints-following benchmark for llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 32926–32944, 2025c.

Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Ö Arık. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems*, 37:132208–132237, 2024b.

Yuzhe Zhang, Feiran Liu, Yi Shan, Xinyi Huang, Xin Yang, Yueqi Zhu, Xuxin Cheng, Cao Liu, Ke Zeng, Terry Jingchen Zhang, et al. Silo-bench: A scalable environment for evaluating distributed coordination in multi-agent llm systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29379–29398, 2026.

Zihan Zhao, Yuwei Guo, Xiaowei Liu, Qun Ma, Hui Yu, and Xiao Xue. Are agents listening to each other? measuring what drives agent actions in llm-based multi-agent systems. *Expert Systems*, 43 (10):e70392, 2026.

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. Thought communication in multiagent collaboration. *Advances in Neural Information Processing Systems*, 38:123389–123418, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Daisy Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, et al. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8580–8622, 2025.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. Language agents as optimizable graphs. *arXiv preprint arXiv:2402.16823*, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

Jiaru Zou, Ruizhong Qiu, Gaotang Li, Xiyuan Yang, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, et al. Latent collaboration in multi-agent systems. *arXiv preprint arXiv:2511.20639*, 2025.

A AI USE STATEMENT

In this work, we used generative AI tools for improving the readability and language of the manuscript, including grammar, wording, and stylistic polishing. We have not used generative AI tools for developing the research methodology, designing experiments, formulating scientific claims, analyzing or interpreting results, or generating the scientific content of this work. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

B SETUP AND REPRODUCIBILITY DETAILS

B.1 COMMUNICATION CHAIN AND EXECUTOR CONFIGURATION

Every experiment in the paper is driven by the same communication-chain implementation. The four communication agents, {`deepseek-v4-flash`, `qwen3.5-flash`, `gpt-4o-mini`, `llama-3.1-8B-instruct`}, are each invoked through their providers' chat-completion endpoints. The executor is `deepseek-v4-flash` throughout, so differences between chains stem from the specifications the communication agents produce, with the executor held fixed. All model calls use temperature 0, a 2048-token completion budget, and a 90-second per-call timeout. Models that expose an internal reasoning mode (`deepseek-v4-flash` and `qwen3.5-flash`) have it disabled to remove a stochastic confounder.

Execution runs inside a subprocess sandbox with a per-task wall-clock timeout: BigCodeBench uses the official `v0.1.4` test runner; EvoEval uses the same subprocess-based unit-test evaluation against the hidden tests of each variant; IFEval uses the official strict evaluator with no modifications.

The executor is invoked with a benchmark-appropriate system prompt (`"You are an expert programmer."` on the two code benchmarks, and `"You are a helpful assistant that carefully follows instructions."` on IFEval) and the user template `"{task_desc}\n\n{format_hint}"`, where `task_desc` is the current relay x_t and `format_hint` is the benchmark's native output-format directive. As an example, the BigCodeBench format hint is:

```
Your output must be a COMPLETE Python function starting
with the necessary import statements and the `def` line,
including the full function body and a `return` statement.
Do not add prose, explanation, or markdown fences.
```

The EvoEval hint is the HumanEval function-completion hint; IFEval passes the task text through with no additional hint.

B.2 Benchmark Subsampling

Each benchmark is sampled down to $N = 100$ tasks with a fixed seed (`seed=42`) so that all four chains solve the same task list. On **EvoEval** we draw 20 tasks per variant across the five variants (*difficult, creative, combine, subtle, tool_use*), yielding a balanced 100-task subset. On **BigCodeBench** we draw a random subsample from the `v0.1.4` split. On **IFEval** we sample 100 prompts from the release. The executor's base accuracies on these subsets are $A_0 = 0.81, 0.49, 0.94$ for EvoEval, BigCodeBench, and IFEval respectively.

B.3 Communication Agent Prompts

Every communication agent call uses the system prompt `"You are an expert technical writer."` The five communication prompts π_R differ only in the user message, into which the incoming relay x_{t-1} is substituted at the `{task_desc}` placeholder. **paraphrase** is the running condition for §5.3–§5.4; the four short conditions appear only in the full endpoint results of Appendix Table 3.

no-extra.

```
Task description:
{task_desc}
```

verbatim.

```
Reproduce the following task description verbatim.  Do not
paraphrase, summarize, or add anything.
{task_desc}
```

summarize.

```
Summarize the following task description concisely.
Do NOT solve the task.  Do NOT write solution steps.  Only
produce a task description in your own words.
Task description:
{task_desc}
```

elaborate.

```
Elaborate on the following task description.  Add
clarifications and details.
Do NOT solve the task.  Do NOT write solution steps.  Only
produce a task description in your own words.
Task description:
{task_desc}
```

paraphrase.

```
You will receive a task description.  Read it carefully,
then write your own version of the task description for
another person to solve.  Your rewrite is the ONLY thing
that person will see.
Do NOT solve the task.  Do NOT write solution steps.  Only
```

Table 3: Executor accuracy before and after the relay. For each benchmark we report **Base** = executor accuracy on the original specification x_0 (no relay), **Relay** = executor accuracy on the terminal relay x_8 after eight hops of relay, and $\Delta = \text{Relay} - \text{Base}$. Rows are grouped by communication agent and vary the communication prompt; the executor is held fixed to DeepSeek V4-Flash.

		EvoEval			BigCodeBench			IFEval		
Model	Setting	Base	Relay	Δ	Base	Relay	Δ	Base	Relay	Δ
DeepSeek V4-Flash	no-extra	0.81	0.14	-0.67	0.49	0.10	-0.39	0.94	0.44	-0.50
	verbatim	0.81	0.82	$+0.01$	0.49	0.47	-0.02	0.94	0.70	-0.24
	summarize	0.81	0.63	-0.18	0.49	0.16	-0.33	0.94	0.43	-0.51
	elaborate	0.81	0.68	-0.13	0.49	0.38	-0.11	0.94	0.34	-0.60
	paraphrase	0.81	0.49	-0.32	0.49	0.33	-0.16	0.94	0.78	-0.16
Qwen3.5-Flash	no-extra	0.81	0.03	-0.78	0.49	0.00	-0.49	0.94	0.34	-0.60
	verbatim	0.81	0.72	-0.09	0.49	0.46	-0.03	0.94	0.73	-0.21
	summarize	0.81	0.69	-0.12	0.49	0.38	-0.11	0.94	0.41	-0.53
	elaborate	0.81	0.58	-0.23	0.49	0.36	-0.13	0.94	0.22	-0.72
	paraphrase	0.81	0.64	-0.17	0.49	0.34	-0.15	0.94	0.70	-0.24
GPT-4o-mini	no-extra	0.81	0.37	-0.44	0.49	0.14	-0.35	0.94	0.45	-0.49
	verbatim	0.81	0.65	-0.16	0.49	0.48	-0.01	0.94	0.61	-0.33
	summarize	0.81	0.50	-0.31	0.49	0.08	-0.41	0.94	0.34	-0.60
	elaborate	0.81	0.67	-0.14	0.49	0.39	-0.10	0.94	0.32	-0.62
	paraphrase	0.81	0.58	-0.23	0.49	0.24	-0.25	0.94	0.58	-0.36
Llama-3.1-8B	no-extra	0.81	0.20	-0.61	0.49	0.36	-0.13	0.94	0.24	-0.70
	verbatim	0.81	0.38	-0.43	0.49	0.46	-0.03	0.94	0.38	-0.56
	summarize	0.81	0.21	-0.60	0.49	0.11	-0.38	0.94	0.23	-0.71
	elaborate	0.81	0.20	-0.61	0.49	0.20	-0.29	0.94	0.26	-0.68
	paraphrase	0.81	0.26	-0.55	0.49	0.15	-0.34	0.94	0.17	-0.77

```
produce a task description in your own words.
Task description:
{task_desc}
```

B.4 Full Endpoint Results

The complete Base-to-Relay endpoint results for all five communication prompts are reported in Table 3. These results support the outcome-level analysis in §5.2; the main text retains the per-hop trajectory for the running **paraphrase** condition.

C Annotation Protocol

The measurement $\Phi(x_0, x_t)$ is produced by a single LLM annotation pass: for each hop-t relay, the annotator receives the pair (x_0, x_t) and outputs a JSON list of behavioral constraint changes, each tagged with a type in $\{L, M, W, +\}$. The conceptual object being judged is the atomic-constraint decomposition described in §3.3; any constraint not mentioned in the output is treated as faithful (state F). The implementation does not enumerate $\mathcal{C}(x_0)$ as a separate preliminary step. We use `deepseek-v4-flash` as the primary annotator.

C.1 Change Identification and Typing

The annotator prompt begins with the following instruction:

```
You are given two versions of a coding task specification:
- ORIGINAL: the task as originally written - REWRITE: the
task after being paraphrased by an LLM agent
Your job is to identify all behavioral constraint changes
--- differences that would cause a programmer implementing
the function to produce a different implementation or
different output.
```

```
Do NOT report: - Purely stylistic or formatting changes
(section headers, bullet vs prose, markdown) - Paraphrases
that preserve the same behavioral requirement - Changes to
variable names or parameter descriptions that don't alter
semantics
```

For each behavioral change the annotator emits a record with fields `type`, `original_text`, `rewrite_text`, and a one-sentence `impact`. The four types are defined verbatim in the prompt as:

```
loss = a requirement in ORIGINAL is absent in REWRITE
addition = a new requirement in REWRITE not present in
ORIGINAL mutation = an ORIGINAL requirement is replaced
by a different one weakening = an ORIGINAL requirement
is softened (e.g., "must" → "should", or example output
labelled "illustrative / may differ")
```

The loss–mutation boundary is the most frequent annotator disagreement; the two are distinguished by whether the corresponding requirement is still enforceable from x_t alone (mutation, present but changed) or has no textual counterpart at all (loss).

C.2 Inter-Annotator Reliability

To measure the reliability of the typing step, we re-classify all 1,293 behavioral changes recorded across the three BigCodeBench chains (V4-Flash, Qwen3.5-Flash, GPT-4o-mini) with an independent judge, `claude-haiku-4-5-20251001`, holding the primary annotator's change list fixed. Overall four-class agreement is 89.9%, giving Cohen's $\kappa = 0.849$. Per-type one-vs-rest agreement is 0.949 (loss), 0.951 (addition), 0.927 (mutation), and 0.971 (weakening); the corresponding per-type κ are 0.870, 0.901, 0.818, and 0.465. The low weakening κ reflects its very small base rate ($n = 29$ of 1,293) rather than genuine label disagreement.

The reliability estimate above quantifies *change-type assignment* only; it does not certify the reliability of change detection, since the second annotator worked from the primary annotator's detected change list rather than from (x_0, x_t) directly. All discriminant and interventional analyses should therefore be read as conditional on the primary annotator's detection.

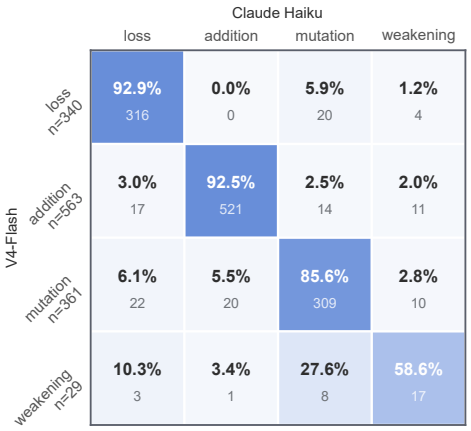

Figure 6: Annotator confusion matrix (primary annotator rows, independent judge columns). Overall agreement 89.9%, $\kappa = 0.849$.

D Related Work

Multi-agent frameworks and architectures. Role-based frameworks assign specialized functions to agents working on shared artifacts: MetaGPT uses meta-programming with structured memory (Hong et al., 2024); ChatDev organizes a waterfall-style software development pipeline (Qian et al., 2024); AgentVerse supports dynamic team composition with adaptive roles (Chen et al., 2024a). Cooperative two-agent setups pair an instructor with an executor directly (Li et al., 2023), while general protocols like AutoGen enable heterogeneous orchestration via conversation (Wu et al., 2023); Language Agents as Optimizable Graphs treats the interaction structure as a differentiable program (Zhuge et al., 2024). Du et al. (2026) enable agents to communicate in continuous latent space, bypassing natural-language generation entirely; Chen et al. (2024b) show that structured and symbolic message formats outperform natural language on certain reasoning tasks. These frameworks treat inter-agent message content as a design parameter; CommAudit tests what natural-language relays actually preserve.

Communication topology, propagation, and message design. A complementary line optimizes who communicates with whom and how. G-Designer learns topologies via graph neural networks (Zhang et al., 2024a); sparse topology reduces debate redundancy (Li et al., 2024); AgentDropout eliminates low-contributing agents at inference time (Wang et al., 2025); information-theoretic treatments bound what coordination gains over single-agent baselines under finite relay budgets (He et al., 2026; Yu et al., 2026). At the message level, Gupta et al. (2026) learn what content to include in messages to maximize downstream performance; Banerjee & Chadha (2026) adaptively select message format based on routing context. Given a fixed topology, Chain-of-Agents relays long documents hop by hop (Zhang et al., 2024b); sharing intermediate reasoning narrows the sender–receiver interpretation gap (Zheng et al., 2026); latent interaction patterns shape information survival (Zou et al., 2025); agent diversity raises accuracy without restructuring communication (Yang et al., 2026); and Liu et al. (2024a) show that language models systematically underuse information in the middle of long contexts, a phenomenon that may amplify constraint loss when task specifications are long. Redundant message pruning (Zhang et al., 2025a), topology-conditioned propagation analysis (Shen et al., 2025), state-delta augmentation (Tang et al., 2025), and faithful prompt compression (Pan et al., 2024) each address transmission efficiency. Lowe et al. (2019) caution that aggregate metrics can mislead: agents may develop task-specific codes that score well on similarity measures while collapsing the representational structure needed for execution. These approaches ask how to make information flow better; CommAudit audits how much task-relevant content survived a fixed natural-language relay.

Iterated generation, specification drift, and inter-agent information gaps. Repeated relay induces systematic distortion. Telephone-game experiments show that transmission amplifies stereotypical forms even when no single agent makes an obvious error (Perez et al., 2024); iterative rewriting accumulates distortion even when individual outputs stay locally fluent (Mohamed et al., 2025); Markov generation chains admit information-theoretic limits on content recovery (Geng et al., 2026); and tandem language model training shows that sequential coupling creates systematic inter-step dependencies (West et al., 2026). Faithfulness evaluation in text summarization (FRANK (Pagnoni et al., 2021); SummaC (Laban et al., 2022)) developed metrics for detecting factual corruptions in generated text; CommAudit applies an analogous constraint-anchored measurement to task-specification relay. Laban et al. (2026) document that LLMs systematically add, remove, and modify facts during document delegation, quantifying corruption with factual-consistency evaluation; Yan et al. (2026) benchmark specification faithfulness loss in long-horizon coding agents, finding constraint coverage degrades with horizon length; MasDrift benchmarks whether authorization constraints survive relay across architectures (Xu et al., 2026); Shawn (2026) tests receiver-side consequences directly, finding that strong agents relay nearly losslessly while weak ones amplify rather than correct errors. At the handoff level, AgentAsk taxonomizes failures into data gaps, signal corruption, and referential drift, finding that agents rarely seek clarification even when task-critical content is absent (Lin et al., 2026); related studies document problem drift in debate (Becker et al., 2026), safety-constraint loss (Li et al., 2026b), and hallucination propagation as state transitions (Singh & Pawar, 2026). Zhao et al. (2026) address the receiver side: measuring what drives agent actions, they find that a substantial share of decisions cannot be attributed to inter-agent messages. CommAudit's matched repair–control interventions test whether a detected message-side gap changes executor output.

Multi-agent failure attribution and evaluation. Diagnostic work asks which agent caused a behavioral failure. Large-scale empirical surveys identify inter-agent miscommunication as a dominant contributor (Cemri et al., 2026); automated attribution methods assign failures to specific agents via trace replay or causal intervention (Zhang et al., 2025b; Zhang et al.); Seeing the Whole Elephant benchmarks attribution quality against known root causes (Chen et al., 2026). Zhang & Emu (2026) causally audit whether latent communication channels transmit task-relevant information, finding many claimed pathways to be inactive; Li et al. (2026a) propose a self-improving error diagnosis loop that iterates over traces to localize and correct root causes; Dover patches identified failure loci at inference time (Ma et al., 2026). System-level evaluation benchmarks include MultiAgentBench (Zhu et al., 2025), AgentBench (Liu et al., 2024b), CRAB for cross-environment multimodal agents (Xu et al., 2025), and Silo-Bench for distributed coordination (Zhang et al., 2026). Attribution methods identify which agent failed; CommAudit identifies which constraint changed and quantifies executor-side consequence.

Instruction following and constraint-level evaluation. CommAudit's per-hop constraint tracking builds on prior work that represents tasks as collections of atomic verifiable requirements. IFEval (Zhou et al., 2023) and FollowBench (Jiang et al., 2024) provide leveled constraint benchmarks; CodeIF extends the paradigm to code generation (Yan et al., 2025). InFoBench, CFBench, and DeCRIM decompose instructions into sub-questions, category-diverse constraints, or self-correcting loops (Qin et al., 2024; Zhang et al., 2025c; Ferraz et al., 2024). ComplexBench (Wen et al., 2024) and Pyatkin et al. (2026) cover multi-constraint compositions and broader constraint types; AGENTIF extends evaluation to agentic, tool-using trajectories (Qi et al., 2026). Sclar et al. (2024) show that surface formatting changes shift executor behavior without altering semantics; Gu et al. (2023) show that task instruction phrasing affects model behavior even when semantic content is preserved. Together these motivate CommAudit's matched stylistic controls, which isolate constraint-level changes from surface-text variation.

E Repair and Matched Control Protocol

The repair and matched-control interventions used in §5.3 and §5.4 both operate on a hop-t relay $x_{j,t}$ conditioned on its annotation. Both are produced by `deepseek-v4-flash` under a fixed system prompt (`"You are an expert at editing technical specifications."`).

E.1 Repair Operator

The aggregate repair operator $R_{\text{all}}(x_{j,t})$ is asked to revert every annotated change back to its state in $x_{j,0}$ while leaving all other content untouched. The rule block inside the repair prompt is:

```
Rules:
- Only revert the specific changes listed.  Do NOT restore
content from ORIGINAL that is not in the change list.
- Preserve all formatting, style, and phrasing from REWRITE
exactly, except where a listed change must be reverted.
- For type=loss:  restore the missing constraint (from
original_text) into the REWRITE.
- For type=addition:  remove the added content
(rewrite_text) from the REWRITE.
- For type=mutation:  replace rewrite_text with
original_text.
- For type=weakening:  restore the mandatory/binding
form of the constraint from original_text (remove any
"illustrative", "may differ", "should" softening).
```

The type-specific repair $R_k(x_{j,t})$ used to estimate conditional impact τ_k and population impact π_k is produced by the same prompt with the change list filtered to type k.

E.2 Matched Control Operator

The matched control operator $C(x_{j,t})$ receives the relay together with a budget `n_changes` that matches the total number of annotated changes, and is asked to apply that many purely stylistic edits without touching any constraint. The prompt's rules are:

```
ALLOWED changes:  - Rephrase section headers ("Parameters:"
→ "Arguments:", "Returns:" → "Return Value:") - Change
markdown formatting style (bullets ↔ numbered list, bold ↔
plain) - Substitute synonyms in non-constraint descriptive
text - Reorder sentences within a descriptive section
(if meaning is unchanged) - Add or remove blank lines for
readability
NOT ALLOWED: - Adding, removing, or modifying any behavioral
constraint or requirement - Changing any example input,
expected output, or numerical constant - Changing function
names, parameter names, or return types - Weakening or
strengthening any "must/should/will" requirement - Adding
or removing any conditional or logical constraint
```

The type-specific control C_k used alongside R_k is generated by the same prompt with `n_changes` set to the count of type-k changes for the affected task.

E.3 Edit-Budget Matching and Fallback

The two operators are edit-budget-matched at the change-count level: the control is instructed to make exactly as many stylistic edits as the repair reverts. If the LLM returns a control identical to the input relay — a low-probability failure mode — the pipeline falls back to appending a neutral note (`"*Note: All constraints must be implemented exactly.*"`) so that the executor is guaranteed to see a textually distinct control; this fallback fires on well under 1% of tasks across the four communication agents. Under the temperature-0 decoding used throughout the paper, the repair–control pair is a deterministic function of the annotated relay and re-running the pipeline reproduces the same executor accuracy.

E.4 Inter-Model Accuracy Gap Collapse

At hop 8 on BigCodeBench, the four communication agents produce executor accuracies of $\{0.33, 0.34, 0.24, 0.15\}$ for V4-Flash, Qwen3.5-Flash, GPT-4o-mini, and Llama respectively — a spread of 0.19. After applying R_{all}, the accuracies converge to $\{0.45, 0.44, 0.45, 0.46\}$ — a spread of 0.02, an 89% reduction. Once the annotated changes are reverted, the identity of the communication agent no longer predicts executor success at hop 8, consistent with the main-text claim that CommAudit's annotations capture most of the between-communication agent variance in terminal executor accuracy.

E.5 Executor Swap

The primary executor and the primary annotator both use `deepseek-v4-flash`. To rule out that the repair effect is tied to this model family, we re-score the four specs from Figure 4 — x_0, x_8, $C_{\text{all}}(x_8)$, $R_{\text{all}}(x_8)$ — on the same 100 BigCodeBench tasks per communication agent with a different-family executor, `claude-haiku-4-5-20251001`; chains, annotations, and repair/control specs are unchanged.

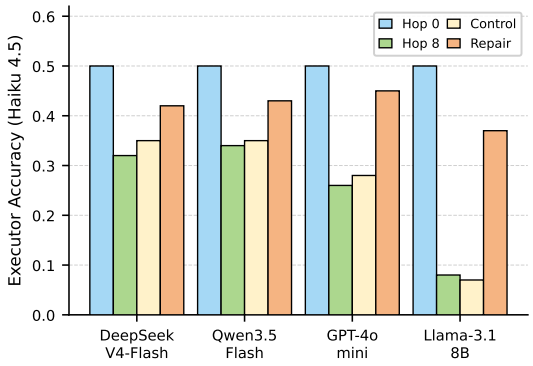

Figure 7: Executor-swap check on hop-8 BigCodeBench: the four specs from Figure 4 re-scored by `claude-haiku-4-5-20251001`.

Under Haiku, Control still tracks Hop 8 within ± 0.03 on every communication agent, and the repair–control differential $\tau = A(R_{\text{all}}) - A(C_{\text{all}})$ is $+0.07$, $+0.08$, $+0.17$, $+0.30$ on V4-Flash, Qwen3.5-Flash, GPT-4o-mini, and Llama, versus $+0.11$, $+0.11$, $+0.21$, $+0.33$ under V4-Flash. τ shrinks by 0.03–0.04 on all four communication agents, yet remains positive on every communication agent. The repair effect is therefore not tied to the executor sharing a model family with the annotator, though its absolute magnitude is executor-dependent.

E.6 Full Four-Type Profiles: Exposure and Population Impact

Table 4 supplies 95% task-level bootstrap CIs (B=5,000) for the values reported in the main-text Table 2, extended to all four change types including Weakening. Point estimates for Loss, Addition, and Mutation are in the main text; the table here shows CI bounds only. Weakening is underpowered ($n_k < 20$) on every (benchmark, communication agent) combination; affected cells in Panel B are marked †.

Table 4: 95% task-level bootstrap CIs (B=5,000) for the exposure profile $\mathbb{E}[n_k]$ (Panel A) and population-impact profile π_k (Panel B) at hop 8, extended to all four change types. Point estimates for Loss / Addition / Mutation are in the main-text Table 2. † marks $n_k < 20$ (underpowered). "—" indicates $n_k = 0$.

		Panel A : Exposure $\mathbb{E}[n_k]$, 95% CI			
Benchmark	Model	Loss	Addition	Mutation	Weakening
BigCodeBench	V4-Flash	$[0.35, 0.60]$	$[2.80, 3.51]$	$[0.90, 1.25]$	$[0.08, 0.22]$
	Qwen3.5-Flash	$[1.00, 1.37]$	$[1.77, 2.28]$	$[1.45, 2.19]$	$[0.01, 0.10]$
	GPT-4o-mini	$[1.44, 2.06]$	$[0.31, 0.61]$	$[0.62, 0.94]$	$[0.03, 0.16]$
	Llama-3.1-8B	$[3.28, 3.83]$	$[0.79, 1.21]$	$[0.20, 0.44]$	$[0.01, 0.08]$
EvoEval	V4-Flash	$[0.32, 0.62]$	$[1.17, 1.71]$	$[0.57, 0.94]$	$[0.00, 0.03]$
	Qwen3.5-Flash	$[0.63, 1.14]$	$[0.98, 1.49]$	$[0.85, 1.28]$	—
	GPT-4o-mini	$[1.72, 2.27]$	$[0.05, 0.20]$	$[0.21, 0.50]$	—
	Llama-3.1-8B	$[2.32, 2.83]$	$[0.04, 0.17]$	$[0.36, 0.65]$	—
IFEval	V4-Flash	$[0.15, 0.39]$	$[1.11, 1.71]$	$[0.43, 0.64]$	$[0.02, 0.13]$
	Qwen3.5-Flash	$[0.44, 0.81]$	$[1.71, 2.77]$	$[0.69, 0.97]$	$[0.07, 0.20]$
	GPT-4o-mini	$[0.40, 0.68]$	$[0.34, 0.65]$	$[0.56, 0.86]$	$[0.10, 0.25]$
	Llama-3.1-8B	$[1.96, 2.45]$	$[0.88, 1.35]$	$[0.64, 0.88]$	$[0.01, 0.11]$

		Panel B : Population impact π_k, 95% CI			
Benchmark	Model	Loss	Addition	Mutation	Weakening
BigCodeBench	V4-Flash	$[-0.02, +0.06]$	$[-0.03, +0.05]$	$[-0.10, +0.01]$	$[-0.03, +0.03]^\dagger$
	Qwen3.5-Flash	$[+0.00, +0.05]$	$[-0.07, +0.02]$	$[-0.04, +0.08]$	$[+0.00, +0.03]^\dagger$
	GPT-4o-mini	$[-0.01, +0.15]$	$[+0.00, +0.07]$	$[-0.04, +0.10]$	$[+0.00, +0.03]^\dagger$
	Llama-3.1-8B	$[+0.16, +0.34]$	$[+0.00, +0.05]$	$[+0.01, +0.08]$	$[+0.00, +0.03]^\dagger$
EvoEval	V4-Flash	$[-0.04, +0.06]$	$[+0.01, +0.15]$	$[-0.01, +0.13]$	$[+0.00, +0.00]^\dagger$
	Qwen3.5-Flash	$[-0.11, +0.01]$	$[-0.06, +0.06]$	$[-0.05, +0.11]$	—
	GPT-4o-mini	$[-0.06, +0.14]$	$[-0.03, +0.00]^\dagger$	$[+0.00, +0.09]$	—
	Llama-3.1-8B	$[+0.18, +0.38]$	$[+0.00, +0.00]^\dagger$	$[+0.05, +0.19]$	—
IFEval	V4-Flash	$[+0.01, +0.08]^\dagger$	$[-0.03, +0.05]$	$[-0.03, +0.05]$	$[+0.00, +0.03]^\dagger$
	Qwen3.5-Flash	$[+0.04, +0.15]$	$[-0.03, +0.07]$	$[-0.05, +0.03]$	$[+0.00, +0.03]^\dagger$
	GPT-4o-mini	$[+0.09, +0.24]$	$[+0.00, +0.05]$	$[+0.04, +0.15]$	$[+0.00, +0.03]^\dagger$
	Llama-3.1-8B	$[+0.52, +0.72]$	$[-0.02, +0.06]$	$[+0.12, +0.28]$	$[+0.00, +0.00]^\dagger$

E.7 Known-Group Classification Protocol

Table 1 reports macro one-vs-rest AUROC for identifying the communication agent from a single hop-8 relay. Sentence-embedding cosine is computed using `text-embedding-3-small` (OpenAI, 1536 dimensions); the cosine is taken between the hop-8 relay $x_{j,8}$ and the original $x_{j,0}$. The classifier for all feature sets is an L2-penalised logistic regression (`C=1.0`, `solver=lbfgs`, `max_iter=2000`), trained and evaluated under 5-fold GroupKFold stratified by task ID so that no task appears in both training and evaluation folds.

F Trajectory Dynamics: Plateau and Embedding Drift

Figure 3 in the main text shows that under the **paraphrase** prompt most of the accuracy decline occurs in the first two or three hops, followed by smaller changes over the remaining observed hops. Three additional views examine this early concentration: Figure 8 resolves correctness churn per hop transition, and Figures 9 and 10 track the drift of the relay in embedding space from two complementary perspectives.

Figure 8 shows the per-hop correctness *churn* as a line chart: each line tracks one (communication agent, benchmark) chain, plotting the fraction of tasks whose executor correctness flipped at each hop transition $t-1 \to t$. Churn concentrates in the early transitions and generally declines over the

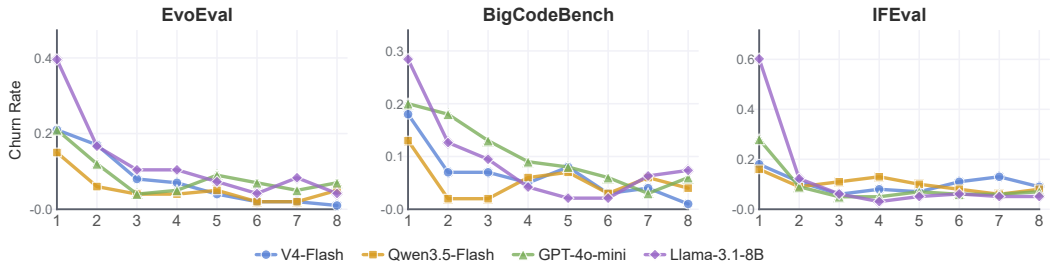

Figure 8: Per-hop correctness churn under **paraphrase**. Each line is one (communication agent, benchmark) chain; the y-axis is the fraction of tasks whose executor correctness flipped at hop transition $t{-}1 \to t$, for $t \in \{1, \ldots, 8\}$. Churn concentrates in the first two or three transitions and declines thereafter — a hop-level reading of the plateau pattern in Figure 3.

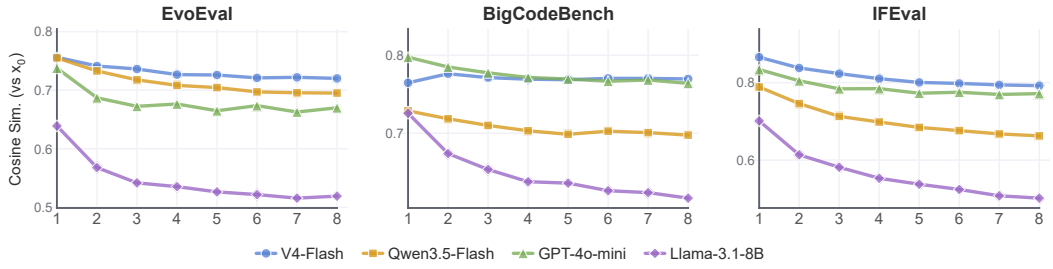

Figure 9: Cumulative embedding drift under **paraphrase**: $\cos(x_t, x_0)$ measures how far the hop-t relay has moved from the original prompt x_0. All four communication agents show the sharpest drop in the first two or three hops; thereafter the curves flatten, indicating that most semantic departure from the original occurs early in the chain.

observed horizon, with Llama-3.1-8B retaining non-trivial churn later than the other three communication agents. Figure 9 shows cumulative drift: $\cos(x_t, x_0)$ falls sharply in the first two or three hops and then flattens, indicating that most semantic departure from the original prompt occurs early. Figure 10 shows step-to-step similarity: $\cos(x_t, x_{t-1})$ rises steeply over the same early window and plateaus near 1, meaning successive relays become nearly identical to each other. Together, the increasingly self-similar relays and stabilizing executor behavior suggest the chain is approaching a communication agent-specific attractor within the observed eight-hop horizon.

G Theoretical Details

This appendix contains the full proofs of Proposition 1 (§G.1) and Proposition 2 (§G.3), along with the critical-bit construction referenced in §3.2 and its symmetric counterpart (§G.2).

G.1 Full Proof of Proposition 1

We work on an underlying probability space $(\Omega_0, \mathcal{F}_0, \mathbb{P})$ that carries the task draw $J : \Omega_0 \to \mathcal{J}$, the relay chain $X_0, X_1, \ldots, X_H$ with $X_t : \Omega_0 \to \mathcal{X}$, and any randomization used by decision rules. All spaces are endowed with σ-algebras that make these mappings measurable. A *decision rule at hop* t is a Markov kernel $\pi_t \colon \mathcal{X} \to \Delta(\mathcal{A})$: for every measurable $B \subseteq \mathcal{A}$ the map $x \mapsto \pi_t(B \mid x)$ is measurable, and $\pi_t(\cdot \mid x)$ is a probability measure for every x. Given π_t and X_t, the action $a_t \sim \pi_t(\cdot \mid X_t)$ is well-defined and measurable.

The utility $u \colon \mathcal{J} \times \mathcal{A} \to \mathbb{R}$ is bounded and measurable, and the communication agent kernel $K_t(dx' \mid x) = q_M(dx' \mid x; \pi_R)$ carries X_t to X_{t+1}. Under (P1)–(P2) the joint law satisfies $\mathbb{P}(X_{t+1} \in \cdot \mid J, X_t) = K_t(\cdot \mid X_t)$, i.e., J is conditionally independent of X_{t+1} given X_t.

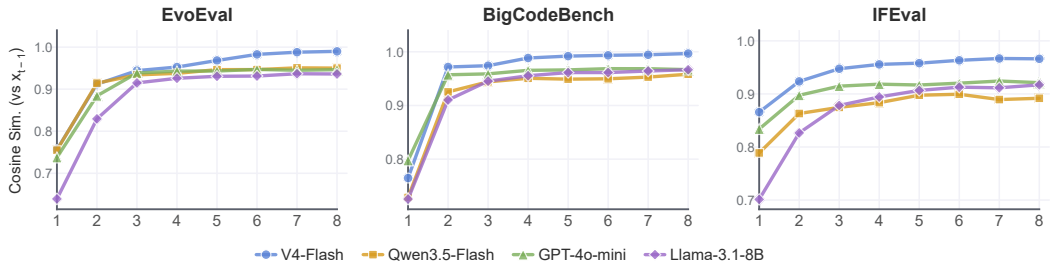

Figure 10: Step-to-step similarity under **paraphrase**: $\cos(x_t, x_{t-1})$ measures how much each relay changes relative to the immediately preceding one. Values rise steeply in the first two or three hops and plateau near 1, showing that successive relays become nearly identical once the chain settles — the mirror image of the cumulative drift in Figure 9.

Proof of Proposition 1. Fix any decision rule $\pi_{t+1}\colon \mathcal{X} \to \Delta(\mathcal{A})$ at hop $t+1$. Define the composite kernel $\tilde{\pi}_t\colon \mathcal{X} \to \Delta(\mathcal{A})$ by

$$\tilde{\pi}_t(B \mid x) \;=\; \int_{\mathcal{X}} \pi_{t+1}(B \mid x')\, K_t(dx' \mid x), \qquad B \subseteq \mathcal{A} \text{ measurable.} \tag{12}$$

The map $x \mapsto \tilde{\pi}_t(B \mid x)$ is measurable by the standard measurability property of integration against a Markov kernel, given the joint measurability of $\pi_{t+1}(B \mid \cdot)$ and $K_t(\cdot \mid \cdot)$; $\tilde{\pi}_t(\cdot \mid x)$ is a probability measure for each x. Hence $\tilde{\pi}_t$ is a valid Markov kernel and admissible in the sup defining $V_t^\star(u)$.

Let $\tilde{a}_t \sim \tilde{\pi}_t(\cdot \mid X_t)$. By construction, sampling $\tilde{a}_t$ has the same conditional law given X_t as first sampling $X_{t+1} \sim K_t(\cdot \mid X_t)$ and then $a_{t+1} \sim \pi_{t+1}(\cdot \mid X_{t+1})$. Combining this with conditional independence of J and X_{t+1} given X_t,

$$\mathbb{E}\big[u(J, \tilde{a}_t)\big] = \mathbb{E}\left[\iint u(J, a)\, \pi_{t+1}(da \mid x')\, K_t(dx' \mid X_t)\right] \tag{13}$$

$$= \mathbb{E}\big[\mathbb{E}\big[u(J, a_{t+1}) \,\big|\, X_t\big]\big] \;=\; \mathbb{E}\big[u(J, a_{t+1})\big]. \tag{14}$$

Thus $V_t^\star(u) \geq \mathbb{E}[u(J, \tilde{a}_t)] = \mathbb{E}[u(J, a_{t+1})]$; taking the sup over π_{t+1} yields $V_t^\star(u) \geq V_{t+1}^\star(u)$. □

Equivalence versus monotonicity. Equality $V_{t+1}^\star(u) = V_t^\star(u)$ for every bounded measurable u is a much stronger statement than the monotonicity of Proposition 1: by the Blackwell–Sherman–Stein equivalence (Blackwell, 1953), it requires the pair (X_t, X_{t+1}) to be Blackwell-equivalent as experiments for J, which is characterized by the existence of a *reverse* kernel $\bar{K}_t\colon \mathcal{X} \to \Delta(\mathcal{X})$ that recovers the joint law of (J, X_t) from the joint law of (J, X_{t+1}). Under (P1)–(P2), the forward kernel $K_t = q_M(\cdot \mid \cdot; \pi_R)$ is generated by a language model producing free-form text conditioned on the previous relay. A relay can discard instance-specific content—a boundary condition, a constraint particular to x_t—that is not recoverable from X_{t+1} even when conditioned on J; in such cases no reverse kernel reconstructing the joint law of (J, X_t) from (J, X_{t+1}) can exist. The relay is therefore a *potential* proper garbling: $V_{t+1}^\star(u) < V_t^\star(u)$ is possible for some u at every hop, as the critical-bit examples of §G.2 illustrate, and lost task-specific content cannot be restored by any downstream decoder.

G.2 Critical-Bit Examples: Bidirectional Decoupling of Similarity and Task Loss

The following two constructions show that the shared-information ratio $I(X_0; X_1)/H(X_0)$ can be driven to either extreme independently of whether the task-relevant content is preserved.

Remark 3 (High similarity, task bit lost). Let $U \sim \mathrm{Unif}\{0,1\}^n$ and $B \sim \mathrm{Bernoulli}(1/2)$ with $U \perp B$. Take the original specification $X_0 = (U, B)$ and let the task depend only on B, formalized by a utility u with $u(1,1) = u(0,0) = 1$, $u(1,0) = u(0,1) = 0$. Consider a relay that drops the critical bit:

$$X_1 \;=\; U. \tag{15}$$

Then $I(X_0; X_1) = H(U) = n$ and $H(X_0) = n + 1$, so the shared-information ratio is

$$\frac{I(X_0; X_1)}{H(X_0)} = \frac{n}{n+1} \xrightarrow[n\to\infty]{} 1. \tag{16}$$

Yet X_1 carries no information about B: any decision rule $\pi\colon \mathcal{X} \to \Delta(\{0,1\})$ achieves $\mathbb{E}[u(B, A)] = 1/2$, so $V_0^\star(u) = 1$ while $V_1^\star(u) = 1/2$.

Remark 4 (Low similarity, task bit preserved). Keep $X_0 = (U, B)$ and the same task. Let $R \sim \mathrm{Unif}\{0,1\}^n$ be independent of X_0 and consider a relay that replaces the surface content but preserves the critical bit:

$$X_1' = (R, B). \tag{17}$$

Then $I(X_0; X_1') = H(B) = 1$ and the shared-information ratio is

$$\frac{I(X_0; X_1')}{H(X_0)} = \frac{1}{n+1} \xrightarrow[n\to\infty]{} 0. \tag{18}$$

Yet the deterministic rule $A = B$ achieves $V_1'^\star(u) = 1 = V_0^\star(u)$: the relay is Blackwell-equivalent to X_0 for this task.

The two constructions decouple the shared-information ratio from task-relevant preservation in both directions. The two constructions illustrate that a similarity-based audit can, in principle, fail to detect consequential losses or flag inconsequential ones: what a relay preserves must be identified relative to a downstream outcome, not from global overlap alone. This motivates pairing the source-anchored constraint measurement of §3.3 with the matched interventions of §4.2.

G.3 Full Proof of Proposition 2, and Matched Intervention as an Operational Fiber Check

We work in the setting of §4.3: Ω is the space of task–relay states $\omega = (j, x_{j,0}, x_{j,t})$ endowed with a σ-algebra $\mathcal{F}$, the summary $S\colon \Omega \to \mathcal{Z}$ is measurable into a measurable space $(\mathcal{Z}, \mathcal{G})$, and the executor-relative consequence $r_E\colon \Omega \to \mathbb{R}$ is bounded and measurable. For the factorisation direction of Proposition 2 we further assume one of the following two hypotheses:

(H1) $(\Omega, \mathcal{F})$ and $(\mathcal{Z}, \mathcal{G})$ are both standard Borel spaces; or

(H2) $\mathcal{G}$ is the pushforward σ-algebra $\{C \subseteq \mathcal{Z} : S^{-1}(C) \in \mathcal{F}\}$ (with no restriction on Ω).

Both hypotheses hold trivially in our concrete instances: Ω is countable (task–relay states are enumerable from the finite task set and the communication agent's countable output space), and the audit summary takes values in a countable set (the constraint state in tuples of $\{F, L, M, W\}$, together with a finite addition set drawn from a countable universe, and the count vector N in $\mathbb{Z}_{\geq 0}^4$). All these spaces are naturally endowed with their power-set σ-algebras, which makes them standard Borel; and with $\mathcal{F}$ the power set of Ω, the pushforward σ-algebra on $\mathcal{Z}$ likewise coincides with its power set. Let $S(\Omega) = \{S(\omega) : \omega \in \Omega\}$.

Proof of Proposition 2. $(ii) \Rightarrow (i)$. Assume r_E is constant on every fiber of S. We show that r_E is $\sigma(S)$-measurable; the Doob–Dynkin lemma then yields a measurable $f\colon \mathcal{Z} \to \mathbb{R}$ with $r_E = f \circ S$ pointwise on Ω, and integrating against any Q on Ω gives $\mathbb{E}_Q[r_E] = \mathbb{E}_Q[f \circ S]$. Values of f on $\mathcal{Z} \setminus S(\Omega)$ contribute zero mass to $\mathbb{E}_Q[f \circ S]$ under any Q supported on Ω and may be set to any measurable extension.

Under (H2), $\sigma(S)$-measurability is direct: for every Borel $D \subseteq \mathbb{R}$, the level set $\{r_E \in D\} \in \mathcal{F}$ is a union of S-fibers by constancy of r_E on fibers, so $\{r_E \in D\} = S^{-1}(S(\{r_E \in D\}))$ with $S(\{r_E \in D\}) \in \mathcal{G}$ by definition of the pushforward, hence $\{r_E \in D\} \in \sigma(S)$.

Under (H1), standard-Borel $\mathcal{Z}$ has measurable singletons, so the fibers $S^{-1}(\{z\})$ lie in $\sigma(S)$ and are its atoms; standard-Borel Ω then places us in a Blackwell measurable space, in which any $\mathcal{F}$-measurable function that is constant on the atoms of a countably generated sub-σ-algebra of $\mathcal{F}$ is measurable with respect to that sub-σ-algebra (Blackwell, 1956). Since $\sigma(S)$ is countably generated (as $\mathcal{G}$ is) and r_E is constant on its atoms, r_E is $\sigma(S)$-measurable.

$(i) \Rightarrow (ii)$. Assume (i) holds and fix any $\omega, \omega' \in \Omega$ with $S(\omega) = S(\omega')$. Applying (i) to the point-mass $Q = \delta_\omega$,

$$r_E(\omega) \;=\; \mathbb{E}_{\delta_\omega}[r_E] \;=\; \mathbb{E}_{\delta_\omega}[f \circ S] \;=\; f(S(\omega)). \tag{19}$$

The same argument with $Q = \delta_{\omega'}$ gives $r_E(\omega') = f(S(\omega'))$. Since $S(\omega) = S(\omega')$, $r_E(\omega) = r_E(\omega')$. □

The population-impact profile is an instance. Let $Q_{m,t}$ denote the empirical distribution over task–relay states produced by communication agent m at hop t on a benchmark. With $r_{E,k}(\omega) = D_k(\omega)\Delta_{E,k}(\omega)$ as in §4.3,

$$\pi_{m,t,k} \;=\; \mathbb{E}_{Q_{m,t}}\big[r_{E,k}(\omega)\big]. \tag{20}$$

For a single fixed $f_k \colon \mathcal{Z} \to \mathbb{R}$ to represent $\pi_{m,t,k}$ across every $(m, t, k, \text{benchmark})$, Proposition 2(ii) requires $r_{E,k}$ to be constant on every fiber of S. At $S = N$, D_k is itself determined by the count vector; on the affected subset, the condition therefore requires $\Delta_{E,k}$ to be constant among states with identical count vectors.

Matched intervention as an operational fiber check. The matched-intervention design of §4.2 implements this comparison directly at the count-vector level. Given a hop-t relay $x_{j,t}$ with $N_{j,t,k} \geq 1$, the targeted repair R_k moves $x_{j,t}$ to a state with type-k count zero, leaving the count fiber via the reverted constraint. The matched control C_k moves $x_{j,t}$ to a state with the same type-k count but equal-budget non-constraint edits, staying within the count fiber through stylistic variation. The pair $\big(R_k(x_{j,t}),\, C_k(x_{j,t})\big)$ therefore isolates the effect of leaving a count fiber against the effect of moving along it. The within-fiber analysis below tests this requirement directly by comparing $\Delta_{E,k}$ across affected tasks with the same original count vector.

G.4 Empirical Within-Fiber Variance of the Matched Contrast

To quantify how far Proposition 2's fiber condition deviates from equality on the environments we evaluate, we group affected tasks by their full count vector $N_{j,8} = (N_L, N_M, N_W, N_+)$ and, per (benchmark, communication agent, change type k), compute the within-fiber variance of the matched contrast $\Delta_{E,k}(\omega_j) = Y_j(R_k(x_{j,8})) - Y_j(C_k(x_{j,8}))$ pooled over fibers with ≥ 2 affected tasks. A fiber-condition-preserving audit would produce $\mathrm{Var}_{\mathrm{within}}/\mathrm{Var}_{\mathrm{total}} \to 0$.

Across the 32 non-sparse cells (≥ 5 usable fibers) over the three benchmarks and four communication agents, 30 have $\mathrm{Var}_{\mathrm{within}}/\mathrm{Var}_{\mathrm{total}} \geq 0.5$; the median ratio is 1.02 with interquartile range $[0.87, 1.14]$. Values above 1 arise when small usable-fiber populations amplify sampling noise. Only two non-sparse cells fall below 0.5: BigCodeBench/Llama-3.1-8B/Mutation (0.36) and BigCodeBench/V4-Flash/Loss (0.32); both involve small affected subsets.

H Limitations

Our main analyses use `deepseek-v4-flash` as the executor across three English, programmatically-graded benchmarks; for the repair experiment (§5.3) we additionally re-score with a different-family executor (App. E.5). A broader executor sweep and generalization to open-ended tasks, non-code/math domains, or non-English relays remain open.

The annotation, repair, and matched-control operators are all LLM-generated. The reliability check reported in App. C.2 covers the change-typing step only ($\kappa = 0.849$); change detection is not independently validated, and all impact estimates should be interpreted relative to that pipeline.

The atomic-constraint decomposition (§3.3) and type-specific repair (§4.2) treat the specification as a flat set of independent constraints. Real tasks may include conditional constraints or dependency chains (e.g., a constraint that is meaningful only when another holds); on such tasks, a type-k-only repair can restore its target while leaving a dependent constraint out of sync, producing a relay that is locally consistent but not globally coherent as a specification. Impact estimates from these tasks should be interpreted relative to this decomposition.

I Design Implications for Multi-Agent Communication

Executor outcomes are the right signal for message quality. Message quality between agents should be judged by what the receiver does with the message, not by text similarity alone. In the communication agent-identification task of §5.3, length ratio and word-count delta reach 0.668 and 0.677 macro AUROC, sentence-embedding cosine reaches 0.776, the constraint profile reaches 0.797, and their combination reaches 0.888. The matched-control experiment supplies the downstream evidence: reverting the annotated constraint changes improves executor accuracy by 0.11–0.33 points across the four communication agents, while a matched stylistic control stays within ± 0.02 of the raw relay (Figure 4). Audit protocols for inter-agent messages should therefore combine semantic diagnostics with a downstream behavioral probe.

Communication quality is a downstream-conditioned property. The quality of an intermediate message is a joint property of sender, message, receiver, and task. In our audit (§5.4, Table 2), exposure fails to provide a stable calibration of population impact across change types and benchmarks; Proposition 2 makes precise the invariance any exposure-only surrogate would require: count-equivalent task–relay states must produce identical executor consequence. Scoring must condition on the receiver and the task family it will be asked to solve.

The inter-agent performance gap is encoded in constraint changes. At hop 8 on BigCodeBench, the four communication agents span 0.19 in executor accuracy; reverting CommAudit's annotated changes closes this spread to 0.02, an 89% reduction (App. E.4). Once constraint changes are repaired, communication agent identity no longer predicts executor success: the performance gap is almost entirely encoded in constraint-change profiles, not in the residual surface-text differences that repair leaves intact.

Change-type frequency does not identify repair priorities. Knowing that a relay contains two losses and one mutation does not predict how executor performance responds to repairing those changes. In 94% of non-sparse (benchmark, communication agent, change type) cells, within-fiber variance in the matched contrast $\Delta_{E,k}$ equals at least half of its total variance (App. G.4): tasks with identical count vectors yield heterogeneous repair outcomes. An audit protocol that ranks change types by frequency cannot identify which to prioritize for repair. CommAudit's matched intervention design, pairing type-specific repairs with matched stylistic controls, provides the only operational test of which changes actually move executor performance. Any content-level message audit should include this intervention step.

J Concrete Case Studies

The following five cases illustrate the Loss / Mutation / Addition taxonomy across BigCodeBench and IFEval. Cases 1–3 give one clean example of each type; Case 4 is a same-count-fiber pair with opposite repair outcomes (an operational instantiation of Proposition 2's fiber failure); Case 5 shows loss on IFEval. ✓ = executor correct; × = executor incorrect.

Case 1: Loss — BigCodeBench / Llama-3.1-8B / `BigCodeBench/1083`. *Annotation record (type = loss, $n_k = 6$):*

> Change 1 [LOSS] `original_text`: Checks if the input data dictionary contains the required keys ('Salary_String' and 'Experience'). Raises a ValueError if the necessary keys are missing. `rewrite_text`: The function should raise a meaningful error if the input dictionary is invalid or if data processing fails. *Impact: Executor is not told the specific required keys or that a ValueError must be raised, so validation logic and exception type may differ or be omitted.* Change 2 [LOSS] `original_text`: Empty Data Handling: Checks if the DataFrame is empty. If so, it returns a default Axes instance with labeled axes but no data plotted. `rewrite_text`: N/A. *Impact: Executor will not implement the empty-data branch, causing different behavior (e.g., errors or empty plots) when the DataFrame is empty.*

Executor outcomes:

	Hop-8	Repair	Control
BigCodeBench/1083	×	✓	×

Case 2: Mutation — BigCodeBench / GPT-4o-mini / `BigCodeBench/318`. *Annotation record (type = mutation, $n_k = 1$):*

> Change 1 [MUTATION] `original_text:` `def task_func(points_count=1000, radius=1):` `rewrite_text:` `def generate_random_points_in_circle(num_points=1000, radius=1):` *Impact: The function name and parameter name are changed, so any code calling the function by the original name or using the original parameter name will fail.*

Executor outcomes:

	Hop-8	Repair	Control
BigCodeBench/318	×	✓	×

Case 3: Addition — BigCodeBench / V4-Flash / `BigCodeBench/198`. *Annotation record (type = addition, $n_k = 3$):*

> Change 1 [ADDITION] `original_text:` N/A `rewrite_text:` No plot should be generated in this case. *Impact: Executor will skip plotting when data is empty, whereas original did not specify this behavior.* Change 2 [ADDITION] `original_text:` N/A `rewrite_text:` Use `statistics.mean` to compute the average. *Impact: Executor is forced to use `statistics.mean` instead of `numpy.mean` or other methods, changing implementation.*

Executor outcomes:

	Hop-8	Repair	Control
BigCodeBench/198	×	✓	×

Case 4: Same-count-fiber contrast — BigCodeBench / GPT-4o-mini. Both tasks share count vector $N_{j,8} = (L{=}2,\ A{=}0,\ M{=}0,\ W{=}0)$ yet produce opposite repair outcomes, directly demonstrating that the fiber invariance condition of Proposition 2 fails. For each task we show one representative loss record; the second loss is omitted for brevity.

Task A: BigCodeBench/548 ($\Delta_k = +1, Y_{\text{repair}} = 1, Y_{\text{ctrl}} = 0$)

> [LOSS] `original_text:` Complete the following Python function. Include all required imports and the full function body. `def task_func(string_length=100):` `rewrite_text:` Choose a suitable name for the function, and set a default string length of 100 characters. *Impact: The function name is no longer fixed to 'task_func', so the executor may choose a different name, breaking any code that expects the original name.*

Task B: BigCodeBench/817 ($\Delta_k = -1, Y_{\text{repair}} = 0, Y_{\text{ctrl}} = 1$)

> [LOSS] `original_text:` Logs are written to a file named 'task_func.log' with encoding 'utf-8' and logging level DEBUG. The log file is created by the function or overwritten if already exists. `rewrite_text:` Set up the logging to output to a file named 'task_func.log' using UTF-8 encoding, and ensure it operates at the DEBUG level. If the log file already exists, it should be overwritten. *Impact: The REWRITE does not specify that the log file is created in the directory given by log_path; it only says 'output to a file named task_func.log'.*

Side-by-side executor outcomes:

	Hop-8	Repair	Control
Task A (BigCodeBench/548)	×	✓	×
Task B (BigCodeBench/817)	✓	×	✓

Case 5: Loss — IFEval / Llama-3.1-8B / `2063`. *Annotation record (type = loss, $n_k = 8$):*

> Change 1 [LOSS] `original_text`: Write an advertisement for the company. `rewrite_text`: Develop a marketing strategy for Titan clothing. *Impact: The response would shift from writing an advertisement to creating a broader marketing strategy, producing a materially different deliverable.* Change 2 [LOSS] `original_text`: Make sentences short. `rewrite_text`: N/A. *Impact: The executor may write long sentences, violating the original brevity requirement.*

Executor outcomes:

	Hop-8	Repair	Control
2063	×	✓	×