Bayes-Sufficient Compression Is Not Enough: How Does Communication Help Multi-Agent Systems?

Yi Xie[1] **Zhanke Zhou**[2] **Yi Fan**[3‡] **Yong Ge**[4] **Bo Han**[2] **Bo Liu**[1†]
[1]Department of Electrical & Computer Engineering, University of Arizona
[2]TMLR Group, Department of Computer Science, Hong Kong Baptist University
[3]Amazon Web Services
[4]Department of Management Information Systems,
Eller College of Management, the University of Arizona
yix@arizona.edu, cszkzhou@comp.hkbu.edu.hk, fnyi@amazon.com,
yongge@arizona.edu, bhanml@comp.hkbu.edu.hk, boliu@arizona.edu

Abstract

Multi-agent LLM systems pair a sender with broad context and an executor with a limited local view. We study when a short message improves the executor's next decision, when raw context is preferable, and when a stronger sender helps. Our framework, *receiver-relative bounded coordination*, expresses message utility as receiver gain minus protocol tax. Compression beats raw context when tax savings exceed losses from omitted information and decoder mismatch. Even *Bayes-sufficient* compression can fail when a bounded executor cannot use its surface form. A three-stage decomposition separates externalization, absorption, and *action closure*, explaining how errors remain after the correct content reaches the receiver. Under a single-crossing condition, sender upgrades help above a receiver-burden threshold. Across six benchmarks, the same Qwen protocol raises ContextBench joint accuracy from 0.633 to 0.775 but lowers ToolSandbox from 0.889 to 0.653. Fixed-message replay reveals closure failures despite correct artifact recovery. These results guide an inference-time selector that improves the accuracy-cost frontier on the evaluated communication regimes. Evaluation artifacts: https://github.com/Ydc/Communication.

1 Introduction

Multi-agent LLM systems increasingly split work across agents with asymmetric contexts, capabilities, and budgets (Yi et al., 2025). In coding, web, and tool-use workflows, a main agent may inspect repository state, prior failures, tool traces, or verifier outputs (Dong et al., 2025). The executor, by contrast, must edit one file, choose one tool, or execute the next action under tighter context and latency limits (OpenAI, 2025; Srivastav and Gil Guzman, 2025; Anthropic, 2025b,a). This split makes communication a central protocol decision. Exposing the full context forces the executor to rediscover relevance within budget; a short message can expose the missing target directly, but may also omit useful information or present it in a form the executor cannot easily use (Shi et al., 2026).

Recent evidence suggests that multi-agent LLM performance depends less on the number of agents than on how agents coordinate (Anthropic, 2025a). Matched-budget studies report coordination tax and topology-dependent error amplification; failure analyses trace breakdowns to specification gaps, inter-agent misalignment, and weak verification; production harnesses depend heavily on routing, aggregation, and context control (Kim et al., 2025; Cemri et al., 2025; Anthropic, 2025b). These

†Corresponding author.
‡This work is not related to the author's position at Amazon Inc.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).



Figure 1: **Bayes-sufficient is not the same as receiver-usable.** Illustrative example, constructed for exposition rather than taken from model logs. Both replies are correct and Bayes-sufficient: each carries the information that determines the right action. The verbose reply leaves the user unsure what to do, whereas the short reply leads directly to the action. The framework attributes this difference to the receiver's bounded decoder and the protocol tax of the message (Section 2); Theorem 3 shows that Bayes sufficiency alone does not make a message receiver-optimal.

observations raise a step-level question: *when does a message improve the executor's next decision rather than adding burden?* Aggregate task metrics conflate these effects. A final failure may come from an uninformative message, a useful message the receiver fails to use, or an error carried across steps. Protocol design therefore needs to evaluate each message at the receiver's next decision.

We study this question through *receiver-relative bounded coordination*. The main agent, or sender, has a broader context and sends a bounded message to the executor, or receiver, which acts from a limited local view. Three commitments distinguish this framework from aggregate analyses: the step is the unit of analysis, the *action-sufficient target* rather than total information defines what is useful, and the executor is modeled as a bounded decoder, so an information-sufficient message need not be usable. Message value decomposes into the executor's next-step gain minus the protocol tax induced.

The framework connects four aspects of communication. Compression trades tax savings against omitted information and decoder mismatch. Bayes sufficiency preserves information, while receiver compatibility determines whether the executor can use it (Figure 1). Fixed-message replay separates content recovery from *action closure*, the conversion of recovered content into an emitted action. Finally, under single crossing, a stronger sender becomes useful above a receiver-burden threshold.

We tested these relationships with paired receiver-only, full-context, and multi-agent runs; replay with a fixed sender message; and sender swaps. The six benchmarks cover function calling, stateful tools, code context, retrieval, software repair, and customer service. Communication helped when the local view lacked useful context, while its effect varied across receiver interfaces on locally informative tasks. These observations also guide a selector that chooses communication actions by their estimated gain, protocol burden, and cost.

Contributions.

- We introduce a step-level theory of receiver-relative bounded coordination, with formal results on representation-aware compression, information sufficiency versus executor usability, post-message action-closure loss, and sender-upgrade burden thresholds (Section 2).
- We develop an inference-time protocol selector that scores communication actions using the same gain, tax, decoder-mismatch, sender-upgrade, and residual-risk quantities (Section 3).
- We provide empirical tests across six benchmark families, showing compression sign flips, receiver-side action-closure failures, sender-upgrade thresholds, and accuracy-cost improvements from adaptive protocol selection (Section 4).

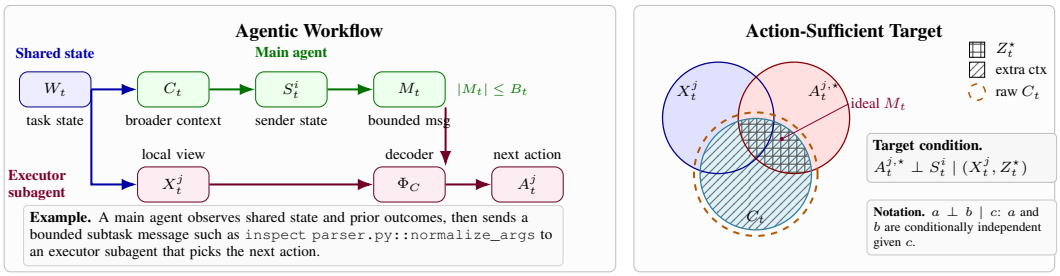

Figure 2: **Setup and action-sufficient target.** *Left:* one decision step in an agentic workflow; Φ_C is the bounded decoder class the executor can implement. *Right:* the Venn view of $Z_t^\star$.

2 A Theoretical Framework for Bounded Coordination

We formalize bounded communication at the receiver's next decision. The framework is *receiver-relative*: communication is useful only to the extent that it improves the deployed receiver's next-step decision. The same quantities are extended to a discrete-action control rule in Section 3. Full proofs and theoretical supplements are in Appendix B.

2.1 Setup

At step t, the team occupies a shared task state W_t. Receiver j acts from local view $X_t^j = g_t^j(W_t, \xi_t^j)$, while sender i has broader context C_t and sender state $S_t^i = s_t(C_t)$; let M_t denote the sender-to-receiver message. We compare three endpoint protocols:

$$\text{RO} : M_t = \varnothing, \qquad \text{MA} : M_t = e_t(S_t^i),\ |M_t| \le B_t, \qquad \text{SF} : M_t = C_t,$$

where RO is receiver-only, MA is bounded multi-agent communication, and SF is the raw-context comparator. The receiver then chooses A_t^j from (X_t^j, M_t) through a bounded decoder class Φ_C defined below. Here and below, the subscript C in $\Phi_C, A^\star_{C,\ell}, R^\star_C$ denotes the receiver's bounded compute class (not the broader context C_t), and $a \perp b \mid c$ denotes conditional independence.

Definition 2.1 (Action-Sufficient Coordination Target)**.** *A variable $Z_t^\star \in \mathcal{Z}_t$ is an* action-sufficient coordination target *for receiver j if $A_t^{j,\star} \perp S_t^i \mid (X_t^j, Z_t^\star)$, where $A_t^{j,\star}$ is the optimal next action.*

Intuitively, $Z_t^\star$ is the sender-side state sufficient for the receiver's next action beyond X_t^j (not necessarily unique or minimal); in our experiments, $Z_t^\star = (Z_t^{\text{art}}, Z_t^{\text{act}})$ is the next target artifact and action type. The decoder class Φ_C consists of admissible maps from (X_t^j, M_t) to internal decision representations; we use a larger probe class $\Phi_{\bar{C}} \supseteq \Phi_C$ to separate message-side recoverability from receiver-side usability, and assume drop-message closure: if $\phi \in \Phi_C$ can process (X_t^j, M_t), then Φ_C also contains an admissible map that ignores M_t.

2.2 One-Step Utility

Fix a step and let $\delta \in \{0, 1\}$ indicate whether the message is exposed: the receiver conditions on (X_t^j, M_t) when $\delta = 1$ and on X_t^j when $\delta = 0$. Let $R^\star_C(\delta)$ be the lowest risk attainable by the bounded receiver class Φ_C on target $Z_t^\star$, and $R_\rho(\delta)$ be the corresponding risk of the deployed protocol ρ.

The receiver-relative gain $A^\star_{C,\ell}(M_t; Z_t^\star)$ is the class-optimal risk reduction from seeing the message; for any other side information Y_t, $A^\star_{C,\ell}(Y_t; Z_t^\star)$ is defined by replacing M_t with Y_t. The protocol tax $\tau_{\rho,\ell}(M_t)$ is the increase in the gap between the deployed risk and the class-optimal risk after the message is exposed. It is signed: positive tax means the message moves the deployed protocol farther from the best decoder in the same bounded class. In multi-agent LLM systems, positive tax can reflect extra parsing burden, distraction from irrelevant content, and prompt-interface mismatch, or other protocol-induced errors. A message, therefore, helps exactly when its receiver-side gain exceeds the extra protocol tax it creates.

Proposition 1 (Communication-Tax Identity)**.** *For any deployed protocol ρ and step loss ℓ,*

$$U_{\rho,\ell}(M_t; Z_t^\star) = R_\rho(0) - R_\rho(1) = A^\star_{C,\ell}(M_t; Z_t^\star) - \tau_{\rho,\ell}(M_t). \tag{1}$$

Proposition 1 is the accounting identity used throughout the theory. Later results ask when the gain grows, when the tax shrinks, and when their difference changes sign. See Appendix B.1 for the proof.

2.3 Representation-Aware Compression

Compression is useful only when the protocol tax it saves exceeds the information and usability costs it introduces. We separate two terms: omitted target information $\mathcal{I}_{\mathrm{omit}}$, the target-relevant information in raw context C_t but not recovered from (X_t^j, M_t); and decoder mismatch $\Lambda_C(M_t \to C_t)$, the difference between how easily the bounded class Φ_C can use the compressed message and how easily it can use raw context (positive Λ_C favors raw; negative mismatch favors compression). Log-loss definitions are in Appendix B.1.

Theorem 2 (Compression Wins by Saving More Tax Than It Loses)**.** *Under log-loss, if $M_t = f(C_t)$, then*

$$\Delta U_{\mathrm{cmp-raw}} = \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - \mathcal{I}_{\mathrm{omit}} - \Lambda_C(M_t \to C_t). \tag{2}$$

Here $\Delta U_{\mathrm{cmp-raw}}$ is the deployed utility of the compressed message minus that of the raw context, and $\Delta\tau_{\mathrm{raw}\to\mathrm{cmp}}$ is the tax saved by not exposing raw context. Thus, an action-sufficient message wins immediately when it reduces tax and is no harder for the executor to use than raw context. Appendix B.1 provides the proof, the expanded entropy form of Eq. (2), and an equivalent receiver-class certificate in terms of the representation gap $B_C(M_t, C_t)$.

Bayes sufficiency is an information-level property, not a usability guarantee. In an LLM harness, the deployed executor is a fixed model with a fixed prompt, parser, context window, and tool interface, and the corresponding bounded receiver class is *representation-sensitive*: it is not closed under arbitrary recodings of the message surface. We therefore require compressed messages to preserve target-relevant information while using a strictly smaller budget than the raw context; the formal condition excludes degenerate copies and same-size recodings.

Definition 2.2 (Proper Bayes-Sufficient Compression)**.** *For $0 < \alpha < 1$, $M_t = T(C_t)$ is an α-*proper Bayes-sufficient compression *of C_t for target $Z_t^\star$ and receiver view X_t^j if (i) $I(Z_t^\star; C_t \mid X_t^j, M_t) = 0$ and (ii) $H(M_t) \le \alpha H(C_t)$.*

Even a proper Bayes-sufficient compression can fail when it is hard for the bounded executor to use.

Theorem 3 (Bayes-Sufficient Compression Need Not Be Receiver-Optimal)**.** *For any $0 < \alpha < 1$ and $\varepsilon > 0$, there exist a task, an α-proper Bayes-sufficient compression (Definition 2.2), and a representation-sensitive receiver class on which the raw context attains zero error, while the compressed message incurs an error of at least $1/2 - \varepsilon$.*

Concretely, a message may contain the correct artifact and action information, yet present it in a surface form that the executor fails to use. Appendix B.2 gives the construction, the finite-alphabet entropy version of the proper-compression condition, and the corresponding receiver-compatible sufficiency condition. It also shows that sufficiency becomes useful when tax savings exceed the receiver-class representation gap.

2.4 Where Post-Message Loss Remains

A useful message need not become a correct action. To localize this failure, we expose one internal bottleneck of the executor. A receiver is *bottleneck-factorized* if it first maps (X_t^j, M_t) to an internal target representation $\widehat{Z}_t$, and then maps $(X_t^j, \widehat{Z}_t)$ to the emitted action A_t^j. Conditioned on the local view, this gives the chain $Z_t^\star \to C_t \to M_t \to \widehat{Z}_t \to A_t^j$, where arrows denote conditional-independence structure rather than causal timing.

Externalization is sender-side loss: useful content in C_t does not reach M_t. *Absorption* is receiver-side uptake loss: useful content in M_t does not enter $\widehat{Z}_t$. *Action closure* is receiver-side decision loss: recovered content in $\widehat{Z}_t$ does not become the correct final action.

Theorem 4 (Post-Message Loss Decomposes into Three Stages)**.** *For any bottleneck-factorized receiver,*

$$L_{\text{total}} = L_{\text{ext}} + L_{\text{abs}} + L_{\text{close}}. \tag{3}$$

Each term is nonnegative. The terms correspond respectively to externalization, absorption, and action closure.

The theorem identifies where post-message loss remains, not merely its magnitude; mutual-information definitions are in Appendix B.3. This motivates the replay experiment: holding the sender message fixed across receivers removes sender-side variation and isolates receiver-side losses.

Proposition 5 (Fixed-Message Replay Isolates Receiver-Side Loss)**.** *For two receivers evaluated on the same local input, broader context, and the same message generated by a fixed sender and held fixed during replay, the sender-side externalization term cancels. The paired difference, therefore, depends only on receiver-side absorption and action-closure terms.*

We instantiate $Z_t^\star = (Z_t^{\text{art}}, Z_t^{\text{act}})$: absorption appears as missing content, while action closure appears as a failure to convert recovered content into the correct action. On non-replay protocols, content-missing metrics conflate absorption with sender-side externalization. Proofs in Appendix B.3.

2.5 When Stronger Upstream Agents Help

A stronger upstream agent is not automatically better. It may produce a more informative message, but it can also introduce additional protocol tax through longer messages, less compatible surface forms, or an increased reasoning burden for the executor. For a fixed executor R, compare two upstream agents $s \preceq s'$. Let $\Delta A_{R,\ell}(s, s')$ denote the extra receiver-relative gain from using s' instead of s, and let $\Delta\tau_{R,\ell}(s, s')$ denote the extra protocol tax.

Definition 2.3 (Burden-Indexed Receiver Family)**.** *A receiver family $\{R_b : b \in \mathcal{B}\}$ is* burden-indexed *if each receiver's local insufficiency is summarized by a scalar burden b, such as $H(Z_t^\star \mid X_t^{R_b})$ or an empirical receiver-only proxy.*

Theorem 6 (Sender-Upgrade Burden Threshold under Single Crossing)**.** *For any step loss ℓ and fixed executor R,*

$$U_{s',R,\ell} - U_{s,R,\ell} = \Delta A_{R,\ell}(s, s') - \Delta\tau_{R,\ell}(s, s').$$

For a burden-indexed family, let $\Psi_{s,s'}(b)$ be this net value at burden b. If $\Psi_{s,s'}$ is continuous and single-crossing, with negative value at low burden and positive value at high burden, then there is a unique threshold $b^\star$ at which the stronger upstream agent changes from harmful to helpful.

The theorem makes sender choice depend on the receiver's local burden (Figure 7). Section 4 estimates this threshold; the proof is in Appendix B.4.

3 Dynamic Coordination and Protocol Control

We turn the step-level theory of Section 2 into an inference-time control rule that scores discrete protocol actions by the same gain, tax, mismatch, sender-upgrade, and residual-risk quantities.

3.1 Residual Risk in Shared-State Workflows

A one-step communication error does not always have the same cost. In loosely coupled tasks, later observations may resynchronize the agents. In shared-state tasks, however, an incorrect intermediate artifact, a stale plan state, or a workspace inconsistency may persist and affect subsequent steps. We represent this with a scalar $D_t \geq 0$: the unresolved task-relevant discrepancy in the shared state.

In our clipped residual model, let k_t be the fresh discrepancy at step t, q_t the resolved discrepancy, and λ_t the carry coefficient measuring how strongly residual persists through the shared state. For a counterfactual trajectory with residual $\widetilde{D}_t$, the dynamics and residual gap are:

$$D_{t+1} = k_t + \lambda_t[D_t - q_t]_+, \qquad \Delta_t := \widetilde{D}_t - D_t. \tag{4}$$

The scalar notation is for readability; vector residuals follow componentwise. A residual gap multiplies forward while both trajectories remain in the carry regime; later observations or repairs can shrink the gap, and the gap vanishes only when the perturbed residual is fully resolved. The following proposition compares trajectories with the same discrepancy, repair, and carry sequence.

Proposition 7 (Conditional Propagation)**.** *Suppose two trajectories share the same (k_t, q_t, λ_t) sequence and differ only through an initial residual gap Δ_t. If $D_u > q_u$ and $\widetilde{D}_u > q_u$ for all $u = t, \ldots, t+h-1$, then*

$$\Delta_{t+h} = \Big(\prod_{u=t}^{t+h-1} \lambda_u\Big)\Delta_t.$$

Proposition 7 is the dynamic complement to the one-step theory. Weak coupling makes local mistakes wash out; persistent coupling turns them into future risk. See Appendix B.5 for the detailed proof.

3.2 Analysis-Guided Protocol Control

The resulting protocol selector chooses a communication action at each step. The action set is discrete, with budgets and model choices treated as action parameters:

$$\mathcal{A}_t \subseteq \{\texttt{skip}, \texttt{compress}(B), \texttt{raw}, \texttt{escalate}(C'), \texttt{refresh}\}.$$

The actions correspond to the theory as follows. `skip` uses no message and implements the RO endpoint. `compress`(B) sends a bounded message under budget B and is evaluated by the compression criterion. `raw` exposes full context and remains necessary because Bayes-sufficient compression can still fail for a bounded executor. `escalate`(C') directly uses a stronger receiver on the compressed message, scored by the corresponding gain–tax comparison. `refresh` triggers shared-state resynchronization and is scored by residual propagation.

For an action α at step t, let Φ_α be the deployed receiver class, ρ_α the induced protocol, and $M_t(\alpha)$ the message. Let $c(\alpha)$ be the immediate cost, and $r_t(\alpha)$ the estimated finite-horizon residual risk under the clipped propagation model of Proposition 7. The selector scores each candidate by receiver-relative utility minus immediate and future costs, with $\eta_c, \eta_r \geq 0$ tuned on a held-out split:

$$J_t(\alpha) = A^\star_{\Phi_\alpha,\ell}(M_t(\alpha); Z^\star_t) - \tau_{\rho_\alpha,\ell}(M_t(\alpha)) - \eta_c c(\alpha) - \eta_r r_t(\alpha), \qquad \alpha^\star_t \in \operatorname*{argmax}_{\alpha\in\mathcal{A}_t} J_t(\alpha). \tag{5}$$

The gain term measures target recovery, tax measures protocol burden, and the remaining terms charge immediate cost and future residual risk. Eq. (5) guides the deployed score $\widehat{J}_t$ through the following procedure.

Offline calibration. We split records by `instance_id`, keeping the receiver interface and action set fixed. Paired RO/SF/MA calibration records supply an action-level offset $\widehat{\tau}$ that captures protocol burden together with gain effects. Held-out RO records also supply a local-sufficiency rate for an early `skip` branch. The offsets, skip threshold, and cost weights are fixed before evaluation; the selector reads instance-level signals at test time.

Scoring and execution. The static ContextBench study ranks stored `skip`, `raw`, and `compress` outcomes. The online ToolSandbox gate selects `skip` before sender calls. Otherwise, the five-action path (`refresh/skip/raw/compress/refresh`) constructs and caches real `compress` and `refresh` messages before nine receiver probes, capped at four tokens. `raw` reads full upstream context; `escalate` reuses the compressed message with a stronger receiver. The gain surrogate $\hat{A}$ is the clipped leading-token log-probability difference. The selected action executes once using its cached candidate. Appendix C.6 specifies these paths and their call counts.

The static one-step ablation uses $\eta_c = 10^{-4}$ on raw token counts and $\eta_r = 0$. Relative costs in tables are normalized for display. For shared-state workflows, logged discrepancy features inform the

Table 1: **Protocol performance by receiver.** Regime labels describe the Qwen condition. hit_joint for RO/SF/MA. In the Qwen rows, MA uses Qwen3-Next-80B-A3B as sender; in the OpenAI extension, the receiver is shown at left, and MA uses `gpt-5.4` as sender. † denotes the matched-sender Qwen3.5-9B condition. Released supports for the displayed Qwen rows are 480 steps for ContextBench, 72 for τ-bench and ToolSandbox, and 100 for BFCL. OpenAI cell counts (60–300) are in Appendix C.5.

	ContextBench *helpful*			**ToolSandbox** *noise*			**τ-bench** *boundary*			**BFCL** *noise*			**Derived summaries**		
Receiver / Setup	RO	SF	MA	RO	SF	MA	RO	SF	MA	RO	SF	MA	$\overline{\Delta}_{\text{comm}}$	$\overline{\Delta}_{\text{cmp}}$	$\Delta^{\text{Ctx}}_{\text{comm}}$
Qwen bounded protocol (Qwen3.5-9B receiver, Qwen3-Next-80B-A3B sender)															
Qwen3.5-9B	.633	.294	**.775**	.889	.903	.653	.139	**.167**	.139	.980	**1.00**	.750	−.08	−.01	+.14
Qwen matched†	*.633*	*.294*	.565	*.889*	*.903*	.917	*.139*	*.167*	.167	*.980*	*1.00*	.910	−.02	+.05	−.07
OpenAI model-family extension (MA sender = `gpt-5.4`; receiver shown at left)															
`gpt-4o-mini`	.650	.743	**.787**	.803	.957	**1.00**	.253	.020	**.373**	.878	.722	.833	+.10	+.14	+.14
`gpt-5.4-nano`	.493	.637	**.783**	.757	.927	**1.00**	.313	.233	.300	**.883**	.811	.839	+.12	+.08	+.29
`gpt-5.4-mini`	.613	.770	**.787**	.697	.890	**1.00**	.293	.233	**.327**	.839	.833	**.844**	+.13	+.06	+.17
`gpt-5.4`	.600	.777	**.853**	.653	1.00	**.993**	.240	.180	.147	.861	.833	**1.00**	+.16	+.05	+.25
`o3`	.223	.133	**.493**	.740	.980	**1.00**	.200	.173	.160	**1.00**	.983	**1.00**	+.12	+.10	+.27
`o4-mini`	.500	.617	**.743**	.677	.733	**1.00**	.273	**.280**	.160	.844	.889	**1.00**	+.15	+.10	+.24

residual score; `refresh` regenerates the shared-state summary. The perturbation and repair studies evaluate this mechanism separately. Appendix B.6 gives the value-regret bound under bounded score-estimation error.

4 Experiments

We tested representation sensitivity (Theorem 3), receiver-side loss (Theorem 4), sender upgrades (Theorem 6), and the selector score (Eq. (5)). Appendix C provides the protocols, paired uncertainty analyses, and supplementary results.

4.1 Experimental Setup

We evaluated six benchmark families (BFCL (Patil et al., 2024), ToolSandbox (Lu et al., 2024), ContextBench (Li et al., 2026), RepoBench (Liu et al., 2024), SWE-bench Lite (Jimenez et al., 2024), and τ-bench (Yao et al., 2025)) covering high-local-sufficiency, context-heavy, capability-transfer, and boundary regimes; LongCLI-Bench provides a shared-state stress test (Appendix C.7). The default sender is Qwen3-Next-80B-A3B, the receiver Qwen3.5-9B, the message budget $B{=}128$ tokens (SWE-Lite uses Qwen3-Coder-Next-FP8 as sender), and matched-sender ($\mathrm{MA}_{\mathrm{match}}$, 9B→9B), stronger-sender ($\mathrm{MA}_{\mathrm{main}}$), and fixed-message replay contrasts are paired at the (instance, step) level. For scoring, we map each benchmark's native supervision into a set of acceptable artifact/action targets $G_t = \{(Z^{\mathrm{art}}_{t,k}, Z^{\mathrm{act}}_{t,k})\}_{k=1}^{K_t}$; these labels are never exposed to the sender, receiver, or selector. Appendix C.1 gives mapping rules and traces. One receiver instruction template is shared across the no-message and with-message paths; the user-turn context varies by protocol. Primary accuracy is `hit_joint` (h_j), computed directly from the paired target. For the content/closure diagnostic, we report an artifact accuracy h_a, conditional action-conversion h_{action}, content loss L_{content}, and closure loss L_{closure}; when all quantities are measured on the same step support, these correspond to the empirical identity $1{-}h_j = \mathbb{E}[1{-}h_a]{+}\mathbb{E}[h_a(1{-}h_{\mathrm{action},t})]$. Replay tables with non-uniform coverage report h_j directly and treat the loss columns as diagnostic component means rather than reconstructing h_j from marginal products. The three theory-aligned contrasts are $\Delta_{\mathrm{comm}}{:=}\mathrm{Acc}(\mathrm{MA}_{\mathrm{main}}){-}\mathrm{Acc}(\mathrm{RO})$, $\Delta_{\mathrm{cmp}}{:=}\mathrm{Acc}(\mathrm{MA}_{\mathrm{main}}){-}\mathrm{Acc}(\mathrm{SF})$, and $\Delta_{\mathrm{strong}}{:=}\mathrm{Acc}(\mathrm{MA}_{\mathrm{main}}){-}\mathrm{Acc}(\mathrm{MA}_{\mathrm{match}})$, with gain surrogate $\hat{A}{:=}\mathbb{E}[\texttt{surprisal_no_msg} - \texttt{surprisal_bits}]$, whose two terms are hit-based proxy scores rather than measured log-probabilities (Appendix C.2), and token cost taken as mean tokens per evaluated step. Evaluation-set definitions and protocol-variant controls in Appendices C.1 and C.5.

Table 2: **Fixed-message replay on ContextBench.** Receivers process the same DeepSeek-V3.1 messages, isolating receiver-side effects. Component coverage is non-uniform across receivers; h_j is computed directly; the content and closure columns are diagnostic component means.

Receiver	Replay accuracy			Observable post-message error			Summary	
	$h_j \uparrow$	$h_a \uparrow$	$h_{\text{action}} \uparrow$	$L_{\text{content}} \downarrow$	$L_{\text{closure}} \downarrow$	frac. closure	$\eta \uparrow$	Regret$\downarrow$
DeepSeek-R1	**0.731**	0.732	0.989	**0.268**	0.008	3%	1.00	0.000
Qwen3.5-9B	0.728	0.728	0.989	**0.272**	0.008	3%	1.00	0.003
Llama-70B	0.714	0.714	0.997	**0.286**	0.002	1%	1.00	0.017
DeepSeek-V3.1	0.707	0.730	0.977	**0.270**	0.017	6%	0.97	0.024
Qwen2.5-7B	0.702	0.702	0.975	**0.298**	0.018	6%	1.00	0.029
Qwen3.5-397B-A17B	0.685	0.717	0.926	**0.283**	0.053	16%	0.95	0.046
GLM-5	0.382	0.741	0.545	0.259	**0.337**	**57%**	0.52	**0.349**
MiniMax-M2.5	0.036	0.613	0.053	0.387	**0.581**	**60%**	0.06	**0.695**

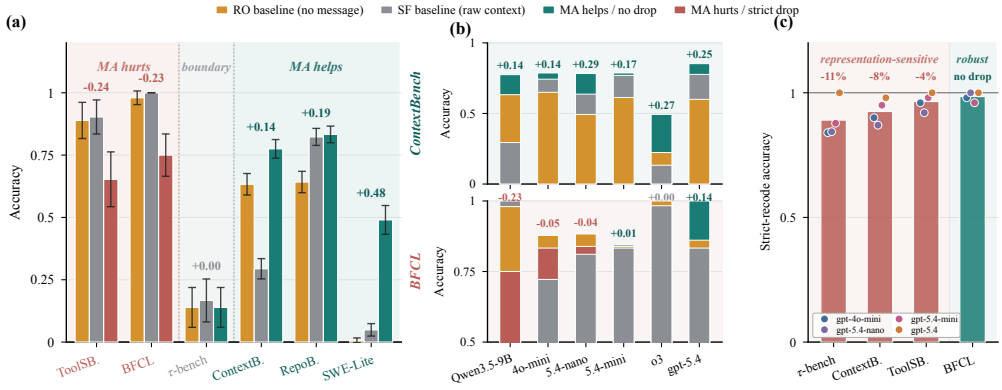

Figure 3: **Helpful, noisy, and boundary regimes for compression.** (a) Six benchmarks under the Qwen3.5-9B protocol, grouped by communication effect. (b) ContextBench (top) and BFCL (bottom) protocol comparison for Qwen and OpenAI receivers. (c) Controlled representation-sensitivity probe motivated by Theorem 3: direct vs. strict-recoded forms across four benchmarks.

4.2 Benchmark-Wide Numerical Summary

Table 1 compares RO, SF, and MA; Table 2 fixes the sender message. Figures 3–6 connect these contrasts to protocol choices. Qwen comparisons are paired within each evaluation set. OpenAI cells use the counts in Appendix C.5, including unequal protocol coverage. The derived columns summarize displayed rates: $\overline{\Delta}_{\text{comm}}$ and $\overline{\Delta}_{\text{cmp}}$ average the four benchmarks, and $\Delta^{\text{Ctx}}_{\text{comm}}$ gives the ContextBench contrast.

Instance-clustered bootstrap analyses support the helpful and harmful regimes on separately stated paired supports (Appendix C.1). ContextBench MA exceeds RO by 0.1711 (95% CI $[0.1417, 0.2012]$; 2,390 steps, 799 instances), and RepoBench by 0.1611 ($[0.1340, 0.1891]$; 2,396 steps, 800 instances). The τ-bench interval includes zero. These paired estimates complement the main table and retain their own sample counts.

4.3 When Compression Beats Raw Context

Communication value depends on the receiver's local view and interface. In Figure 3(a), the default Qwen protocol hurts BFCL and ToolSandbox, helps ContextBench ($0.633 \rightarrow 0.775$) and RepoBench ($0.642 \rightarrow 0.833$), and has little effect on τ-bench. ContextBench also improves across the OpenAI receivers in panel (b), including `gpt-5.4` ($0.60 \rightarrow 0.85$). ToolSandbox behaves differently across families: `gpt-4o-mini` improves from 0.803 to 1.000, whereas the default Qwen receiver drops from 0.889 to 0.653 (Table 1). Thus the regime belongs to the receiver and protocol condition as well as the task.

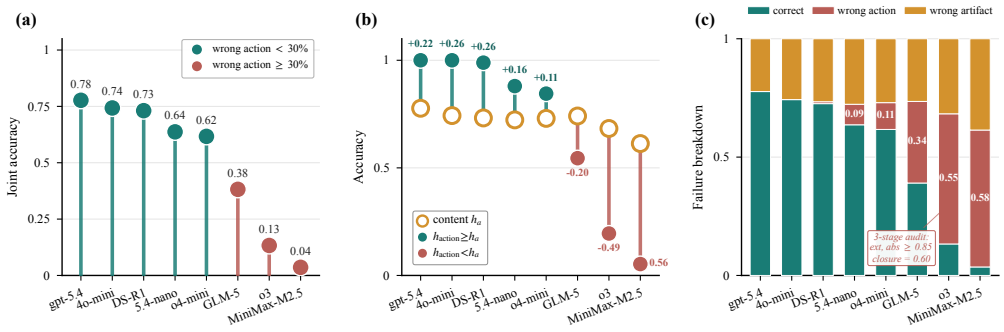

Figure 4: **Receiver-side content and closure bottlenecks on ContextBench.** (a) Joint accuracy by receiver, colored by wrong-action share. (b) Content h_a (open ring) and action h_{action} (filled, direction-colored) per receiver, with $h_{\text{action}} - h_a$ annotated. (c) Observable outcome decomposition into correct/wrong action/wrong artifact; inline three-stage diagnostic motivated by Theorem 4.

ContextBench gains persist when message truncation is controlled. On the 480-step audit, MA exceeds RO by 10.64 percentage points among 404 capped messages and by 32.89 points among 76 uncapped messages. The respective paired 95% intervals are $[5.7, 15.8]$ and $[22.4, 43.4]$ points. All capped messages retain both decision fields. Appendix C.2 reports the split, a paired budget increase, and gain-surrogate diagnostics.

Surface form matters even with the target held fixed. The strict-recoding probe in Figure 3(c) lowers accuracy by 13.5 points on τ-bench, 6.0 on ContextBench, and 3.5 on ToolSandbox; BFCL is unchanged. This controlled comparison links Theorem 3 to the deployed interface and motivates retaining `raw` in Section 4.6.

4.4 Closure-Dominated Receiver-Side Loss

Correct artifact recovery can coexist with frequent action errors. Figure 4 tests the output events motivated by Theorem 4. DeepSeek-R1, `gpt-5.4`, and `gpt-4o-mini` have $h_{\text{action}} \geq 0.98$ and closure fractions at or below 3%, so joint accuracy tracks artifact accuracy. GLM-5, `o3`, and MiniMax-M2.5 retain $h_a \geq 0.61$ but have action-conversion rates of 0.545, 0.195, and 0.053. The separate `gpt-5.4`-sender audit in panel (c) places `o3`'s externalization and absorption success at ≥ 0.85, with conditional closure at 0.60.

Parsing audits locate action errors after valid output and correct artifact recovery. In a `gpt-5.4`-sender replay, wrong-action rates among such outputs are 44.5% for GLM-5, 94.0% for MiniMax-M2.5, and 40.8% for `o3`. Outputs explicitly choose `read` when the evaluated action is `edit`. On SWE-Lite, RO artifact accuracy spans 0.885–0.917 across three parse-valid seed slices; the reported zero-joint slice reflects action errors (Appendix C.1). Appendices C.3 and D provide parsing rates, evaluation supports, and traces.

4.5 The Stronger-Sender Threshold

Sender upgrades help on SWE-Lite, ContextBench, and RepoBench but hurt the other three Qwen conditions. Table 1 and Figure 5 show gains of $+0.52$, $+0.21$, and $+0.09$, compared with -0.03 on τ-bench, -0.26 on ToolSandbox, and -0.16 on BFCL. The negative τ-bench effect at low RO accuracy shows that local accuracy alone does not determine the regime; message fit and protocol burden also matter. Appendix C.4 extends the sender swaps to OpenAI receivers.

Expanding the paired study from 22 to 41 anchors preserves the crossover in RO accuracy: the estimate is 0.6289 with bootstrap 95% CI $[0.552, 0.706]$, compared with the original 0.638 and $[0.491, 0.780]$. Of 38 nonzero anchors, 34 (89.5%) follow a single positive-to-negative crossing as local sufficiency increases. A hierarchical analysis of 12,540 paired steps favors the single-crossing structure over linear and two-crossing alternatives in all 10 anchor-separated folds. Appendix C.4 reports the fits and leave-family-out analysis; Appendix C.5 gives the model-family controls.

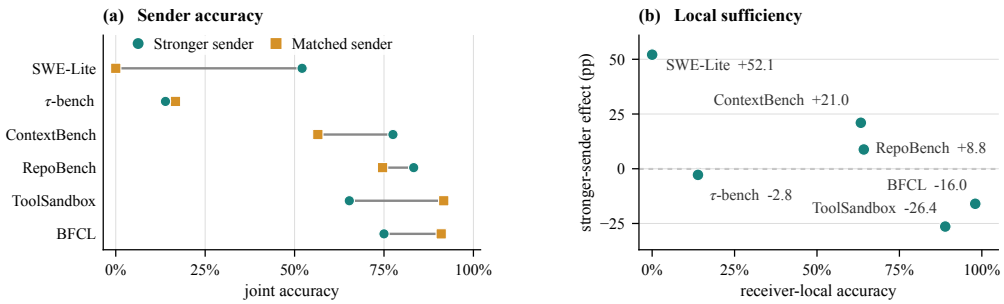

Figure 5: **Stronger-sender effects are regime-dependent.** Throughout, $\Delta_{\text{strong}} := \text{Acc}(\text{MA}_{\text{main}}) - \text{Acc}(\text{MA}_{\text{match}})$; positive means the stronger sender helps. (a) Stronger- and matched-sender accuracies for the six Qwen benchmark conditions. (b) Reported net effects against receiver-only accuracy. SWE-Lite uses seeds 2 and 1 for stronger and matched senders. RepoBench's 0.088 effect uses unrounded endpoints.

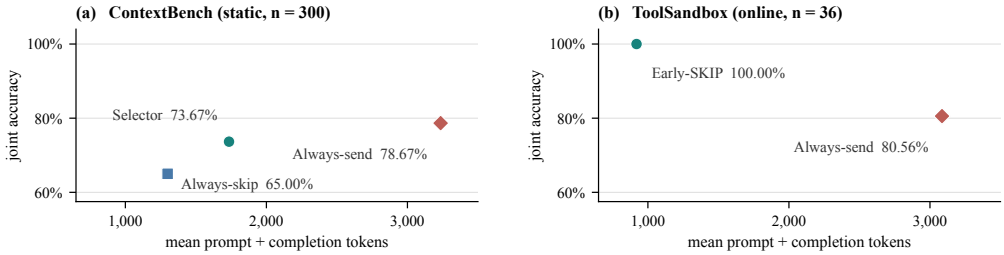

Figure 6: **Accuracy and recorded token cost under protocol selection.** (a) ContextBench static audit on 300 records: always-skip, selector, and always-send. (b) ToolSandbox online audit on 36 steps: early-skip and always-send. Both panels use absolute joint accuracy and mean prompt-plus-completion tokens; supports are separate.

4.6 Inference-Time Protocol Selection in Practice

The selector trades communication cost against accuracy according to the receiver's needs. On the 300-record ContextBench audit in Figure 6(a), it reaches 0.7367 accuracy at 1,735 tokens per step, between always-skip (0.6500, 1,300 tokens) and always-send (0.7867, 3,235.4 tokens). It recovers 8.67 accuracy points over always-skip while using 46.4% fewer tokens than always-send. This static comparison ranks stored outcomes for the same three actions.

On the online 36-step ToolSandbox slice, the early-skip path reached 1.000 accuracy with 919.4 prompt-plus-completion tokens and 6.60 seconds per step, versus 0.8056, 3,085.4 tokens, and 9.56 seconds for always-send. On ContextBench, instance-disjoint calibration splits gave accuracy 0.7368–0.7458 across calibration fractions of 5–40%. The learned-router comparison reached utility 0.5692, versus 0.5191 for gradient boosting under the same cost weight. The static τ-bench result retains a useful failure case: rewriting reached 0.139 accuracy versus the selector's 0.083 at similar input-token cost. Appendices C.6 and C.7 give the baselines, ablations, call accounting, and residual studies.

5 Conclusion

Receiver-relative communication trades information loss and decoder compatibility against protocol tax and local burden. This step-level view explains why one protocol can help or hurt: compression pays off only when tax savings exceed omitted information and decoder mismatch, Bayes sufficiency does not guarantee receiver usability, post-message loss splits into externalization, absorption, and action closure, and a stronger sender helps only above a receiver-burden threshold. On six benchmarks, one fixed Qwen protocol lifts ContextBench joint accuracy from 0.633 to 0.775 while reducing ToolSandbox accuracy from 0.889 to 0.653, and fixed-message replay exposes receivers that recover the correct artifact yet choose the wrong action. The resulting selector adapts communication to receiver needs, and Appendix E discusses trajectory-level success and calibration across interfaces.

References

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. Llm-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8053–8072, 2025. doi: 10.18653/v1/2025.findings-naacl.448.

Anthropic. Effective harnesses for long-running agents. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents, 2025a. Anthropic Engineering, Nov. 26, 2025.

Anthropic. How we built our multi-agent research system. https://www.anthropic.com/engineering/multi-agent-research-system, 2025b. Anthropic Engineering, Jun. 13, 2025.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. *arXiv preprint arXiv:2308.10848*, 2023. doi: 10.48550/arXiv.2308.10848. URL https://arxiv.org/abs/2308.10848.

Yingyun Cui, Yi Xie, Piaohong Wang, Jiawei Ma, Bo Liu, and Liangliang Cao. PERFOPT-Bench: Evaluating coding agents on software performance optimization. *arXiv preprint arXiv:2607.07744*, 2026. URL https://arxiv.org/abs/2607.07744.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*, 2025.

Hongyi Du, Jiaqi Su, Jisen Li, Lijie Ding, Yingxuan Yang, Peixuan Han, Xiangru Tang, Kunlun Zhu, and Jiaxuan You. Which llm multi-agent protocol to choose? *arXiv preprint arXiv:2510.17149*, 2025.

Yijia Fan, Jusheng Zhang, Kaitong Cai, Jing Yang, Chengpei Tang, Jian Wang, and Keze Wang. Cost-Effective Communication: An auction-based method for language agent interaction. *arXiv preprint arXiv:2511.13193*, 2025. doi: 10.48550/arXiv.2511.13193.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. doi: 10.48550/arXiv.2308.00352. URL https://arxiv.org/abs/2308.00352.

Yoichi Ishibashi and Yoshimasa Nishimura. Self-organized agents: A llm multi-agent framework toward ultra large-scale code generation and optimization. *arXiv preprint arXiv:2404.02183*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Mark Malhotra, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. CAMEL: Communicative agents for "Mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023. doi: 10.48550/arXiv.2303.17760. URL https://arxiv.org/abs/2303.17760.

Han Li, Letian Zhu, Bohan Zhang, Rili Feng, Jiaming Wang, Yue Pan, Earl T. Barr, Federica Sarro, Zhaoyang Chu, and He Ye. Contextbench: A benchmark for context retrieval in coding agents. *arXiv preprint arXiv:2602.05892*, 2026. URL https://arxiv.org/abs/2602.05892.

Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pPjZIOuQuF.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024. URL https://arxiv.org/abs/2408.04682.

Zhenyu Mao, Jacky Keung, Fengji Zhang, Shuo Liu, Yifei Wang, and Jialong Li. Towards engineering multi-agent llms: A protocol-driven approach. *arXiv preprint arXiv:2510.12120*, 2025.

Boye Niu, Yiliao Song, Kai Lian, Yifan Shen, Yu Yao, Kun Zhang, and Tongliang Liu. Flow: Modularized agentic workflow automation. *arXiv preprint arXiv:2501.07834*, 2025.

OpenAI. A practical guide to building agents. https://openai.com/business/guides-and-resources/a-practical-guide-to-building-ai-agents/, 2025. OpenAI business guide.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=tBRNC6YemY.

Run Peng, Ziqiao Ma, Amy Pang, Sikai Li, Zhang Xi-Jia, Yingzhuo Yu, Cristian-Paul Bara, and Joyce Chai. Communication and verification in llm agents towards collaboration under information asymmetry. *arXiv preprint arXiv:2510.25595*, 2025.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

Humza Sami, Mubashir ul Islam, Samy Charas, Asav Gandhi, Pierre-Emmanuel Gaillardon, and Valerio Tenace. Nexus: A lightweight and scalable multi-agent framework for complex tasks automation. *arXiv preprint arXiv:2502.19091*, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Jingzhe Shi, Qinwei Ma, Hongyi Liu, Hang Zhao, Jenq-Neng Hwang, and Lei Li. Intrinsic entropy of context length scaling in llms. In *The Fourteenth International Conference on Learning Representations*, 2026.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik R. Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023. doi: 10.48550/arXiv.2303.11366. URL https://arxiv.org/abs/2303.11366.

Vaibhav Srivastav and Katia Gil Guzman. Openai for developers in 2025. https://developers.openai.com/blog/openai-for-developers-2025, 2025. OpenAI Developer Blog, Dec. 30, 2025.

Vali Tawosi, Keshav Ramani, Salwa Alamir, and Xiaomo Liu. Almas: an autonomous llm-based multi-agent software engineering framework. *arXiv preprint arXiv:2510.03463*, 2025.

Zhao Wang, Sota Moriyama, Wei-Yao Wang, Briti Gangopadhyay, and Shingo Takamatsu. Talk structurally, act hierarchically: A collaborative framework for llm multi-agent systems. *arXiv preprint arXiv:2502.11098*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023. doi: 10.48550/arXiv.2308.08155. URL https://arxiv.org/abs/2308.08155.

Xi Xiao, Yunbei Zhang, Xingjian Li, Tianyang Wang, Xiao Wang, Yuxiang Wei, Jihun Hamm, and Min Xu. Visual instance-aware prompt tuning. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 2880–2889, 2025.

Xi Xiao, Xingjian Li, Cheng Han, Tianyang Wang, Lin Zhao, Yunbei Zhang, Guosheng Hu, Runmin Jiang, Xi Li, Xiao Wang, and Min Xu. Adapting vision foundation models with cascaded semantics. *Transactions on Machine Learning Research*, 2026a. ISSN 2835-8856. URL https://openreview.net/forum?id=SSsobNZJPO.

Xi Xiao, Xingjian Li, Yunbei Zhang, Cheng Han, Tianming Liu, Tianyang Wang, Runmin Jiang, Jihun Hamm, Xiao Wang, and Min Xu. Layer-specific prompt fusion discovery via differentiable search in vision foundation models. In *European Conference on Computer Vision*, pages 391–409. Springer, 2026b.

Xi Xiao, Chen Liu, Chih-Ting Liao, Yunbei Zhang, Qizhen Lan, Yuxiang Wei, Lin Zhao, Janet Wang, Jianyang Gu, Muchao Ye, et al. Staying vigil ant: Mitigating visual laziness via counterfactual visual alignment in mllms. In *European Conference on Computer Vision*, pages 287–305. Springer, 2026c.

Xi Xiao, Chenrui Ma, Yunbei Zhang, Chen Liu, Zhuxuanzi Wang, Yanshu Li, Lin Zhao, Guosheng Hu, Tianyang Wang, and Hao Xu. Not all directions matter: Towards structured and task-aware low-rank model adaptation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2132–2154, 2026d.

Yi Xie, Ziqing Zhou, Chun Ouyang, Siao Liu, Linqiang Hu, and Zhongxue Gan. ACORN: Acyclic coordination with reachability network to reduce communication redundancy in multi-agent systems. In *Proceedings of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 2190–2198, 2025.

Yi Xie, Siao Liu, Falong Fan, Yuanqi Yao, Yue Zhao, and Bo Liu. TeamTR: Trust-region fine-tuning for multi-agent LLM coordination. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026a.

Yi Xie, Yangyang Xu, Yi Fan, and Bo Liu. SAT: Sequential agent tuning for coordinator free plug and play multi-LLM training with monotonic improvement guarantees. In *Proceedings of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2026b.

Yi Xie, Zhanke Zhou, Chentao Cao, Bo Liu, and Bo Han. DICE: Entropy-regularized equilibrium selection for stable multi-agent LLM coordination. *arXiv preprint arXiv:2606.08068*, 2026c. URL https://arxiv.org/abs/2606.08068.

Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. Beyond self-talk: A communication-centric survey of llm-based multi-agent systems. *arXiv preprint arXiv:2502.14321*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R. Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024. doi: 10.48550/arXiv.2405.15793. URL https://arxiv.org/abs/2405.15793.

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. A survey of ai agent protocols. *arXiv preprint arXiv:2504.16736*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Xie Yi, Zhanke Zhou, Chentao Cao, Qiyu Niu, Tongliang Liu, and Bo Han. From debate to equilibrium: Belief-driven multi-agent LLM reasoning via Bayesian Nash equilibrium. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Yanwei Yue, Guibin Zhang, Boyang Liu, Guancheng Wan, Kun Wang, Dawei Cheng, and Yiyan Qi. MasRouter: Learning to route LLMs for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.acl-long.757. URL https://aclanthology.org/2025.acl-long.757/.

Ke Zhang, Patricio Gallardo Candela, Sudhir Murthy, Yi Xie, Zhi Wang, and Maziar Raissi. Beyond compilation: Evaluating faithful natural-language-to-Lean statement formalization. *arXiv preprint arXiv:2606.31002*, 2026. URL https://arxiv.org/abs/2606.31002.

Jialong Zhou, Lichao Wang, and Xiao Yang. GUARDIAN: Safeguarding LLM multi-agent collaborations with temporal graph modeling. In *Advances in Neural Information Processing Systems*, 2025a. doi: 10.48550/arXiv.2505.19234. URL https://openreview.net/forum?id=6j9xJ9pBjm.

Zhanke Zhou, Chentao Cao, Xiao Feng, Xuan Li, Zongze Li, Xiangyu Lu, Jiangchao Yao, Weikai Huang, Tian Cheng, Jianghangfan Zhang, Tangyu Jiang, Linrui Xu, Yiming Zheng, Brando Miranda, Tongliang Liu, Sanmi Koyejo, Masashi Sugiyama, and Bo Han. AlphaApollo: A system for deep agentic reasoning. *arXiv preprint arXiv:2510.06261*, 2025b.

Zhanke Zhou, Zongze Li, Weikai Huang, Xuan Li, Chentao Cao, Xiao Feng, Xiangyu Lu, Jinbo Hu, Menghan Lu, Yi Xie, Nico Pelleriti, Shiyang Liu, Max Zimmer, Brando Miranda, Jiangchao Yao, Bo Liu, Sanmi Koyejo, Sebastian Pokutta, and Bo Han. Reasoning is more than the model: Harness-aware evaluation of agents on verifiable reasoning tasks. In *3rd AI for Math Workshop: Toward Self-Evolving Scientific Agents*, 2026a.

Zhanke Zhou, Xiangyu Lu, Chentao Cao, Brando Miranda, Tongliang Liu, Bo Han, and Sanmi Koyejo. The easy, the hard, and the learnable: Confidence and difficulty-adaptive policy optimization for LLM reasoning. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026b.

Zhanke Zhou, Zhaocheng Zhu, Xuan Li, Mikhail Galkin, Xiao Feng, Sanmi Koyejo, Jian Tang, and Bo Han. Landscape of thoughts: Visualizing the reasoning process of large language models. In *The Fourteenth International Conference on Learning Representations*, 2026c.

Kunlun Zhu, Hongyi Du, Zhaochen Hong, Xiaocheng Yang, Shuyi Guo, Zhe Wang, Zhenhailong Wang, Cheng Qian, Xiangru Tang, Heng Ji, and Jiaxuan You. Multiagentbench: Evaluating the collaboration and competition of llm agents. *arXiv preprint arXiv:2503.01935*, 2025a.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025b. doi: 10.48550/arXiv.2509.25370.

Reproducibility statement

The theoretical assumptions and full proofs are in Appendix B. Appendix C specifies benchmark construction, evaluation protocols, and supporting results; Appendix C.6 gives selector settings, candidate construction, calibration splits, and call accounting. Appendix D provides fixed-message traces.

AI use statement

During manuscript revision, we used OpenAI Codex to assist with language editing, LaTeX formatting, methodological consistency checks, inspection of existing code and result records, analysis and presentation of existing results, and clarification of implementation parameters. Implementation settings were incorporated following author confirmation. This revision used existing experimental records and did not include new experimental runs. The authors are responsible for the final content of the paper.

Appendix Contents

A Extended Related Work **15**

B Additional Theory and Proofs **17**

B.1 One-Step Utility and Representation-Aware Compression 17

B.2 Bayes-Sufficient Compression and Receiver Compatibility 19

B.3 Bottleneck Decomposition and Fixed-Message Replay 21

B.4 Sender-Gap Identity and Burden Threshold 22

B.5 State Propagation 23

B.6 Protocol-Control Regret Bound 23

B.7 Idealized Specializations 24

C Additional Experimental Details and Results **24**

C.1 Benchmark and Protocol Summary 25

C.2 Extended Compression Results 30

C.3 Extended Closure Results 34

C.4 Extended Stronger-Sender Results 34

C.5 Cross-Family Model Ablations 36

C.6 Protocol-Level Tax Knobs 38

C.7 Conditional Propagation in Shared-State Tasks 42

D Illustrative Case Studies **44**

E Scope and Limitations **45**

The appendix is organized in the order body sections reference it: Appendix A expands the related-work discussion previewed in Section 1; Appendix B provides full proofs for Sections 2 and 3; Appendix C provides experimental details and supporting results for Section 4; Appendix D collects trajectory-level case studies; Appendix E expands scope and limitations from Section 5.

A Extended Related Work

Harness engineering and multi-agent workflow design. A large body of recent work treats LLM-based multi-agent systems as an engineering problem of workflow decomposition, role allocation, and protocol design. Early systems introduced role-playing and conversational agent societies (Li et al., 2023), programmable multi-agent conversations (Wu et al., 2023), dynamic agent-group orchestration (Chen et al., 2023), standardized operating procedures for collaborative agents (Hong et al., 2023), and chat-based software-development teams (Qian et al., 2024). More recent software and agentic workflow settings instantiate explicit multi-agent or sub-agent structures to scale context handling, task decomposition, and execution, including self-organized coding agents for large codebases (Ishibashi and Nishimura, 2024), modularized workflow optimization (Niu et al., 2025), lightweight multi-supervisor orchestration (Sami et al., 2025), structured communication with hierarchical refinement (Wang et al., 2025), agile-role software engineering agents (Tawosi et al., 2025), protocol-driven multi-agent engineering (Mao et al., 2025), and agentic reasoning systems that orchestrate models with tools and refine solutions over multiple rounds (Zhou et al., 2025b). These works primarily propose better harnesses or workflow templates. Harness-aware evaluation makes the complementary point on the measurement side: reported reasoning performance depends on the harness around the

model as well as on the model itself (Zhou et al., 2026a). Our paper instead fixes the general sender–receiver setting and asks a different question: what information should move between components when the receiver is bounded?

Training and equilibrium approaches to multi-agent coordination. A complementary line of work improves coordination by changing the agents or their interaction structure. Equilibrium-based methods cast multi-agent LLM reasoning as a Bayesian Nash equilibrium (Yi et al., 2025) or select stable coordination conventions through an entropy-regularized quantal response equilibrium (Xie et al., 2026c). Trust-region and sequential fine-tuning methods train LLM teams with per-update improvement guarantees and plug-and-play replacement of individual agents (Xie et al., 2026a,b), and in multi-agent reinforcement learning, acyclic reachability-based coordination reduces redundant communication between agents (Xie et al., 2025). These methods train or restructure the team, whereas our analysis holds the deployed sender and receiver fixed and asks when a single message improves the receiver's next decision.

Tool use, feedback, and agent-computer interfaces. Single-agent language-agent work also motivates our focus on next-step artifact/action decisions (Xiao et al., 2025, 2026d,c). ReAct interleaves reasoning and environment actions (Yao et al., 2023), Toolformer studies learned API-use decisions (Schick et al., 2023), Reflexion uses verbal feedback and memory to improve later actions (Shinn et al., 2023), and SWE-agent shows that the interaction interface itself can materially change software-agent performance (Yang et al., 2024). These systems emphasize that action spaces, tool schemas, feedback channels, and interface design shape what a model can turn into an action. Our receiver-relative view treats these concerns as part of the executor's bounded decoder and protocol tax rather than as fixed properties of a benchmark. Work on single-model reasoning visualizes reasoning trajectories by their distances to candidate answers (Zhou et al., 2026c) and reweights policy optimization by rollout confidence and question difficulty (Zhou et al., 2026b); these methods study how one model reaches an answer, whereas our step-level diagnostics locate where a message fails between two models.

Communication-centric surveys, protocol taxonomies, and failure analyses. A second line of work organizes the MA-LLM literature around communication itself. Beyond Self-Talk surveys LLM-based multi-agent systems from a communication-centric perspective (Yan et al., 2025), while A Survey of AI Agent Protocols analyzes the protocol layer and compares existing designs along dimensions such as scalability and latency (Yang et al., 2025). Most closely aligned with our motivation, MAST provides a fine-grained taxonomy of specification, misalignment, and verification failures across multiple MAS frameworks (Cemri et al., 2025). These papers clarify the design space and its failure modes but remain primarily descriptive or taxonomic. We recast these concerns as receiver-relative quantities for analysis, diagnosis, and protocol selection.

Coordination benchmarks and protocol evaluation. Recent evaluation work has also made multi-agent coordination and protocol choice first-class benchmark targets. MultiAgentBench evaluates collaboration and competition across diverse interactive scenarios (Zhu et al., 2025a); LLM-Coordination studies pure coordination settings involving environment comprehension, theory-of-mind reasoning, and joint planning (Agashe et al., 2025); and ProtocolBench argues that protocol choice should be evaluated not only by task success but also by latency, message overhead, and robustness (Du et al., 2025). Recent routers and resource-rational protocols, including MasRouter and auction-based communication, make related inference-time choices over collaboration modes, model assignments, or speaking rights (Yue et al., 2025; Fan et al., 2025). Agent evaluations in other domains likewise separate surface success from task-relevant correctness, scoring reproducible speedups rather than functional patches in software performance optimization (Cui et al., 2026) and semantic faithfulness rather than compilation in natural-language-to-Lean formalization (Zhang et al., 2026). Our use of benchmarks is narrower and identification-oriented: RO / SF / MA contrasts, same-message replay, and sender swap serve here as tools for isolating communication gain, receiver-side loss, and alignment effects rather than as a broad leaderboard target.

Communication under asymmetric information and protocol-sensitive collaboration. Communication and Verification in LLM Agents explicitly studies collaboration under information asymmetry and unequal verification capability (Peng et al., 2025). This is closely related to our concern that the same message may be recoverable in principle yet unusable in a deployed protocol. Our paper

differs in two ways. First, we separate externalization, absorption, and action closure as distinct loci of coordination loss. Second, we tie these distinctions to same-message replay and sender-swap diagnostics, which enable us to localize receiver-side versus sender-side failures under a common step-level objective.

Protocol choice as a control problem. Across harness engineering, protocol surveys, and protocol-evaluation work, one recurring practical theme is that communication choices trade off capability, latency, and message budget (Du et al., 2025; OpenAI, 2025; Anthropic, 2025a). Our closest connection to this theme is conceptual rather than architectural: we cast those choices as an inference-time control problem driven by receiver-relative gain, communication tax, and dynamic residual risk. The contribution is therefore not another routing architecture, but a common problem formulation that links modeling, diagnosis, and protocol selection for bounded receivers.

Error propagation and recovery. Agent failure studies and safety frameworks also model how local errors spread through multi-step or multi-agent interaction (Xiao et al., 2026b,a). AgentDebug studies cascading failures through a modular error taxonomy and targeted feedback for recovery (Zhu et al., 2025b), while GUARDIAN uses temporal graph modeling to detect and mitigate hallucination and error propagation in multi-agent collaborations (Zhou et al., 2025a). Our conditional propagation law is complementary: it is a small analytic model for when a local residual gap carries forward or resynchronizes, rather than a full debugging or safety framework.

B Additional Theory and Proofs

This appendix provides full proofs and auxiliary results for Sections 2 and 3; the subsections below are presented in the order in which their results are stated.

Table 3: **Notation roadmap.** Workflow primitives are shown in Figure 2; endpoint protocols are defined in Section 2.1. Several quantities listed here are defined in Sections 2 and 3.

$Z_t^\star$	action-sufficient target	$A^\star_{C,\ell}, \tau_{\rho,\ell}, U_{\rho,\ell}$	gain, tax, utility
$\Gamma_C, \mathcal{I}_{\mathrm{omit}}, \Lambda_C$	slack, omitted info, mismatch	$L_{\mathrm{ext}}/L_{\mathrm{abs}}/L_{\mathrm{close}}$	externalization / absorption / closure
$b, \Psi(b)$	burden, upgrade value	D_t, Δ_t, λ_t	residual, gap, carry
k_t, q_t	fresh / resolved residual	$\alpha, J_t(\alpha)$	protocol action, selector value
W_t, C_t, S_t^i, M_t	state, context, sender, message	X_t^j, Φ_C, A_t^j	local view, decoder, next action

B.1 One-Step Utility and Representation-Aware Compression

We first record a basic fact about the log-loss approximation slack.

Lemma B.1 (Slack is Nonnegative)**.** *For any conditioning variable Y and any receiver class Φ_C of predictive distributions under log-loss,*

$$\Gamma_C(Y) := \mathcal{R}^\star_{C,\log}(Z_t^\star \mid Y) - H(Z_t^\star \mid Y) \geq 0.$$

Proof. For any admissible predictor $q_\phi(\cdot \mid Y)$,

$$\begin{aligned}
\mathbb{E}[-\log q_\phi(Z_t^\star \mid Y)] &= \mathbb{E}[-\log p(Z_t^\star \mid Y)] + \mathbb{E}\left[\log \frac{p(Z_t^\star \mid Y)}{q_\phi(Z_t^\star \mid Y)}\right] \\
&= H(Z_t^\star \mid Y) + \mathbb{E}\big[\mathrm{KL}\big(p(\cdot \mid Y) \,\|\, q_\phi(\cdot \mid Y)\big)\big] \\
&\geq H(Z_t^\star \mid Y).
\end{aligned}$$

Taking the infimum over $\phi \in \Phi_C$ gives the claim. □

Proof of Proposition 1. By definition,

$$U_{\rho,\ell}(M_t; Z_t^\star) = R_\rho(0) - R_\rho(1).$$

Add and subtract $R^\star_C(0)$ and $R^\star_C(1)$:

$$\begin{aligned} U_{\rho,\ell}(M_t; Z_t^\star) &= R_\rho(0) - R_\rho(1) + R^\star_C(0) - R^\star_C(0) + R^\star_C(1) - R^\star_C(1) \\ &= \big(R^\star_C(0) - R^\star_C(1)\big) - \Big[\big(R_\rho(1) - R^\star_C(1)\big) - \big(R_\rho(0) - R^\star_C(0)\big)\Big]. \end{aligned}$$

The first bracket is $A^\star_{C,\ell}(M_t; Z_t^\star)$ and the second bracket is $\tau_{\rho,\ell}(M_t)$. □

Proof of Theorem 2. By Lemma B.1 and the definition of Γ_C,

$$\mathcal{R}^\star_{C,\log}(Z_t^\star \mid Y) = H(Z_t^\star \mid Y) + \Gamma_C(Y)$$

for every conditioning variable Y considered below.

Apply this identity to $Y = X_t^j$ and $Y = (X_t^j, M_t)$:

$$\begin{aligned} A^\star_{C,\log}(M_t; Z_t^\star) &= \mathcal{R}^\star_{C,\log}(Z_t^\star \mid X_t^j) - \mathcal{R}^\star_{C,\log}(Z_t^\star \mid X_t^j, M_t) \\ &= H(Z_t^\star \mid X_t^j) - H(Z_t^\star \mid X_t^j, M_t) + \Gamma_C(X_t^j) - \Gamma_C(X_t^j, M_t) \\ &= I(Z_t^\star; M_t \mid X_t^j) + \Gamma_C(X_t^j) - \Gamma_C(X_t^j, M_t). \end{aligned}$$

Similarly, conditioning on raw context gives

$$A^\star_{C,\log}(C_t; Z_t^\star) = I(Z_t^\star; C_t \mid X_t^j) + \Gamma_C(X_t^j) - \Gamma_C(X_t^j, C_t).$$

Subtracting the compressed-message gain from the raw-context gain yields

$$\begin{aligned} A^\star_{C,\log}(C_t; Z_t^\star) - A^\star_{C,\log}(M_t; Z_t^\star) = I(Z_t^\star; C_t \mid X_t^j) - I(Z_t^\star; M_t \mid X_t^j) \\ + \Gamma_C(X_t^j, M_t) - \Gamma_C(X_t^j, C_t). \end{aligned}$$

Because $M_t = f(C_t)$ is deterministic, the conditional chain rule gives

$$I(Z_t^\star; C_t \mid X_t^j) = I(Z_t^\star; M_t \mid X_t^j) + I(Z_t^\star; C_t \mid X_t^j, M_t).$$

Therefore

$$A^\star_{C,\log}(C_t; Z_t^\star) - A^\star_{C,\log}(M_t; Z_t^\star) = I(Z_t^\star; C_t \mid X_t^j, M_t) + \Lambda_C(M_t \to C_t),$$

where

$$\Lambda_C(M_t \to C_t) := \Gamma_C(X_t^j, M_t) - \Gamma_C(X_t^j, C_t).$$

Apply Proposition 1 to the compressed protocol and to the raw-context protocol:

$$\begin{aligned} U_{\rho_{\mathrm{cmp}},\log}(M_t; Z_t^\star) &= A^\star_{C,\log}(M_t; Z_t^\star) - \tau_{\rho_{\mathrm{cmp}},\log}(M_t), \\ U_{\rho_{\mathrm{raw}},\log}(C_t; Z_t^\star) &= A^\star_{C,\log}(C_t; Z_t^\star) - \tau_{\rho_{\mathrm{raw}},\log}(C_t). \end{aligned}$$

Subtracting the second identity from the first gives

$$\begin{aligned} \Delta U_{\mathrm{cmp-raw}} &= U_{\rho_{\mathrm{cmp}},\log}(M_t; Z_t^\star) - U_{\rho_{\mathrm{raw}},\log}(C_t; Z_t^\star) \\ &= \big(A^\star_{C,\log}(M_t; Z_t^\star) - A^\star_{C,\log}(C_t; Z_t^\star)\big) + \big(\tau_{\rho_{\mathrm{raw}},\log}(C_t) - \tau_{\rho_{\mathrm{cmp}},\log}(M_t)\big) \\ &= \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - I(Z_t^\star; C_t \mid X_t^j, M_t) - \Lambda_C(M_t \to C_t), \end{aligned}$$

which is Eq. (2). □

Receiver-class representation gap. The omitted information and decoder-mismatch terms can be combined into a single bounded-receiver certificate. Define

$$B_C(M_t, C_t) := \mathcal{R}^\star_{C,\log}(Z_t^\star \mid X_t^j, M_t) - \mathcal{R}^\star_{C,\log}(Z_t^\star \mid X_t^j, C_t).$$

This is the class-optimal log-loss gap between using the compressed message and using raw context.

Proposition 8 (Receiver-Class Compression Certificate)**.** *Under log-loss, if $M_t = f(C_t)$, then*

$$B_C(M_t, C_t) = I(Z_t^\star; C_t \mid X_t^j, M_t) + \Lambda_C(M_t \to C_t).$$

Consequently,

$$\Delta U_{\mathrm{cmp-raw}} = \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - B_C(M_t, C_t).$$

Thus, compressed communication dominates raw-context execution exactly when the tax saved by compression exceeds the bounded-receiver representation gap.

Proof. By the log-loss slack identity,

$$\begin{aligned} B_C(M_t, C_t) &= \Big[H(Z_t^\star \mid X_t^j, M_t) + \Gamma_C(X_t^j, M_t)\Big] \\ &\quad - \Big[H(Z_t^\star \mid X_t^j, C_t) + \Gamma_C(X_t^j, C_t)\Big] \\ &= H(Z_t^\star \mid X_t^j, M_t) - H(Z_t^\star \mid X_t^j, C_t) + \Lambda_C(M_t \to C_t). \end{aligned}$$

Because $M_t = f(C_t)$, the entropy difference equals

$$H(Z_t^\star \mid X_t^j, M_t) - H(Z_t^\star \mid X_t^j, C_t) = I(Z_t^\star; C_t \mid X_t^j, M_t).$$

This proves the first identity. The second identity follows by substituting this expression into Eq. (2). □

Corollary B.2 (Compatibility Condition for Compression Dominance)**.** *If $B_C(M_t, C_t) \le \kappa_t$, then compressed communication dominates raw-context execution whenever*

$$\Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} > \kappa_t.$$

Equivalently, if $\Lambda_C(M_t \to C_t) \le \bar{\kappa}_t$, then compression dominates whenever

$$\Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} > I(Z_t^\star; C_t \mid X_t^j, M_t) + \bar{\kappa}_t.$$

Proof. The first statement is immediate from Proposition 8. The second follows by substituting

$$B_C(M_t, C_t) = I(Z_t^\star; C_t \mid X_t^j, M_t) + \Lambda_C(M_t \to C_t).$$

□

B.2 Bayes-Sufficient Compression and Receiver Compatibility

This subsection proves Theorem 3. The result separates information sufficiency from receiver usability. A compressed message can preserve all target-relevant information for an unconstrained Bayes decoder while changing the representation geometry available to a bounded receiver class.

Operationally, a deployed LLM executor induces a bounded receiver class through an interface tuple

$$\mathcal{I} = (\text{model}, \text{tokenizer}, \text{prompt template}, \text{schema/parser}, \text{tool validator}, L, \text{decoding policy}).$$

A compression is receiver-compatible for this interface when the required artifact/action fields can be extracted and converted into a valid emitted action with small excess risk. Under this view, field names, action verbs, file paths, and tool schemas are part of the communication channel rather than cosmetic formatting. For example, a direct payload such as $\{\texttt{action_type} : \texttt{edit}, \texttt{target_artifact} : \texttt{path}\}$ may be compatible with an edit executor, while $\{\texttt{f1} : \texttt{A2}, \texttt{f2} : \texttt{path}\}$ can preserve the Bayes target but fail under the deployed parser or policy.

We use the following finite-alphabet proper-compression condition. A message $M = T(C)$ is an α-proper Bayes-sufficient compression of raw context C for target Z if

$$I(Z; C \mid M) = 0 \qquad \text{and} \qquad H(M) \le \alpha H(C), \tag{6}$$

for some $\alpha \in (0, 1)$. The first condition is Bayes sufficiency. The second excludes both raw context and same-size recodings.

The entropy constraint is an analysis-level abstraction of the finite token caps used in the experiments. If a token sequence over vocabulary $\mathcal{V}$ is hard-capped at length B, its support size is at most $\sum_{\ell=0}^{B} |\mathcal{V}|^\ell$, so $H(M) \le B \log |\mathcal{V}| + \log(B+1)$. More generally, realized token lengths upper-bound entropy up to length-coding constants. The appendix experiments therefore report realized token counts and truncation rates; they are operational budget controls, not direct entropy estimates.

Lemma B.3 (Alternating Labels Are Hard for Thresholds)**.** *Let M be uniform on $\{0, 1, \ldots, 2m-1\}$, and let the binary label alternate on the ordered message axis:*

$$Z(M) = \begin{cases} 0, & M \text{ even}, \\ 1, & M \text{ odd}. \end{cases}$$

For the threshold class

$$\Phi_{\mathrm{thr}} = \{h_\theta(M) = \mathbf{1}\{M \ge \theta\} : \theta \in \mathbb{R}\},$$

the optimal $0/1$ error is at least $(m-1)/(2m)$.

Proof. Any threshold is equivalent to choosing a cut index $k \in \{0, 1, \ldots, 2m\}$, predicting 0 on the first k ordered message values and 1 on the remaining $2m - k$ values. If $k = 2r$ is even, the first segment contains r positive labels and the second segment contains $m - r$ negative labels, so the threshold makes m mistakes. If $k = 2r + 1$ is odd, the first segment contains r positive labels and the second segment contains $m - r - 1$ negative labels, so it makes $m - 1$ mistakes. The boundary cases $k = 0$ and $k = 2m$ each make m mistakes. Therefore, each threshold incurs at least $m - 1$ mistakes among the $2m$ equiprobable states. □

Proof of Theorem 3. Fix any compression ratio $\alpha \in (0, 1)$ and any $\varepsilon > 0$. Choose $m \geq 2$ large enough that

$$\frac{m-1}{2m} \geq \frac{1}{2} - \varepsilon.$$

Let

$$Y \sim \mathrm{Unif}\{0, 1, \ldots, 2m-1\}, \qquad U \sim \mathrm{Unif}\{1, \ldots, K\},$$

independently, and define the raw context

$$C = (Y, U).$$

The nuisance coordinate U carries no target information but contributes to raw-context size. Let

$$Z := \mathbf{1}\{Y \geq m\}.$$

The bounded receiver class is representation-sensitive. It thresholds the visible scalar coordinate delivered by the protocol. In the raw context, the visible coordinate is the natural coordinate Y; on a compressed scalar message, the visible coordinate is the message value. Formally, define

$$\mathrm{coord}(Y, U) = Y, \qquad \mathrm{coord}(M) = M,$$

and let

$$\Phi_{\mathrm{thr}} = \{h_\theta(v) = \mathbf{1}\{\mathrm{coord}(v) \geq \theta\} : \theta \in \mathbb{R}\}.$$

Thus, the raw context is decoded by thresholding Y, whereas the compressed message is decoded by thresholding the delivered surface value M.

On raw context, the threshold $\theta^\star = m - \frac{1}{2}$ achieves zero error:

$$h_{\theta^\star}(Y, U) = \mathbf{1}\{Y \geq m\} = Z.$$

Therefore

$$\mathcal{R}^\star_{\Phi_{\mathrm{thr}}, 0/1}(Z \mid C) = 0.$$

Now define a recoding $f : \{0, \ldots, 2m-1\} \to \{0, \ldots, 2m-1\}$ by

$$f(i) = 2i, \qquad i = 0, \ldots, m-1,$$

and

$$f(m+i) = 2i + 1, \qquad i = 0, \ldots, m-1.$$

The compressed message drops the nuisance coordinate and sends

$$M := T(C) := f(Y).$$

Because f is a bijection on Y, the value of Y is a deterministic function of M. Since Z is a deterministic function of Y, we have

$$I(Z; C \mid M) = 0.$$

Thus M is Bayes-sufficient.

The message is also properly compressed. Since f is bijective on Y,

$$H(M) = H(Y) = \log(2m).$$

Because Y and U are independent,

$$H(C) = H(Y, U) = \log(2m) + \log K.$$

Choose

$$K \geq \left\lceil (2m)^{(1-\alpha)/\alpha} \right\rceil.$$

Then

$$H(M) = \log(2m) \leq \alpha\big(\log(2m) + \log K\big) = \alpha H(C).$$

Thus M is an α-proper Bayes-sufficient compression.

It remains to be shown that the error on the compressed representation is lower-bounded. Under the ordered message values $M = 0, 1, \ldots, 2m-1$, labels alternate:

$$Z = 0, 1, 0, 1, \ldots, 0, 1.$$

The negative states $Y = 0, \ldots, m-1$ map to even message values, while the positive states $Y = m, \ldots, 2m-1$ map to odd message values. By Lemma B.3,

$$\mathcal{R}^{\star}_{\Phi_{\mathrm{thr}},0/1}(Z \mid M) \geq \frac{m-1}{2m} \geq \frac{1}{2} - \varepsilon.$$

Thus, the raw context is decoded perfectly by the bounded receiver class, while an α-proper Bayes-sufficient compressed message has error arbitrarily close to balanced binary chance error. □

Receiver-compatible sufficiency. Theorem 3 shows that Bayes sufficiency alone does not imply usability for a bounded receiver. The missing condition is receiver compatibility.

Definition B.4 (Receiver-Compatible Sufficient Compression)**.** *A Bayes-sufficient compression $M_t = f(C_t)$ is κ-receiver-compatible for class Φ_C if*

$$B_C(M_t, C_t) = \mathcal{R}^{\star}_{C,\log}(Z_t^{\star} \mid X_t^j, M_t) - \mathcal{R}^{\star}_{C,\log}(Z_t^{\star} \mid X_t^j, C_t) \leq \kappa.$$

Corollary B.5 (Sufficient Compression Wins When Compatibility Beats Tax)**.** *If $M_t = f(C_t)$ is Bayes-sufficient for $Z_t^{\star}$ given X_t^j, then*

$$I(Z_t^{\star}; C_t \mid X_t^j, M_t) = 0$$

and

$$\Delta U_{\mathrm{cmp-raw}} = \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - \Lambda_C(M_t \to C_t).$$

More generally, if M_t is κ-receiver-compatible, compressed communication dominates raw context whenever

$$\Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} > \kappa.$$

Proof. Bayes sufficiency sets the omitted-information term in Eq. (2) to zero, leaving only the decoder-mismatch term. The general statement follows from Proposition 8: if $B_C(M_t, C_t) \leq \kappa$, then

$$\Delta U_{\mathrm{cmp-raw}} = \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - B_C(M_t, C_t) \geq \Delta\tau_{\mathrm{raw}\to\mathrm{cmp}} - \kappa.$$

Thus, compression beats raw context whenever the saved tax exceeds the receiver-class representation gap. □

Remark B.6 (Design implication)**.** *The useful target is not Bayes sufficiency alone, but receiver-compatible sufficiency: the message should preserve the action-sufficient target, reduce protocol tax, and present the target in a surface form usable by the bounded receiver.*

B.3 Bottleneck Decomposition and Fixed-Message Replay

We now prove the loss decomposition and replay result. Throughout this subsection, condition on X_t^j unless stated otherwise. The bottleneck-factorized receiver satisfies the conditional-independence chain

$$Z_t^{\star} \to C_t \to M_t \to \widehat{Z}_t \to A_t^j.$$

Proof of Theorem 4. Define the adjacent drops

$$L_{\mathrm{ext}} := I(Z_t^{\star}; C_t \mid X_t^j) - I(Z_t^{\star}; M_t \mid X_t^j),$$

$$L_{\mathrm{abs}} := I(Z_t^{\star}; M_t \mid X_t^j) - I(Z_t^{\star}; \widehat{Z}_t \mid X_t^j),$$

and

$$L_{\text{close}} := I(Z_t^\star; \widehat{Z}_t \mid X_t^j) - I(Z_t^\star; A_t^j \mid X_t^j).$$

By the data-processing inequality applied along

$$Z_t^\star \to C_t \to M_t \to \widehat{Z}_t \to A_t^j$$

conditional on X_t^j, each adjacent drop is nonnegative. Summing the adjacent drops gives the telescoping identity

$$I(Z_t^\star; C_t \mid X_t^j) - I(Z_t^\star; A_t^j \mid X_t^j) = L_{\text{ext}} + L_{\text{abs}} + L_{\text{close}}.$$

The left-hand side is L_{total}. □

Proof of Proposition 5. For a receiver r evaluated on the fixed triple (X_t, C_t, M_t), Theorem 4 gives

$$I(Z_t^\star; A_t^r \mid X_t) = I(Z_t^\star; M_t \mid X_t) - L_{\text{abs}}^r - L_{\text{close}}^r.$$

The term $I(Z_t^\star; M_t \mid X_t)$ is shared by all receivers in a fixed-message replay setting. Therefore, for two receivers r, r',

$$I(Z_t^\star; A_t^r \mid X_t) - I(Z_t^\star; A_t^{r'} \mid X_t) = \big(L_{\text{abs}}^{r'} + L_{\text{close}}^{r'}\big) - \big(L_{\text{abs}}^{r} + L_{\text{close}}^{r}\big).$$

Thus, sender-side externalization cancels in the paired comparison, leaving receiver-side absorption and action closure. □

Observable factorization. When $Z_t^\star = (Z_t^{\text{art}}, Z_t^{\text{act}})$, this motivates the observable $0/1$ factorization used in Section 4: a diagnostic, not an MI estimate.

B.4 Sender-Gap Identity and Burden Threshold

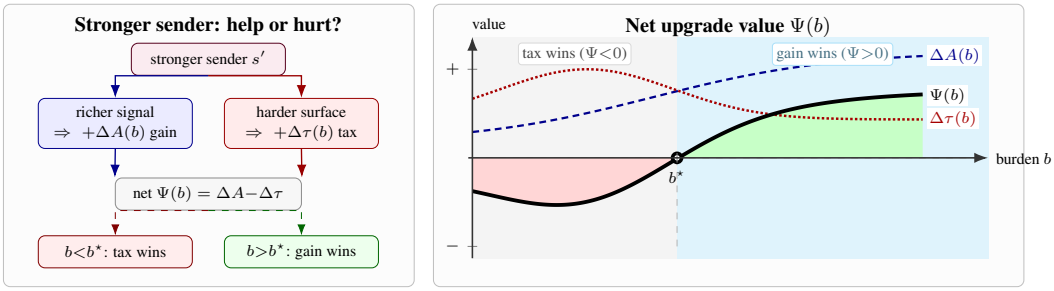

Figure 7: **Sender-upgrade threshold.** A stronger upstream agent s' raises both extra gain (richer signal, blue) and extra tax (harder surface, red); under single-crossing of $\Psi(b)=\Delta A(b)-\Delta\tau(b)$, the upgrade helps only above the burden threshold $b^\star$ (Theorem 6).

Proof of Theorem 6. For a fixed executor R, apply Proposition 1 to the sender s and to the stronger sender s':

$$U_{s,R,\ell} = A_{R,\ell}^\star(M_t^s; Z_t^\star) - \tau_{s,R,\ell}(M_t^s),$$

$$U_{s',R,\ell} = A_{R,\ell}^\star(M_t^{s'}; Z_t^\star) - \tau_{s',R,\ell}(M_t^{s'}).$$

Subtracting gives

$$U_{s',R,\ell} - U_{s,R,\ell} = \Delta A_{R,\ell}(s,s') - \Delta\tau_{R,\ell}(s,s').$$

Now let $\{R_b : b \in \mathcal{B}\}$ be a burden-indexed receiver family and define

$$\Psi_{s,s'}(b) := U_{s',R_b,\ell} - U_{s,R_b,\ell}.$$

If $\Psi_{s,s'}$ is continuous and strictly single-crossing, its sign changes at most once. If it is negative at low burden and positive at high burden, the intermediate value theorem gives a crossing $b^\star$ with $\Psi_{s,s'}(b^\star) = 0$, and strict single-crossing makes this crossing unique since the stronger sender helps exactly when $\Psi_{s,s'}(b) > 0$, it helps exactly on the high-burden side of $b^\star$. □

B.5 State Propagation

Proof of Proposition 7. The residual dynamics are

$$D_{u+1} = k_u + \lambda_u[D_u - q_u]_+, \qquad \widetilde{D}_{u+1} = k_u + \lambda_u[\widetilde{D}_u - q_u]_+.$$

If $D_u > q_u$ and $\widetilde{D}_u > q_u$, then clipping is inactive:

$$[D_u - q_u]_+ = D_u - q_u, \qquad [\widetilde{D}_u - q_u]_+ = \widetilde{D}_u - q_u.$$

Therefore

$$\Delta_{u+1} = \widetilde{D}_{u+1} - D_{u+1} = \lambda_u(\widetilde{D}_u - D_u) = \lambda_u \Delta_u.$$

Iterating for $u = t, \dots, t+h-1$ gives

$$\Delta_{t+h} = \left(\prod_{u=t}^{t+h-1} \lambda_u \right) \Delta_t.$$

If clipping becomes active for either trajectory, the difference is no longer propagated linearly; this is the resynchronization regime, as represented by the clipped residual model. For vector-valued residuals, the same argument applies componentwise when the carry condition holds componentwise. □

B.6 Protocol-Control Regret Bound

This subsection records the elementary plug-in guarantee for the deployed protocol controller. Let

$$\alpha_t^\star \in \operatorname*{argmax}_{\alpha \in \mathcal{A}_t} J_t(\alpha)$$

be the ideal action under the analysis objective, and let

$$\widehat{\alpha}_t \in \operatorname*{argmax}_{\alpha \in \mathcal{A}_t} \widehat{J}_t(\alpha)$$

be the deployed action chosen by calibrated estimates.

Proposition 9 (Plug-In Selector Regret)**.** *Suppose that for every $\alpha \in \mathcal{A}_t$,*

$$|\widehat{J}_t(\alpha) - J_t(\alpha)| \le \epsilon.$$

Then the one-step value regret of the deployed selector satisfies

$$J_t(\alpha_t^\star) - J_t(\widehat{\alpha}_t) \le 2\epsilon.$$

Proof. By optimality of $\widehat{\alpha}_t$ under $\widehat{J}_t$,

$$\widehat{J}_t(\widehat{\alpha}_t) \ge \widehat{J}_t(\alpha_t^\star).$$

Using the uniform error bound twice,

$$J_t(\alpha_t^\star) \le \widehat{J}_t(\alpha_t^\star) + \epsilon \le \widehat{J}_t(\widehat{\alpha}_t) + \epsilon \le J_t(\widehat{\alpha}_t) + 2\epsilon.$$

Rearranging gives the claim. □

Corollary B.7 (Termwise Calibration Error)**.** *Suppose the deployed score uses*

$$\widehat{J}_t(\alpha) = \widehat{A}_t(\alpha) - \widehat{\tau}_t(\alpha) - \eta_c \widehat{c}(\alpha) - \eta_r \widehat{r}_t(\alpha),$$

and for every $\alpha \in \mathcal{A}_t$,

$$|\widehat{A}_t(\alpha) - A_t(\alpha)| \le \epsilon_A, \qquad |\widehat{\tau}_t(\alpha) - \tau_t(\alpha)| \le \epsilon_\tau,$$
$$|\widehat{c}(\alpha) - c(\alpha)| \le \epsilon_c, \qquad |\widehat{r}_t(\alpha) - r_t(\alpha)| \le \epsilon_r.$$

Then

$$J_t(\alpha_t^\star) - J_t(\widehat{\alpha}_t) \le 2\big(\epsilon_A + \epsilon_\tau + \eta_c\epsilon_c + \eta_r\epsilon_r\big).$$

Proof. The termwise bounds imply

$$|\widehat{J}_t(\alpha) - J_t(\alpha)| \le \epsilon_A + \epsilon_\tau + \eta_c\epsilon_c + \eta_r\epsilon_r$$

uniformly over α. Apply Proposition 9. □

B.7 Idealized Specializations

These two specializations connect the receiver-relative accounting to factorized losses and conditional mutual information under the stated assumptions.

Proposition 10 (Factorized Target under Additive Loss)**.** *Suppose $Z_t^\star = (Z_t^{\mathrm{art}}, Z_t^{\mathrm{act}})$, the loss decomposes additively across the two factors, and the admissible receiver class factorizes into independent artifact and action predictors. Then the class-optimal risk and the receiver-relative gain decompose into artifact and action components.*

Proof. Under additive loss and a product receiver class, optimizing the joint predictor separates into optimizing the artifact predictor and the action predictor. The class-optimal joint risk is therefore the sum of the two class-optimal factor risks. Subtracting the no-message and message-conditioned versions gives the same additive decomposition for the receiver-relative gain. □

Proposition 11 (Variational Lower Bound on Idealized Benefit)**.** *Under the zero-slack specialization,*

$$A^\star_{C,\log}(M_t; Z_t^\star) = I(M_t; Z_t^\star \mid X_t^j).$$

Hence, for any uniformly bounded critic class $\mathcal{T}$,

$$A_{\mathcal{T}}(M_t; Z_t^\star \mid X_t^j) := \sup_{T \in \mathcal{T}} \left\{ \mathbb{E}_{p(X,M,Z)}[T] - \log \mathbb{E}_{p(X)p(M|X)p(Z|X)}[e^T] \right\} \le A^\star_{C,\log}(M_t; Z_t^\star).$$

Consequently,

$$\tau_{\rho,\log}(M_t) \ge A_{\mathcal{T}}(M_t; Z_t^\star \mid X_t^j) - U_{\rho,\log}(M_t; Z_t^\star).$$

Proof. Under zero slack, the log-loss receiver-relative gain equals conditional mutual information:

$$A^\star_{C,\log}(M_t; Z_t^\star) = I(M_t; Z_t^\star \mid X_t^j).$$

Conditional mutual information is the KL divergence

$$I(M_t; Z_t^\star \mid X_t^j) = \mathrm{KL}\Big(p(X_t^j, M_t, Z_t^\star) \;\Big\|\; p(X_t^j) p(M_t \mid X_t^j) p(Z_t^\star \mid X_t^j) \Big).$$

The Donsker–Varadhan representation of KL, restricted to the uniformly bounded critic class $\mathcal{T}$, gives the stated lower bound. The tax certificate follows by rearranging Proposition 1:

$$\tau_{\rho,\log}(M_t) = A^\star_{C,\log}(M_t; Z_t^\star) - U_{\rho,\log}(M_t; Z_t^\star).$$

□

C Additional Experimental Details and Results

This appendix extends the three main experimental claims with setup details, robustness checks, cross-family ablations, protocol-level tax knobs, and shared-state propagation studies. The main text contains the central identification results; this appendix adds the supporting evaluation sets, alternative protocols, and larger family sweeps that are useful for stress-testing the same theory under more varied conditions.

Each table states its evaluation support. Table 1 gives the Qwen main evaluation counts, while appendix audits use their stated supports. Paired analyses use common $(\mathrm{instance}, \mathrm{step})$ keys. OpenAI protocol accuracies use cell-specific counts, which can differ across conditions; those marginal rates are reported with Wilson intervals. Comparisons between tables retain each table's protocol and support definitions.

Appendix C.1 records the benchmark and protocol summary. Appendix C.2 adds robustness checks, token-accounting and no-overflow controls, strict-recoding diagnostics, gain-surrogate diagnostics, and additional raw/stronger-receiver controls. Appendix C.3 adds replay variants and additional closure plots. Appendix C.4 adds sender-replacement and cross-family threshold analyses. Appendix C.5 collects the broader OpenAI and weak-family ablations. Appendix C.6 reports the fuller protocol-knob, static-baseline, and selector results beyond Figure 6. Appendix C.7 reports the shared-state perturbation and repair analyses behind Proposition 7.

Table 4: **Benchmark and protocol summary.** The table centralizes the default sender/receiver pair, message budget, and role of each benchmark in the paper's experimental program.

Benchmark	Sender	Receiver	Budget	Main role	Note
BFCL	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	noise / high suff.	Main paired set.
ToolSandbox	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	noise / high suff.	Main paired set.
SWE-Lite	Qwen3-Coder-Next-FP8	Qwen3.5-9B	128	capability transfer	Main paired set.
τ-bench	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	weakly helpful / boundary	Main paired set.
ContextBench	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	helpful / context-heavy	Main paired set.
RepoBench	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	helpful / context-heavy	Main paired set.
LongCLI	Qwen3-Next-80B-A3B	Qwen3.5-9B	128	shared-state stress	Appendix trajectory set.

Table 5: **Benchmark-specific construction of action-sufficient targets.** The benchmarks do not share a native next-action label space. For scoring only, we project each benchmark's native supervision into a set $G_t = \{(Z^{\mathrm{art}}_{t,k}, Z^{\mathrm{act}}_{t,k})\}_{k=1}^{K_t}$ of acceptable artifact/action targets. `hit_joint` is one iff the parsed receiver output matches at least one pair in G_t. `hit_artifact` matches the artifact component, and h_{action} is action correctness conditional on artifact correctness. Sender, receiver, and selector never observe G_t.

Benchmark	Native supervision	Scorer projection	Scoring rule
BFCL	canonical function-call answer	artifact: function/API name; action: `tool-call` or `no-call`	match canonical function identity and action type; native BFCL evaluation separately normalizes arguments
ToolSandbox	scenario milestones over tool traces and world-state changes	artifact: valid milestone target; action: tool call, state update, add/remove, or user response	set-valued match against the current valid milestone frontier
ContextBench	human-verified gold code context, plus our harness-stage repair label when used	artifact: gold file, block, function, or line span; action: retrieval/read, or repair-stage edit	normalized overlap with gold context, or normalized target-file match for repair-stage diagnostics
RepoBench	retrieval target, next-line completion target, or retrieval+completion pipeline target	artifact: snippet index, candidate file, or cursor location; action: retrieval or completion	retrieval match for RepoBench-R/P; completion match for RepoBench-C/P
SWE-bench Lite	issue, gold patch, test patch, `FAIL_TO_PASS`, `PASS_TO_PASS`	artifact: gold-patch file/module or test ID; action: patch-stage edit or harness-stage test	scorer-side projection from gold patch and test metadata; final correctness remains SWE-bench patch-and-test execution
τ-bench	user task, domain tools/policy, action trace, and final goal database state	artifact: tool, entity, state field, or goal-state diff; action: tool call, state update, or response	set-valued projection from accepted tool/action traces or goal-state differences
LongCLI	task metadata, project specification, reference solution, and step-level test outcomes	artifact: task folder, command target, file, workspace artifact, or requirement; action: command, edit-feature, or test	normalized match to the reference command target, task folder, or step-level requirement target

C.1 Benchmark and Protocol Summary

The table is the bookkeeping layer behind the main-text experimental setup. The first six benchmarks support the three main empirical claims; LongCLI is reserved for the dynamic shared-state analyses in Appendix C.7. Unless otherwise stated, runs use the default Qwen3-Next-80B-A3B sender, the Qwen3.5-9B receiver, and a 128-token message budget on the support stated in the corresponding table. SWE-Lite uses Qwen3-Coder-Next-FP8 for the main MA sender condition, as listed in Table 4.

Concrete scorer-side mapping cases. The following boxes show one real benchmark row or trace per benchmark and how its native supervision is projected into the artifact/action vocabulary used by our scorer.

Case C.1: BFCL: canonical function-call row

Native benchmark row. `simple_python_0`: the user asks for the area of a triangle with base 10 and height 5.

Native target source. The canonical BFCL answer calls:

`calculate_triangle_area(base=10, height=5, unit=...).`

Projected scorer target. For the artifact/action scorer, we project the row to

$$G_t = \{(\texttt{calculate_triangle_area}, \texttt{tool-call})\}.$$

The argument values are retained by the native BFCL function-call evaluator, but the paper's $Z_t^{\text{art}}/Z_t^{\text{act}}$ projection uses the function identity and action type.

Scoring examples. An output that emits `calculate_triangle_area(...)` has `hit_artifact`=1 and `hit_joint`=1 under this projection. An output that emits no call has `hit_action`=0 and `hit_joint`=0.

Case C.2: ToolSandbox: weather-tool milestone

Native benchmark trace. Scenario family: `find_temperature`. The user asks: "What's the temperature here right now?"

Native target source. The scenario exposes the weather tool `search_weather_around_lat_lon`, and the valid milestone frontier includes a tool-trace milestone that requires this weather lookup at the current location.

Projected scorer target. For the tool-call step, we project the current milestone frontier to

$$G_t = \{(\texttt{search_weather_around_lat_lon}, \texttt{tool-call})\}.$$

If the scorer is evaluating the final user-response milestone rather than the tool-call milestone, the same scenario can instead project to a response target such as

$$G_t = \{(\texttt{temperature}, \texttt{user-response})\}.$$

Scoring examples. An output `tool-call/search_weather_around_lat_lon` is a joint hit for the tool-call milestone. An output `read/messages` is an artifact/action miss for that milestone, even if it is a plausible natural-language protocol step.

Case C.3: ContextBench: Django cascade-deletion case

Native benchmark case. Django cascade-deletion bug from the evaluated ContextBench set. The issue concerns a `UnicodeDecodeError` during `.delete()` because cascade collection fetches unnecessary fields with invalid UTF-8.

Native target source. The gold context/repair target identifies the cascade-collection implementation path:

`django/db/models/deletion.py`

and the relevant collector logic, including `Collector.collect`.

Projected scorer targets. For context-retrieval scoring:

$$G_t^{\text{retrieve}} = \{(\texttt{django/db/models/deletion.py}, \texttt{retrieve-context})\}.$$

For the repair-stage diagnostic used in our agent harness:

$$G_t^{\text{repair}} = \{(\texttt{django/db/models/deletion.py}, \texttt{edit})\}.$$

Scoring examples. An output `edit/django/db/models/deletion.py` is a joint hit under the repair-stage projection. An output `edit/django/db/models/base.py` has the right action type but the wrong artifact, so `hit_joint`=0.

Case C.4: RepoBench: `sastix/cms` retrieval/completion row

Native benchmark row. RepoBench-R Java row with

`repo_name=sastix/cms`

and file `server/src/main/java/com/sastix/cms/server/services/content/impl/ZipHandlerServiceImpl.java`. The row's next line is:

```
HashedDirectoryService hashedDirectoryService;
```

and the native retrieval target is `gold_snippet_index=5`.

Native target source. RepoBench-R supplies the gold snippet index for retrieval. RepoBench-C/P additionally evaluates next-line completion at the cursor.

Projected scorer targets. For retrieval:

$$G_t^{\text{retrieve}} = \{(\texttt{context[5]}, \texttt{retrieve-snippet})\}.$$

For completion:

$$G_t^{\text{complete}} = \{(\texttt{ZipHandlerServiceImpl.java@cursor}, \texttt{complete-line})\}.$$

Scoring examples. A retrieval output that selects `context[5]` is an artifact hit for RepoBench-R/P. A completion output at the correct cursor that emits the next-line action is scored under `complete-line`. Selecting a different context index is an artifact miss, even if the generated line is plausible.

Case C.5: SWE-bench Lite: `sqlfluff__sqlfluff-1625`

Native benchmark row. `sqlfluff__sqlfluff-1625`: the issue states that TSQL rule L031 incorrectly triggers "Avoid using aliases in join condition" when no join is present.

Native target source. The gold patch modifies:

`src/sqlfluff/rules/L031.py`.

The test patch includes:

`test/cli/commands_test.py`,

and `FAIL_TO_PASS` contains:

`commands_test.py::test__cli__command_directed`.

Projected scorer targets. For the patch-stage diagnostic:

$$G_t^{\text{patch}} = \{(\texttt{src/sqlfluff/rules/L031.py}, \texttt{edit})\}.$$

For the test-stage diagnostic:

$$G_t^{\text{test}} = \{(\texttt{commands_test.py::test__cli__command_directed}, \texttt{run-test})\}.$$

Scoring examples. An output `edit/src/sqlfluff/rules/L031.py` is a diagnostic joint hit for the patch-stage target. Running the listed failing test is a diagnostic joint hit for the test-stage target. Final task success is still decided by the native SWE-bench patch-and-test evaluator, not by our diagnostic projection alone.

Case C.6: τ-bench: retail exchange task

Native benchmark row. Retail task: Yusuf Rossi in ZIP code `19122` wants to exchange items in order `#W2378156`: a mechanical keyboard for a similar clicky one and a smart thermostat compatible with Google Home.

Native target source. The public task action trace includes:

```
find_user_id_by_name_zip(first_name=Yusuf, last_name=Rossi, zip=19122)
```

followed by order/product lookups and a final exchange call:

```
exchange_delivered_order_items(order_id=#W2378156, ...).
```

Projected scorer targets. For the first lookup step:

$$G_t^{\text{lookup}} = \{(\texttt{find_user_id_by_name_zip}, \texttt{tool-call})\}.$$

For the exchange step:

$$G_t^{\text{exchange}} = \{(\texttt{exchange_delivered_order_items:\#W2378156}, \texttt{tool-call})\}.$$

Scoring examples. A call to `find_user_id_by_name_zip` with the correct user-identifying intent is a joint hit for the lookup projection. A call to `exchange_delivered_order_items` on order `#W2378156` is a joint hit for the exchange projection. A tool call on an unrelated order or entity is an artifact miss.

Case C.7: LongCLI: `61810_cow`

Native benchmark task. LongCLI task `61810_cow`, a long-horizon CLI programming task derived from the copy-on-write fork lab.

Native target source. The task metadata points to a project workspace and instruction file for implementing copy-on-write fork behavior. LongCLI evaluates the agent through task-level and step-level requirement checks, including fail-to-pass and pass-to-pass test outcomes.

Projected scorer targets. For the implementation-stage diagnostic:

$$G_t^{\text{impl}} = \{(\texttt{61810_cow/}, \texttt{edit-feature})\}.$$

For the test-stage diagnostic:

$$G_t^{\text{test}} = \{(\texttt{task-test-harness}, \texttt{run-test})\}.$$

Scoring examples. An action that edits the `61810_cow` workspace toward the copy-on-write implementation is an artifact hit under the implementation-stage projection. An action that runs the task's test harness is an action hit for the test-stage projection. Whether the solution ultimately succeeds is determined by LongCLI's native requirements and regression tests.

Per-benchmark regime. Under the default Qwen protocol, BFCL and ToolSandbox are high-local-sufficiency settings where communication is fragile or harmful. ContextBench and RepoBench are context-heavy settings where the receiver's local view is incomplete and sender-side curation drives the signal. SWE-bench Lite probes capability transfer, and τ-bench is a weakly helpful, protocol-sensitive customer-service setting. LongCLI-Bench drives the shared-state stress test in Appendix C.7.

Replay frontier regret. On the frozen-message replay set, $\text{Regret}_r^{\text{replay}} = \max_{r'} h_{j,r'} - h_{j,r}$ measures how far a receiver's h_j falls below the best replay receiver on the same frozen message.

Run bookkeeping. Table 6 records the support convention used by each appendix numerical family. The OpenAI entries are aggregated from the stored per-step API records and their summary CSVs; no scorer-side targets are exposed to the models. The table distinguishes main-text supports from table-specific appendix diagnostic supports.

Table 6: **Run bookkeeping for appendix numerical tables.** The table records which support convention each appendix family uses. Main-text estimates use the supports stated in Table 1. Appendix diagnostics use table-specific supports over stored model outputs, except for the explicitly marked OpenAI receiver-side budget sweep.

Run family	Benchmarks	Records / evaluation support	Sender(s)	Receiver(s)	Decoding and budget
Released Qwen table cells	ContextBench, τ-bench, ToolSandbox, BFCL	ContextBench $n = 480$; τ-bench $n = 72$; ToolSandbox $n = 72$; BFCL $n = 100$	Qwen3-Next-80B-A3B	Qwen3.5-9B	det.; $B = 128$
Appendix diagnostic controls	ContextBench, ToolSandbox, τ-bench, BFCL; RepoBench; SWE-Lite	diagnostic supports: ContextBench 480; ToolSandbox/τ-bench 72; BFCL 100; RepoBench 480; SWE-Lite 96	Qwen3-Next-80B-A3B; Qwen3-Coder-Next-FP8 for SWE-Lite MA	Qwen3.5-9B	det.; $B = 128$
OpenAI model-family extension	ContextBench, ToolSandbox, τ-bench, BFCL	stored API run summaries, $n = 60$–300 per displayed cell	`gpt-5.4` in MA	six OpenAI receivers listed in Tables 18–19	API defaults; $B = 128$
Selector diagnostic ablation	ContextBench	stored selector-ablation file, $n = 300$	`gpt-5.4`	`gpt-4o-mini`	frozen selector
Static selector diagnostics	ToolSandbox, τ-bench	stored selector-policy files, $n = 36$ per policy	Qwen3-Next-80B-A3B	Qwen3.5-9B	static selector policies
Budget/no-overflow diagnostics	ContextBench, ToolSandbox	paired diagnostic rows $480/72$; no-overflow subsets $100/39$	Qwen3-Next-80B-A3B	Qwen3.5-9B	det.; threshold 8192 input tokens
OpenAI message-budget sweep	ContextBench, ToolSandbox	hard sample: 20 steps $\times$ 3 receivers $\times$ 6 budgets per benchmark	stored Qwen messages	`gpt-5.4-nano`, `gpt-5.4-mini`, `gpt-5.4`	receiver-side sweep

RepoBench and SWE-bench Lite diagnostic breakdowns. Table 7 reports appendix diagnostic breakdowns for the two supplemental benchmark families on the stated supports. These diagnostics examine the same regime patterns as the main text. The SWE-Lite main sender is Qwen3-Coder-Next-FP8. Its released sender-comparison manifest selects the last record per $(\text{instance}, \text{step})$ key: the main-sender slice uses seed 2 and the matched slice uses seed 1. The SWE-Lite contrast compares the recorded sender configurations; the matched run's exported receiver inputs predominantly contain an empty coordination-message field.

Paired uncertainty. The communication effects remained positive on ContextBench and RepoBench, negative on BFCL and ToolSandbox, and near zero on τ-bench. Table 8 reports paired differences on strict common $(\text{instance}, \text{step})$ keys. We computed instance-clustered percentile bootstrap intervals with fixed seeds and 5,000 resamples, and two-sided exact McNemar tests on discordant pairs. ContextBench, RepoBench, SWE-Lite, and BFCL remained significant after Bonferroni correction for six contrasts; τ-bench had $p = 1.0$. ContextBench and RepoBench use larger paired diagnostic supports than the earlier appendix audits, separate from the main-table evaluation sets.

The discordant wins were 499 versus 90 on ContextBench, 50 versus 0 on SWE-Lite, and 1 versus 24 on BFCL; matched-sender accuracy on the SWE-Lite support was zero. The clustered intervals on BFCL and τ-bench are wider than their step-level intervals, $[-0.3200, -0.1500]$ and $[-0.0694, +0.0694]$, while retaining the same conclusions. On ToolSandbox, the released common support contains 72 steps, with 64 correct RO outputs and 47 correct MA outputs. This gives a descriptive RO advantage of $17/72$, or 23.6 percentage points.

Table 7: **RepoBench and SWE-bench Lite appendix diagnostic breakdowns.** Reported step-level rates on the stated supports. The released sender contrasts use common $(\text{instance}, \text{step})$ keys after the manifest's duplicate policy is applied; the SWE-Lite seed mapping is specified above.

Benchmark	n	RO	SF	MA_{main}	MA_{match}	Δ_{comm}	Δ_{cmp}	Δ_{strong}
RepoBench	480	0.642	0.823	0.833	0.746	+0.192	+0.010	+0.088
SWE-bench Lite	96	0.000	0.052	0.521	0.000	+0.521	+0.469	+0.521

Table 8: **Paired communication effects with uncertainty.** Intervals are clustered by instance. Support is the number of paired steps; ContextBench and RepoBench contain 799 and 800 instances, respectively. The SWE-Lite contrast pairs instance-step keys across main-sender seed 2 and matched-sender seed 1.

Benchmark	Contrast	Steps	Effect	95% CI
ContextBench	MA − RO	2,390	$+0.1711$	$[0.1417, 0.2012]$
RepoBench	MA − RO	2,396	$+0.1611$	$[0.1340, 0.1891]$
SWE-Lite	$\text{MA}_{\text{main}} - \text{MA}_{\text{match}}$	96	$+0.5208$	$[0.3854, 0.6563]$
τ-bench	MA − RO	72	0.0000	$[-0.0833, +0.0694]$
BFCL	MA − RO	100	-0.2300	$[-0.3900, -0.0800]$

SWE-Lite action validity. The zero receiver-only joint score reflected the selected action despite accurate file recovery. All outputs on the audited 96-step slices from each of three seeds were parse-valid and had nonempty action and artifact fields; artifact accuracy ranged from 0.885 to 0.917. Released seeds 0 and 2 each contained 79 `read` and 17 `run_test` outputs on 96 steps, with zero joint hits where the repair-stage target was `edit`. Seed 1 contained two `edit` outputs and had 2/96 joint hits. These are the next-action scores defined in Table 5; final SWE-bench correctness uses patch and test execution.

C.2 Extended Compression Results

Table 9 and Figure 8 examine larger receivers and raw-context access on ToolSandbox. Substantial error remains in these MA and SF configurations, including with 80B and 397B receivers. Together with the main 9B results, these controls motivate evaluating communication separately for each receiver and interface.

Budget accounting and no-overflow controls. Table 10 reports realized protocol-level token accounting from stored diagnostic step records. The columns separate the bounded sender message from the total input tokens consumed by each protocol row, so the RO, SF, and MA rows are comparable under the same tokenizer accounting. The final column is a no-overflow control: we pair rows by $(\text{instance}, t)$, keep only examples for which all compared RO/SF/MA rows stay below 8192 total input tokens, and recompute `hit_joint`. This is conservative for MA because the protocol-level total includes both sender-side and receiver-side calls. The same sign pattern remains on this short-context diagnostic subset. ContextBench still has MA above both RO and SF, while ToolSandbox still has MA below RO/SF. Thus the reported communication gains and failures are not explained only by SF context-window overload.

Message truncation. Communication improved ContextBench accuracy within both capped and uncapped message groups. We split the 480-step, $B = 128$ audit in Table 10 by whether the sender message was truncated, keeping the protocol fixed. Table 11 shows that MA exceeded both RO and SF in each group. The groups pool to the reported RO/SF/MA accuracies of 0.633/0.294/0.775.

The paired 95% intervals for MA−RO were $[+5.7, +15.8]$ pp in the capped group and $[+22.4, +43.4]$ pp in the uncapped group. All 404 capped messages retained `action_type` and `target_artifact`, and the receiver followed both fields in 402 cases. The cap therefore removed trailing content while preserving the decision fields. In a separate paired Qwen run on 72 ContextBench steps, doubling the message budget reduced truncation from 0.8889 to 0.2778 and increased accuracy from 0.8194 to 0.8333.

Table 9: **Receiver-scaling and SF controls on ToolSandbox.** MA uses 80B or 397B receivers; SF uses an 80B receiver. These configurations complement the 9B receiver results in Table 1.

Setting	Benchmark	Mean	Note
MA 80B→80B	ToolSandbox	0.750	stronger receiver baseline
MA 80B→397B	ToolSandbox	0.694	stronger receiver baseline
SF (80B receiver)	ToolSandbox	0.260	full-context baseline (80B receiver)

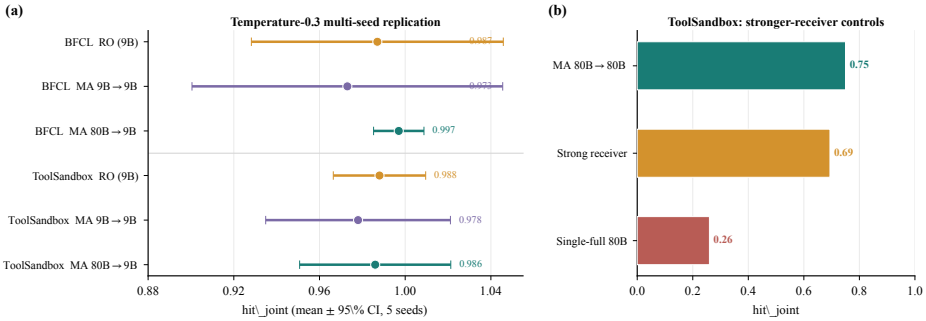

Figure 8: **Additional compression-effect controls.** (a) Five-seed replication at temperature 0.3 on BFCL and ToolSandbox. (b) Stronger-receiver and SF controls on ToolSandbox.

Table 10: **Token accounting and no-overflow controls.** Token counts are realized counts from the stored diagnostic records. The no-overflow column filters to paired rows where RO, SF, and MA all use at most 8192 protocol-level input tokens, giving $n = 100$ ContextBench rows and $n = 39$ ToolSandbox rows. These diagnostics use the stated appendix support and are not used as replacements for the main-text estimates.

Benchmark	Protocol	n	`hit_joint`	Message tok. mean / p90	Total protocol input tok. mean / p90	Trunc. rate	No-overflow `hit_joint`
ContextBench	RO	480	0.633	0.0 / 0.0	6,899 / 12,267	0.000	0.610
ContextBench	SF	480	0.294	0.0 / 0.0	24,486 / 51,642	0.000	0.280
ContextBench	MA, B=128	480	0.775	127.2 / 128.0	16,365 / 28,160	0.842	0.790
ToolSandbox	RO	72	0.889	0.0 / 0.0	1,050 / 1,672	0.000	0.923
ToolSandbox	SF	72	0.903	0.0 / 0.0	1,273 / 2,054	0.000	0.923
ToolSandbox	MA, B=128	72	0.653	115.2 / 128.0	2,534 / 4,000	0.208	0.590

Table 11: **ContextBench accuracy by message truncation.** Both groups use the same $B = 128$ protocol. The final two columns report paired accuracy differences in percentage points.

Messages	Steps	RO	SF	MA	MA−RO	MA−SF
Capped	404	0.644	0.248	0.750	+10.64	+50.25
Uncapped	76	0.579	0.539	0.908	+32.89	+36.84

Message form and receiver instruction. On a common ToolSandbox slice, canonical JSON and a decision-header template both reached 0.750 accuracy, compared with 0.556 for prose, a 19.4 pp difference. These comparisons varied the user-turn content while keeping `RECEIVER_SYSTEM_PROMPT` fixed. The released sequential agent and four-action threshold controller share this receiver instruction. The five-action scorer described in Appendix C.6 uses user-message probes.

OpenAI receiver-side message-budget sweep. We reran the receiver-side budget sweep with stored Qwen sender messages, varying only the visible coordination-message budget. The hard sample prioritized stored sender failures, already-truncated messages, and longer protocol inputs. The sweep covered three receiver aliases, `gpt-5.4-nano`, `gpt-5.4-mini`, and `gpt-5.4`, on 20 sampled steps per benchmark and six budgets. Table 12 shows distinct budget responses. ContextBench improves from $B = 4$ to the $B \geq 16$ range, while ToolSandbox degrades as more of the sender

message is exposed. The effect of message budget therefore depends on the receiver's task: additional content can supply useful context or add protocol tax.

Table 12: **OpenAI receiver-side hard-sample budget sweep.** Rows report `hit_joint` on 20 hard-sampled steps per benchmark and receiver alias, using stored Qwen sender messages and varying only the visible message budget. The hard sample prioritizes stored sender failures, sender-message truncation, and longer protocol inputs. The run completed 720 API calls with no failed calls.

Benchmark	B	`nano`	`mini`	`gpt-5.4`	Mean
ContextBench	4	0.700	0.600	0.400	0.567
ContextBench	8	0.650	0.650	0.600	0.633
ContextBench	16	0.700	0.650	0.750	0.700
ContextBench	32	0.700	0.700	0.700	0.700
ContextBench	64	0.650	0.700	0.700	0.683
ContextBench	128	0.700	0.700	0.700	0.700
ToolSandbox	4	0.900	0.950	0.800	0.883
ToolSandbox	8	0.550	0.700	0.750	0.667
ToolSandbox	16	0.350	0.600	0.800	0.583
ToolSandbox	32	0.350	0.400	0.450	0.400
ToolSandbox	64	0.250	0.300	0.250	0.267
ToolSandbox	128	0.250	0.250	0.250	0.250

Strict recoding probe. The strict-recoding probe holds the scorer-side artifact/action target fixed while changing only the receiver-facing surface form of the sender recommendation. The direct condition exposes canonical field names and action words; the strict condition replaces them with semantically indirect labels that preserve the benchmark target for an unbounded decoder but stress the deployed receiver interface. Table 13 reports the stored direct-versus-strict diagnostic comparison. BFCL is unaffected because its function-call schema is already highly canonicalized, while ContextBench, ToolSandbox, and especially τ-bench lose joint accuracy under the recoded surface. This is the empirical counterpart of Theorem 3: preserving the Bayes target is not enough when the receiver class is representation-sensitive.

Table 13: **Direct versus strict recoded sender messages.** The evaluated target G_t is unchanged across conditions; only the receiver-facing representation changes. Values are recomputed from the stored recoding-probe records on the stated diagnostic supports.

Benchmark	n	Direct	Strict recode	Drop
BFCL	200	0.985	0.985	0.0 pp
ContextBench	100	1.000	0.940	6.0 pp
ToolSandbox	200	1.000	0.965	3.5 pp
τ-bench	155	1.000	0.865	13.5 pp

Gain-surrogate diagnostic. The main text uses

$$\hat{A} := \texttt{surprisal_no_msg} - \texttt{surprisal_bits}$$

as a surrogate for receiver-relative gain. Both terms are hit-based proxy scores, not measured log-probabilities. The scorer sets `surprisal_bits` to 0.5, 3, or 6 bits when the receiver's output is a joint hit, a partial hit (action or artifact only), or a miss, and sets `surprisal_no_msg` in the same way from a receiver call without the message. $\hat{A}$ therefore records how the hit outcome changes when the message is exposed, on a coarse surprisal scale, rather than a measured log-loss reduction; under an idealized zero-slack calibrated log-loss specialization, the expected log-loss reduction would correspond to conditional mutual information. Figure 9 compares the distribution of $\hat{A}$, conditional joint accuracy, and benchmark averages. The high-$\hat{A}$ bucket contains 74% of BFCL steps, 56% of ToolSandbox steps, 67% of ContextBench steps, and 25% of SWE-Lite steps. High $\hat{A}$ therefore occurs in both helpful and harmful communication regimes. The surrogate describes the gain component; assessing deployed utility also requires the protocol tax.

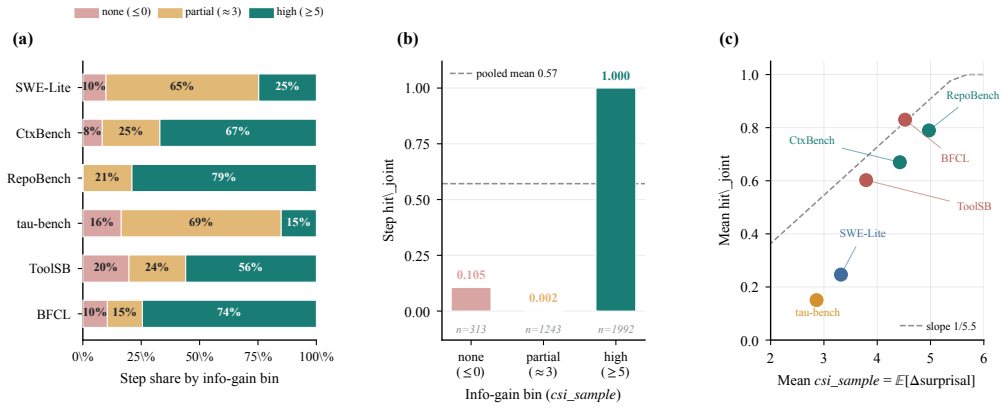

Figure 9: **Gain-surrogate view of message utility.** (a) Per-benchmark composition of the step-level hit-based proxy gain surrogate `csi_sample`, split into none, partial, and high bins. (b) Step-level `hit_joint` conditional on gain-surrogate bin. (c) Benchmark-level mean gain surrogate versus mean `hit_joint`. High-surrogate steps occur in both helpful and harmful communication regimes.

These diagnostics complement the format comparisons in Table 22. The selector in Section 3.2 compares each bounded-message action with `raw` using both gain and tax.

C.3 Extended Closure Results

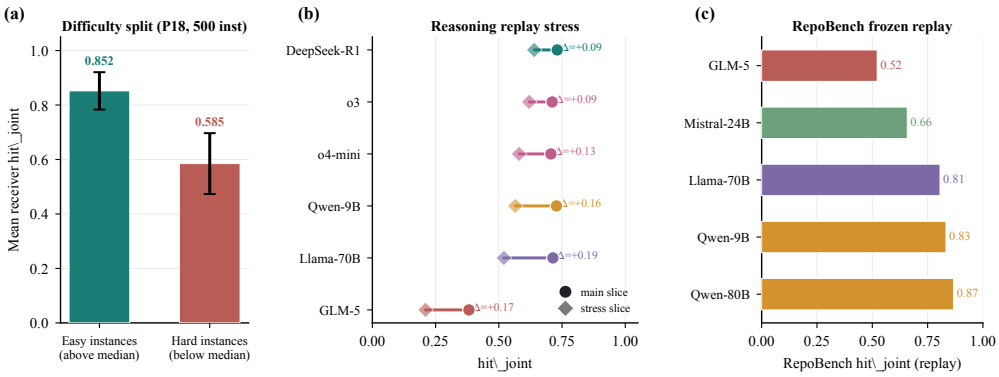

Figure 10: **Additional replay and closure analyses.** Panels report easy/hard replay partitions, replay stress across receivers, and RepoBench replay accuracy under frozen messages. The same decomposition remains evident: some receivers are primarily content-limited, whereas the main outliers are primarily closure-limited.

The main text uses fixed-message replay and the OpenAI full-context rows to examine action-closure loss. Figure 10 extends this in three ways. First, it shows that the content/closure split remains stable under harder/easier partitions of the same replay set. Second, it adds replay stress on additional receivers, confirming that the main outliers remain outliers under a frozen message. Third, it applies the same logic to RepoBench, where the absolute accuracy scale changes but the interpretation of the bottleneck axes remains the same.

With the sender message fixed, these additional receivers and task partitions show that post-message error can concentrate in action closure, as described by Proposition 5 and Theorem 4.

Parsing and action choice. The closure failures persisted after valid outputs recovered the correct artifact. We re-parsed a separate replay set with a `gpt-5.4` sender using robust and permissive parsers. Table 14 shows that permissive parsing changed validity by at most 0.2 pp, while wrong-action rates remained 40.8–94.0% among valid outputs with the correct artifact. GLM-5 and MiniMax-M2.5 named `astropy/modeling/separable.py`, and `o3` named `astropy/table/table.py`, while explicitly selecting `read` where the evaluated action was `edit`.

Table 14: **Parsing audit under a `gpt-5.4` sender replay.** Validity columns use all audited outputs. Wrong-action rates condition on valid parsing and a correct artifact. This is a separate sender condition from Table 2. All entries are percentages.

Receiver	Robust valid	Permissive valid	Wrong action
GLM-5	99.6	99.6	44.5
MiniMax-M2.5	86.8	87.0	94.0
`o3`	97.3	97.3	40.8

An audit of all 300 paired `o3` RO/SF steps reproduced accuracies 0.223/0.133. Robust parse validity was 99.33% for RO and 89.00% for SF. The discordant counts were 52 RO wins versus 25 SF wins; 41 of the 52 SF losses retained the correct artifact and selected the wrong action.

C.4 Extended Stronger-Sender Results

Figure 11 extends the main-text sender-gap story in two directions. First, it shows that the strong-sender effect is not a same-family phenomenon: across additional sender families, the sign and rough ranking remain task-dependent. Second, it makes clear that the threshold is not only a benchmark-level average, but also visible in sender-family matrices and pilot-set frontier plots.

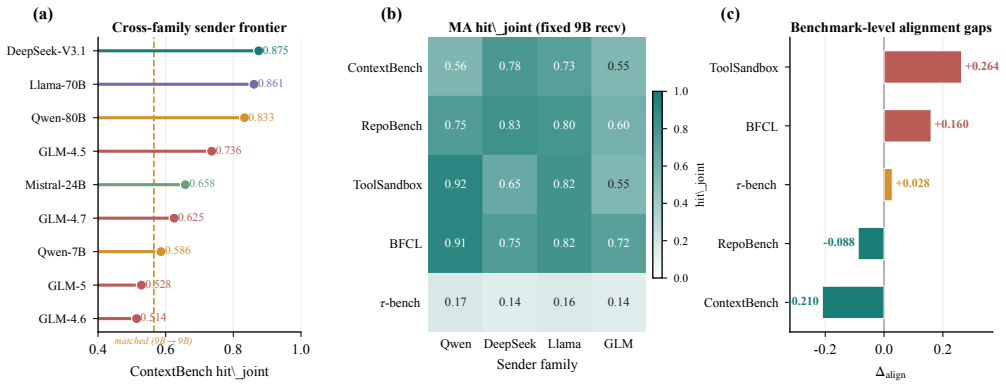

Figure 11: **Additional sender-threshold analyses.** Panels show a ContextBench sender frontier, a sender-family accuracy matrix, and benchmark-level effects with a fixed receiver. Panel (c) uses $\Delta_{\text{align}} = -\Delta_{\text{strong}}$, so positive values favor the matched sender.

Sender-replacement ablation. The sender-replacement study in Figure 12 fixes the Qwen3.5-9B receiver and swaps the sender across seven candidates spanning multiple families and scales. This is the most direct test of the sender-gap theorem under a fixed receiver. On ContextBench, the non-GLM strong senders remain above the RO baseline, while the weakest GLM senders fall below it. On ToolSandbox, where local sufficiency is already high, even some strong senders that help on ContextBench fall below the RO threshold. The point is not that one sender family is universally better, but that sender helpfulness is regime-specific because both the extra receiver-relative gain and the induced tax vary across task settings.

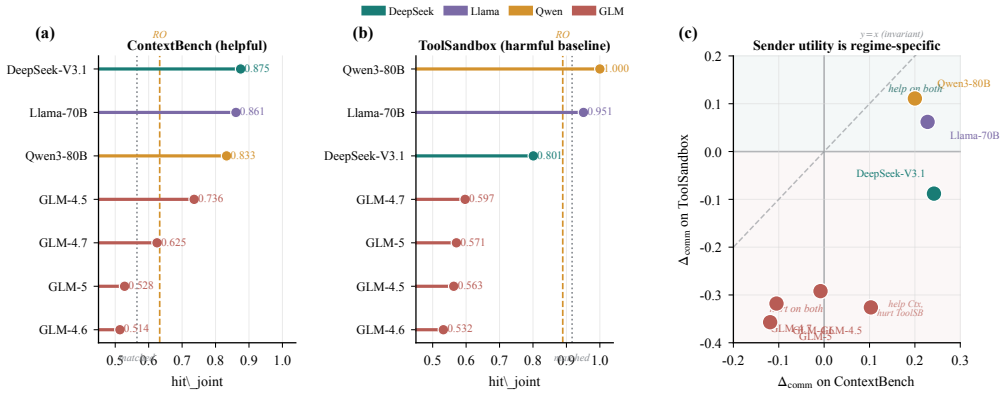

Figure 12: **Sender-replacement ablation (fixed Qwen3.5-9B receiver).** (a) ContextBench sender ranking against RO and matched-sender references. (b) The same sender set on ToolSandbox. (c) Per-sender signed communication effect on ContextBench versus ToolSandbox. A sender that helps in one setting can hurt in another, so sender choice is a task-level protocol decision rather than a fixed ranking of models.

This ablation complements Figure 5. Figure 5 fixes the sender pair and varies the benchmark regime. Figure 12 fixes the receiver and benchmark condition and varies the sender identity. Together, they show that the threshold binds to the sender–receiver–task combination, not only to a coarse family label.

Statistical sender-threshold fit. The sender-upgrade theorem predicts a sign change in the value of a stronger sender as receiver-local sufficiency increases: when RO is low, the upstream sender has room to add receiver-relative gain; when RO is already high, additional sender complexity is more likely to appear as protocol tax. We fit this relationship on the released benchmark-cell anchors,

using $\Delta_{\text{strong}} = \text{MA}_{\text{strong}} - \text{MA}_{\text{matched}}$ as the response and RO accuracy as the local-sufficiency proxy. Across 22 anchors, consisting of 16 OpenAI-family cells and 6 Qwen cells, the linear fit is

$$\widehat{\Delta}_{\text{strong}} = 0.286 - 0.449\,\text{RO}, \qquad R^2 = 0.492.$$

The fitted zero-crossing is $\text{RO} = 0.638$, with bootstrap 95% interval $[0.491, 0.780]$. A segmented hinge fit gives a similar R^2 but a wider interval because the released anchors contain few very-low-RO cells. We therefore use the linear fit as the appendix summary statistic and keep the theorem's condition as a sufficient-condition interpretation rather than a universal scaling law.

Table 15: **Sender-upgrade threshold fit.** The response is $\Delta_{\text{strong}} = \text{MA}_{\text{strong}} - \text{MA}_{\text{matched}}$; RO accuracy is used as the observable local-sufficiency proxy. The negative fitted slope supports the theorem's regime prediction: stronger senders help where local sufficiency is low and become tax-dominated where RO is high.

Fit	Anchors	Slope on RO	R^2	Crossover RO	95% CI
Linear	22	-0.449	0.492	0.638	[0.491, 0.780]
Segmented / hinge	22	piecewise	0.503	0.300	[0.300, 0.900]

Expanded sender-swap support. Adding the stronger-sender condition to every receiver/benchmark cell that lacked one expanded the paired anchor set from 22 to 41. The fitted crossover remained close to the original estimate, while its bootstrap interval narrowed from $[0.491, 0.780]$ to $[0.552, 0.706]$ (Table 16). Among 38 anchors with nonzero effects, 34 (89.5%) agreed with one positive-to-negative crossing as RO accuracy increased. A hierarchical model with anchor random effects over the 12,540 paired steps preferred this structure to the linear/no-threshold and two-crossing alternatives on all 10 folds separated by anchor.

Table 16: **Expanded sender-upgrade crossover analysis.** The response is Δ_{strong} and the predictor is receiver-only accuracy, as in Table 15.

Analysis	Anchor set	Crossover	95% CI
Submitted fit	22	0.6376	$[0.491, 0.780]$
Expanded fit	41	0.6289	$[0.552, 0.706]$
Leave one benchmark family out	41	0.6451	
Excluding SWE-Lite	41	0.6154	

Within-family identity check. Table 17 records the OpenAI same-family self-pair controls available in the stored run. These controls separate the effect of changing the sender identity from the effect of adding a same-family message channel.

Table 17: **Within-family sender identity check.** Rows use the same-family MA self-pairs from the stored OpenAI run. Deltas compare each self-pair against the receiver's RO value for the same benchmark condition.

Sender → receiver	ContextBench acc.	Δ vs RO	ToolSandbox acc.	Δ vs RO
`gpt-4o-mini→gpt-4o-mini`	0.867	+0.217	0.960	+0.157
`gpt-5.4-nano→gpt-5.4-nano`	0.873	+0.380	0.960	+0.203
`gpt-5.4-mini→gpt-5.4-mini`	0.867	+0.253	0.913	+0.217

C.5 Cross-Family Model Ablations

These model-family sweeps compare compression, action closure, and sender upgrades beyond the default Qwen protocol.

OpenAI model-family extension. On four of the six benchmarks, we evaluated the API model aliases `gpt-4o-mini`, `gpt-5.4-nano`, `gpt-5.4-mini`, `gpt-5.4`, `o3`, and `o4-mini` in SF, RO, and the `gpt-5.4`→receiver MA configuration. The model names identify the recorded API

aliases; Llama-70B denotes `meta-llama/Llama-3.3-70B-Instruct-Turbo`. The OpenAI extension preserves the ContextBench communication gains and reveals a different ToolSandbox response: MA improves over RO for these receivers, whereas it reduces accuracy for the default Qwen receiver. Tables 18 and 19 report the stored step counts and Wilson intervals for the stated diagnostic supports.

Table 18: **OpenAI model-family extension counts and Wilson intervals, ContextBench and ToolSandbox.** Each entry is `hit_joint` with Wilson 95% interval over the stored step records for that diagnostic cell. MA uses `gpt-5.4` as the sender; the stated counts are table-specific appendix supports.

Benchmark	Receiver	$n_{\mathrm{RO}}/n_{\mathrm{SF}}/n_{\mathrm{MA}}$	RO h_j	SF h_j	MA h_j
ContextBench	`gpt-4o-mini`	300/300/300	0.650 [0.594,0.702]	0.743 [0.691,0.789]	0.787 [0.737,0.829]
ContextBench	`gpt-5.4-nano`	300/300/300	0.493 [0.437,0.550]	0.637 [0.581,0.689]	0.783 [0.733,0.826]
ContextBench	`gpt-5.4-mini`	300/300/300	0.613 [0.557,0.667]	0.770 [0.719,0.814]	0.787 [0.737,0.829]
ContextBench	`gpt-5.4`	300/300/150	0.600 [0.544,0.654]	0.777 [0.726,0.820]	0.853 [0.788,0.901]
ContextBench	`o3`	300/300/150	0.223 [0.180,0.274]	0.133 [0.099,0.176]	0.493 [0.414,0.573]
ContextBench	`o4-mini`	300/300/300	0.500 [0.444,0.556]	0.617 [0.561,0.670]	0.743 [0.691,0.790]
ToolSandbox	`gpt-4o-mini`	300/300/300	0.803 [0.755,0.844]	0.957 [0.927,0.975]	1.000 [0.987,1.000]
ToolSandbox	`gpt-5.4-nano`	300/300/300	0.757 [0.705,0.802]	0.927 [0.891,0.951]	1.000 [0.987,1.000]
ToolSandbox	`gpt-5.4-mini`	300/300/300	0.697 [0.642,0.746]	0.890 [0.850,0.921]	1.000 [0.987,1.000]
ToolSandbox	`gpt-5.4`	300/300/150	0.653 [0.598,0.705]	1.000 [0.987,1.000]	0.993 [0.963,0.999]
ToolSandbox	`o3`	150/150/150	0.740 [0.664,0.804]	0.980 [0.943,0.993]	1.000 [0.975,1.000]
ToolSandbox	`o4-mini`	300/300/150	0.677 [0.622,0.727]	0.733 [0.681,0.780]	1.000 [0.975,1.000]

Table 19: **OpenAI model-family extension counts and Wilson intervals, τ-bench and BFCL.** Each entry is `hit_joint` with Wilson 95% interval over the stored step records for that diagnostic cell. MA uses `gpt-5.4` as the sender; the stated counts are table-specific appendix supports.

Benchmark	Receiver	$n_{\mathrm{RO}}/n_{\mathrm{SF}}/n_{\mathrm{MA}}$	RO h_j	SF h_j	MA h_j
τ-bench	`gpt-4o-mini`	150/150/150	0.253 [0.190,0.328]	0.020 [0.007,0.057]	0.373 [0.300,0.453]
τ-bench	`gpt-5.4-nano`	150/150/150	0.313 [0.245,0.391]	0.233 [0.173,0.307]	0.300 [0.232,0.378]
τ-bench	`gpt-5.4-mini`	150/150/150	0.293 [0.226,0.371]	0.233 [0.173,0.307]	0.327 [0.257,0.405]
τ-bench	`gpt-5.4`	150/150/75	0.240 [0.179,0.314]	0.180 [0.127,0.249]	0.147 [0.084,0.244]
τ-bench	`o3`	75/75/75	0.200 [0.125,0.304]	0.173 [0.104,0.274]	0.160 [0.094,0.259]
τ-bench	`o4-mini`	150/150/75	0.273 [0.208,0.350]	0.280 [0.214,0.357]	0.160 [0.094,0.259]
BFCL	`gpt-4o-mini`	180/180/180	0.878 [0.822,0.918]	0.722 [0.653,0.782]	0.833 [0.772,0.881]
BFCL	`gpt-5.4-nano`	180/180/180	0.883 [0.828,0.922]	0.811 [0.748,0.862]	0.839 [0.778,0.885]
BFCL	`gpt-5.4-mini`	180/180/180	0.839 [0.778,0.885]	0.833 [0.772,0.881]	0.844 [0.784,0.890]
BFCL	`gpt-5.4`	180/180/60	0.861 [0.803,0.904]	0.833 [0.772,0.881]	1.000 [0.940,1.000]
BFCL	`o3`	60/60/60	1.000 [0.940,1.000]	0.983 [0.911,0.997]	1.000 [0.940,1.000]
BFCL	`o4-mini`	180/180/60	0.844 [0.784,0.890]	0.889 [0.835,0.927]	1.000 [0.940,1.000]

Panels (a) and (b) compare compressed and full-context execution. On ContextBench, OpenAI MA exceeds SF for each displayed receiver. On ToolSandbox, the SF and MA endpoints often approach ceiling, and MA improves over RO. This differs from the default Qwen condition, where MA reduces accuracy. The comparison shows that the communication regime depends on the receiver and protocol as well as the benchmark.

Panel (c) extends the closure diagnostic. Non-reasoning OpenAI receivers remain largely content-limited, while reasoning receivers continue to exhibit closure-sensitive behavior, the cross-family analog of Figure 4.

Panel (d) adds weak-family baselines: at a smaller scale, matched MA sits below RO even on ContextBench, matching the sender-gap theorem when the extra gain fails to cover the protocol tax. Panel (e) reports five-seed BFCL and ToolSandbox results at temperature 0.3. Panel (f) aligns all receivers on a single ContextBench axis, summarizing the preceding panels.

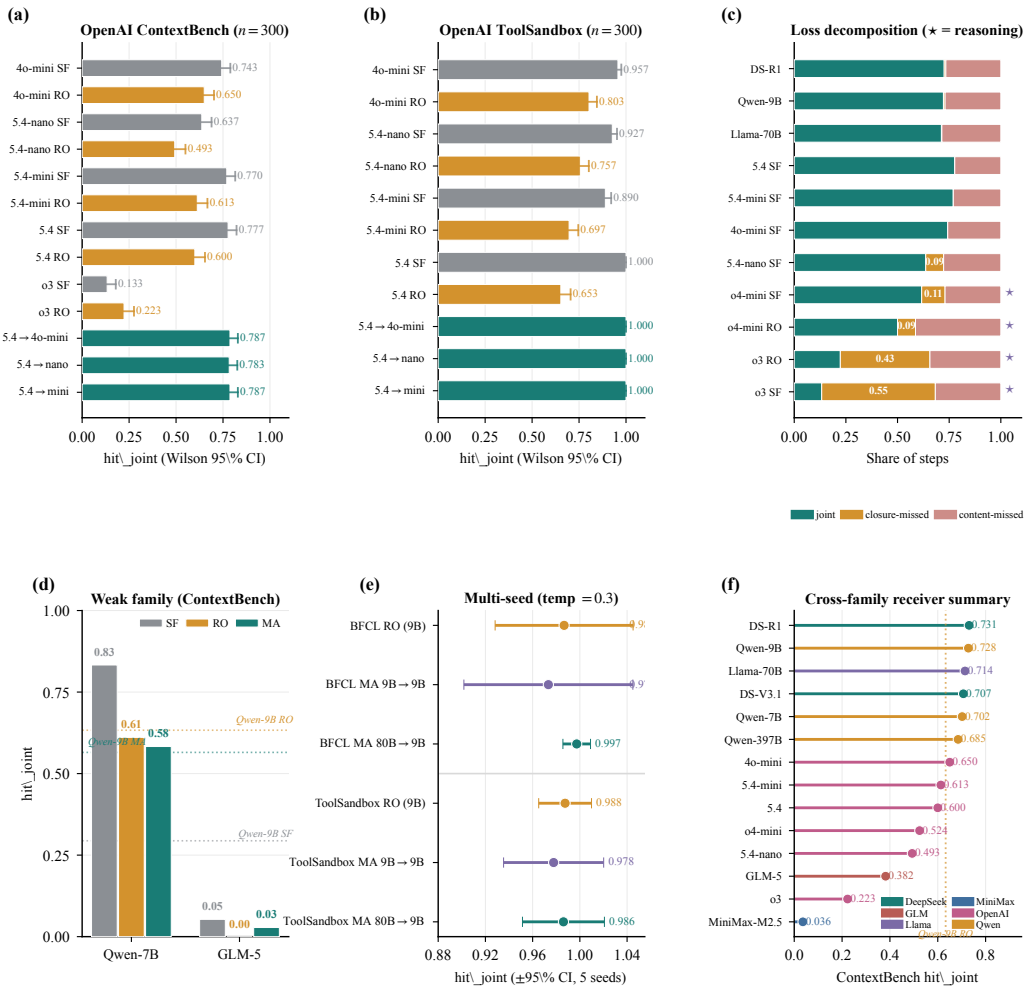

Figure 13: **Cross-family ablations.** (a) OpenAI model extensions on ContextBench. (b) OpenAI ToolSandbox. (c) ContextBench content/closure diagnostics across OpenAI and replay receivers. (d) Weak-family full-protocol baselines. (e) Five-seed replication at temperature 0.3 on BFCL and ToolSandbox. (f) Cross-family receiver summary on ContextBench. Each panel extends one of the three main claims under a different family, receiver scale, or protocol stress.

C.6 Protocol-Level Tax Knobs

Calibration and execution. The early gate compares a receiver-sufficiency rate with a fixed threshold of 0.80. The rate is the mean joint-hit indicator on held-out RO records for the deployed interface. The gate selects `skip` when the rate meets or exceeds the threshold, with its configuration fixed before testing. This path returns one receiver generation before any sender or scorer calls.

After the gate, the five-action path generates and caches two real message candidates: `compress` summarizes upstream context under the message budget, and `refresh` summarizes the updated shared state. The `raw` condition appends full upstream context to the receiver observation. `escalate` reuses the compressed message and directly invokes the stronger receiver. Scoring uses five no-message and four conditioned probes, each capped at four tokens. Each probe pair uses the receiver that executes its action; conditioned probes use the actual candidate input. The gain surrogate clips the conditioned-minus-unconditioned leading-token log-probability difference at zero. The selected action reuses its cached candidate for one final receiver generation. Table 27 includes both candidate generations, all scoring probes, and the selected receiver call in each decision's cost.

The default calibration and routing split assigns 20% of instances to calibration or router training and 80% to testing, grouped by `instance_id`. Repeated-split studies use 20 partitions with split seeds 0–19 and a fixed receiver, interface, and action set. Metrics are computed within each test split and averaged equally across the 20 splits. Selector decoding defaults to temperature 0. Paired calibration records provide an action-level protocol-burden offset. This offset can absorb gain effects as well as protocol burden, so the per-instance gain and cost terms supply the instance-specific ranking signal. The selector reads no benchmark label, gold target, hit field, or realized correctness at test time. The static score uses $\eta_c = 10^{-4}$ on raw token counts and $\eta_r = 0$ for its one-step horizon. The shared-state experiments evaluate residual propagation and targeted refresh separately.

Selector ablation records. Table 20 reports the ContextBench selector ablation for the OpenAI `gpt-4o-mini` receiver cell on the stated appendix diagnostic support. Gain and tax removals are obtained from the stored ablation file; the cost-removal row is recomputed from the same stored RO/SF/MA records, with the cost coefficient set to zero. The file records `lambda_c`$= 10^{-4}$, which is the code-level η_c applied to raw token counts. It disables finite-horizon residual propagation on this static one-step condition, so $\eta_r = 0$ and the residual-removal row equals the full score. Cost is displayed as mean token count per step divided by the always-send baseline in the same file.

Table 20: **Selector ablation on ContextBench.** Rows are computed from stored OpenAI RO/SF/MA records for the `gpt-4o-mini` receiver and `gpt-5.4` sender on the stated diagnostic support. The comparison is an accuracy–cost frontier diagnostic: always-send has the highest raw accuracy in this static cell, while the full selector uses less than $0.54\times$ the always-send cost and remains above always-skip.

Selector variant	n	Acc.	Cost / send	Skip	Compress	Raw
Full score	300	0.737	0.536	175	11	114
No gain term	300	0.650	0.402	300	0	0
No tax term	300	0.737	0.536	175	11	114
No cost term	300	0.730	0.550	175	0	125
No residual term	300	0.737	0.536	175	11	114
Always skip	300	0.650	0.402	300	0	0
Always send	300	0.787	1.000	0	300	0
Always raw	300	0.743	0.669	0	0	300

Removing the gain term makes the selector choose `skip` on every step, reducing accuracy from 0.737 to 0.650. Removing the tax term leaves every selected action unchanged on this evaluation set. The residual-removal row also matches the full score because the static configuration sets $\eta_r = 0$. Removing the cost term changes the mixture from 11 compressed messages and 114 raw-context actions to 125 raw-context actions, with 175 skips; accuracy is 0.730. Always-send reaches the highest accuracy, 0.787, at the highest cost; the full selector uses 0.536 times that cost.

Static selector baselines. Table 21 compares the released four-action threshold controller with fixed policies on ToolSandbox and τ-bench. Here, `skip` suppresses communication, `send` uses the bounded MA message, and `rewrite` forwards a normalized evidence bundle. The adaptive controller uses local sufficiency, parsing, budget pressure, and schema fidelity to choose an action. On ToolSandbox, it attains 1.000 accuracy at 581 input tokens per step, close to the skip policy's cost. On τ-bench, rewriting achieves higher accuracy than the adaptive controller (0.139 versus 0.083) at nearly identical cost (2,506 versus 2,504 tokens).

Table 22 reports concrete protocol changes, including `skip`, bounded `compress`, evidence rewriting, and selective suppression. The `raw` action retains its full-context meaning. Figure 14 gives the corresponding format and budget sweeps, complementing the static and online comparisons in Figure 6.

Theorem 3 motivates including `raw` in protocol comparisons. Rewriting and budget changes vary preserved content, decoder compatibility, and protocol tax.

Calibration sensitivity. The selector changed little across calibration fractions from 5% to 40%. We evaluated 20 instance-disjoint splits per fraction with split seeds 0–19, for 100 splits in total. Mean accuracy spanned 0.90 pp and cost relative to always-send spanned 0.30 pp (Table 23). Pooling

Table 21: **Static selector diagnostics on protocol-sensitive benchmarks.** Rows are aggregated from the stored selector-policy records on the stated diagnostic supports. Token cost is realized total input tokens per step. These baselines complement the ContextBench term ablation in Table 20.

Benchmark	Policy	`hit_joint`	Mean input tokens
ToolSandbox	adaptive selector	1.000	581
ToolSandbox	always skip	0.972	577
ToolSandbox	always send	0.806	2,474
ToolSandbox	always rewrite	0.833	1,930
τ-bench	adaptive selector	0.083	2,504
τ-bench	always skip	0.056	711
τ-bench	always send	0.056	3,087
τ-bench	always rewrite	0.139	2,506

Table 22: **Representative protocol ablations on a harmful and a helpful benchmark.** The rows compare `skip`, bounded `compress`, evidence rewriting, and suppression under rewritten prompts and per-knob cost accounting. The evidence-rewrite row forwards a compact, normalized evidence bundle; its input and token cost differ from the full-context SF protocol in Table 1.

	ToolSandbox			ContextBench		
Condition	Acc.	Δ vs RO	Cost	Acc.	Δ vs RO	Cost
Receiver-only	0.889	0.000	1,374	0.633	0.000	7,251
MA (default)	0.653	-0.236	2,995	0.775	0.142	16,801
Prose	0.556	-0.333	2,905	0.694	0.061	4,183
Evidence-rewrite (knob set)	—	—	—	0.778	0.144	1,853
Canonical JSON	0.750	-0.139	3,004	0.764	0.131	4,162
Budget =16	0.896	0.007	3,676	0.736	0.103	3,705
Budget =32	1.000	0.111	3,717	0.833	0.200	3,729
Selective suppression	1.000	0.111	919	0.750	0.117	3,960

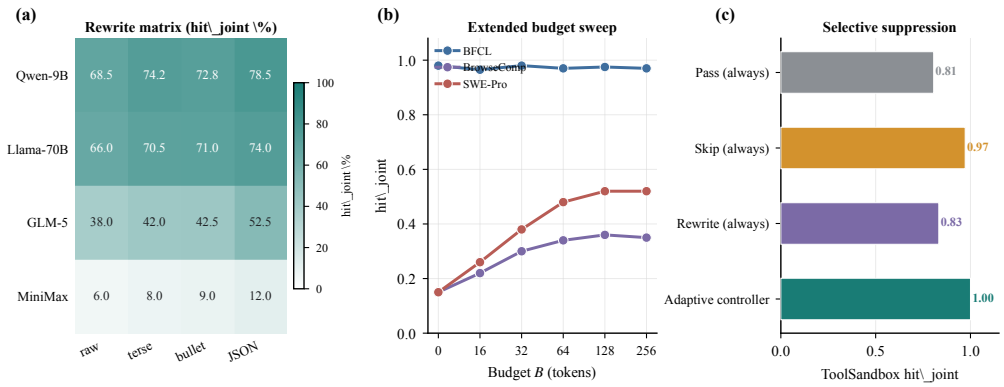

Figure 14: **Expanded protocol-knob analyses.** Panels report the grounded rewrite matrix, broader budget sweeps, and selective-suppression controls.

the ContextBench and ToolSandbox offset records without benchmark labels changed ContextBench accuracy from 0.7425 to 0.7389 and preserved all 36/36 ToolSandbox `skip` decisions. On the separate 300-step ablation support in Table 20, removing tax changed 0/300 choices, while removing gain selected `skip` on all 300 steps and reduced accuracy from 0.7367 to 0.6500. The full score used 1,735.0 tokens per step; always-send reached 0.7867 accuracy at 3,235.4 tokens.

Table 23: **Selector sensitivity to calibration size.** Each column averages 20 instance-disjoint calibration/test splits.

Calibration fraction	5%	10%	20%	30%	40%
Mean accuracy	0.7368	0.7415	0.7425	0.7452	0.7458
Cost / always-send	0.5358	0.5366	0.5356	0.5372	0.5386

Learned routing comparison. The receiver-relative selector achieved the highest accuracy minus token cost among the evaluated routing policies (Table 24). All policies used `skip`/`raw`/`compress` and the same 20 instance-disjoint calibration/test partitions described above. The learned routers used receiver-local input length and step index as test features. The selector reached 0.7425 accuracy at 1,732.9 tokens per step; always-send reached 0.7917 accuracy but used 3,227.0 tokens. Utilities apply the same 10^{-4} cost weight.

Table 24: **ContextBench routing policies on a shared action set.** Utility is accuracy minus 10^{-4} times mean tokens.

Policy	Accuracy	Tokens/step	Utility
Receiver-relative selector	0.7425	1,732.9	0.5692
Always raw	0.7517	2,151.7	0.5365
Multinomial logistic (always skip)	0.6525	1,300.0	0.5225
Gradient boosted	0.6617	1,425.3	0.5191
Learned centroid	0.7292	2,181.5	0.5110
Always send	0.7917	3,227.0	0.4690

Reported MasRouter configuration. The MasRouter comparison matches sender and receiver identities, receiver prompt, evaluation support, split seeds, message budget, decoding, and token accounting with the receiver-relative selector. MasRouter uses an always-send communication policy. Table 25 gives its recorded accuracy and cost.

Table 25: **Separately reported MasRouter measurements.** Token counts are per step; the utility column uses ContextBench accuracy and the 10^{-4} token weight.

	ContextBench			ToolSandbox	
Policy	Accuracy	Tokens	Utility	Accuracy	Tokens
MasRouter (always-send)	0.6317	3,227	0.3090	0.7056	3,085

Executed cost and latency. The pre-sender gate reduced the executed ToolSandbox path to one receiver generation. On the 36-step online support, it reduced recorded tokens by 70.2% and latency by 31.0% relative to always-send (Table 26). A five-seed endpoint comparison with 720 step-runs per policy reduced tokens by 57.9% and latency by 26.0%, with 711/720 correct SKIP outputs versus 710/720 for always-send. The online and endpoint supports differ; the online slice was harder for always-send.

Table 26: **ToolSandbox runtime measurements.** Recorded tokens include prompt and completion tokens on the executed path. The 36-step input-only counts in Table 21 are 581 for the selector and 2,474 for always-send. The endpoint study used 144 steps across five seeds per policy.

Policy	Step-runs	Accuracy	Tokens/step	Seconds/step
Pre-sender gate	36	1.000	919.4	6.60
Always send	36	0.8056	3,085.4	9.56
SKIP endpoint	720	0.9875	1,801.4	14.37
Always send	720	0.9861	4,282.4	19.42

Table 27: **Model calls for the gate and five-action selector.** The gate precedes candidate construction. Each post-gate decision generates and caches both message candidates, uses nine probes capped at four tokens, and executes one receiver generation. `escalate` uses the stronger receiver; other actions use the base receiver.

Action	Scorer probes	Sender calls	Receiver generations
`skip` through gate	0	0	1
`skip` after scoring	9	2	1
`raw`	9	2	1
`compress`	9	2	1
`refresh`	9	2	1
`escalate`	9	2	1

C.7 Conditional Propagation in Shared-State Tasks

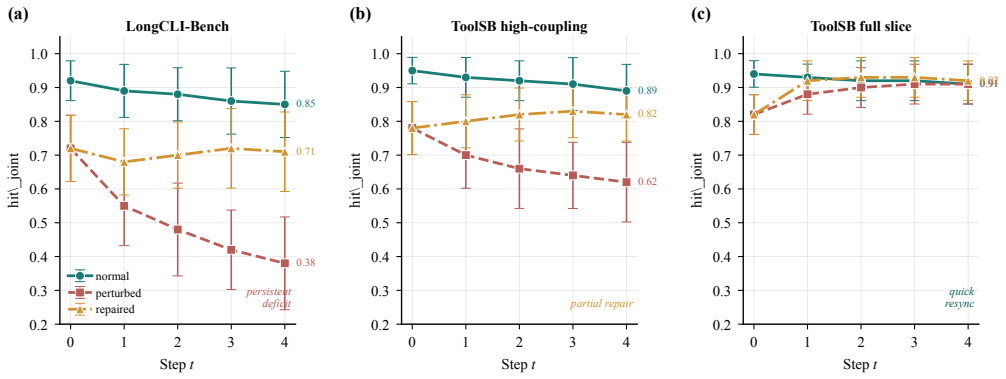

Figure 15: **Conditional propagation in shared-state tasks.** (a) LongCLI-Bench: perturbations produce a persistent downstream deficit, and repair only partially closes it. (b) A higher-coupling ToolSandbox condition: downstream effects are visible, and repair helps but does not fully erase the gap. (c) A broader ToolSandbox condition: the deficit is largely local, and the trajectory quickly resynchronizes.

These studies test the coupling dependence in Proposition 7. LongCLI and the higher-coupling ToolSandbox condition behave like positive-coupling settings: a local perturbation leaves a downstream deficit, and repair only partially closes it. The broader ToolSandbox condition behaves like a resynchronizing regime: the same intervention is mostly local, and the trajectory re-aligns once fresh observations arrive.

The fresh-observation ablation in Figure 16 makes the same point from the opposite direction. If a benchmark continually refreshes the local view, residual coupling is weak, and the perturbation dissipates quickly. If not, the same local error may persist for several steps. This is why the main-text selector includes a residual-risk term rather than optimizing one-step utility alone.

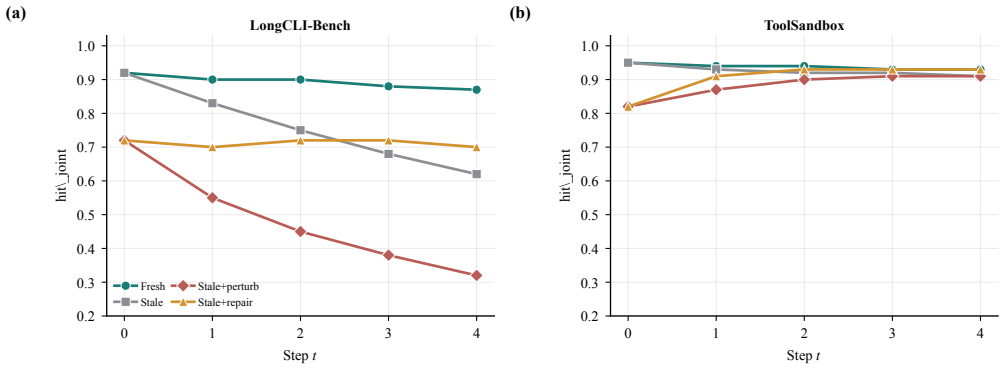

Figure 16: **Fresh-observation ablation.** (a) LongCLI-Bench: stale observations degrade downstream performance even without an injected perturbation, and targeted repair helps most under stale conditions. (b) ToolSandbox: fresh observations largely neutralize downstream perturbation effects in broader conditions, reinforcing the resynchronization interpretation.

D Illustrative Case Studies

The following traces are extracted from actual benchmark runs. Each case shows the receiver's local observation, the sender's message (when present), and the resulting action. These records illustrate the mechanisms in the main text.

Case D.1: Message overrides a correct local view (noise regime)

Benchmark: ToolSandbox **Task:** `find_temperature` **Step:** 1

Receiver's local observation:

```
Goal: What's the temperature here right now
Available tools: [search_weather_around_lat_lon]
Recent calls: [search_weather invoked in Step 0]
```

Recorded gold label: `run_test` / `search_weather_around_lat_lon`

Without sender (`receiver_only`): `run_test` / `search_weather_around_lat_lon` **Correct**

Sender message (80B):

```
{"action_type": "read", "target_artifact": "messages",
"rationale": "The weather tool was already invoked in Step 0;
now we must read the response, which should be stored in
messages as the tool's output."}
```

With sender (`ma_seq`): `read` / `messages` **Wrong**

Mechanism: The receiver-only output names the weather tool recorded as the target. The sender instead directs the receiver to `messages`, and the receiver adopts that target. This artifact mismatch identifies the harmful redirection independently of the trace's literal `run_test` action label.

Case D.2: Sender provides cross-module context the receiver lacks (helpful coordination)

Benchmark: ContextBench **Task:** Django cascade deletion bug **Step:** 0

Bug: `UnicodeDecodeError` during `.delete()` because cascade collection fetches fields with invalid UTF-8.

Without sender (`receiver_only`):

Action: `edit` / `django/db/models/base.py` **Wrong file**

The receiver edits `base.py`, a plausible module that does not contain the cascade logic.

Sender message (80B):

```
{"action_type": "edit",
"target_artifact": "django/db/models/deletion.py",
"rationale": "The UnicodeDecodeError occurs because .delete()
fetches unnecessary fields (e.g., text_log_error.line) during
cascade collection. We must modify Collector.collect() to only
retrieve fields required for deletion."}
```

With sender (`ma_seq`):

Action: `edit` / `django/db/models/deletion.py` **Correct**

Mechanism: The receiver's local view does not reveal which module implements cascade collection. The sender, with access to the full repository context, identifies the correct file (`deletion.py`) and the specific function (`Collector.collect`). This is the helpful-coordination regime: the message transfers cross-module knowledge that the receiver cannot derive from its local observation alone.

Case D.3: Same message and different action compliance (conversion bottleneck)

Benchmark: ContextBench **Task:** Astropy mask propagation bug **Step:** 0
Protocol: Same-message replay (frozen 80B sender output)

Frozen sender message:

```
{"action_type": "edit",
"target_artifact": "astropy/nddata/mixins/ndarithmetic.py",
"rationale": "Mask propagation fails when one operand has no
mask (None) and handle_mask=np.bitwise_or is used. Must
pre-process masks to use zeros for missing masks."}
```

Qwen-9B receiver output:

```
{"action_type": "edit",
"target_artifact": "astropy/.../ndarithmetic.py"}
```

Correct

GLM-5 receiver output:

```
{"action_type": "read",
"target_artifact": "astropy/.../ndarithmetic.py",
"rationale": "Need to examine the current mask handling
implementation before making the fix."}
```

Wrong action

Decomposition:

	Qwen-9B	GLM-5
Artifact correct (`hit_artifact`)	✓	✓
Action correct (`hit_action`)	✓	✗
Joint correct (`hit_joint`)	✓	✗

Mechanism: Both receivers process the identical frozen message. Both identify the correct target file. But GLM-5 emits `read` instead of `edit`. The information was recovered at the artifact level, but the action did not close correctly. This is why the fixed-message bottleneck in the main text appears as a conversion loss rather than as a failure of raw content recovery.

E Scope and Limitations

Our account is step-level and protocol-relative: the framework makes a sharp prediction at this granularity and isolates the message-conditioning effect from trajectory-level confounders. Extending to end-to-end task success requires modeling cross-step error correlation; a natural extension would estimate fresh discrepancies and carry coefficients separately rather than folding them into one-step contrasts. The selector is analysis-guided; its gains are strongest when closure failure or protocol tax dominates, and richer online estimators, together with receiver-side adaptation, will extend its reach to other regimes. Paired MA, RO, and SF contrasts measure deployed protocol effects, and instance-clustered intervals quantify their uncertainty. The theoretical gain and tax depend additionally on the bounded decoder class; empirical calibration offsets combine these effects. Shared-state results hold when trajectory coupling is non-trivial; in loosely coupled settings, trajectories resynchronize regardless of intervention, a direct consequence of the dynamic framework.

Broader impacts. Reducing unnecessary sender calls can lower inference cost and latency, making multi-agent assistance more accessible. The same communication mechanisms can spread incorrect directives or expose sensitive context to additional model services. Our observed target overrides and action-closure failures identify concrete failure modes for deployments. Applications should limit shared context to task-relevant information and validate consequential actions against the user's request and available tool state. Lower communication cost also makes automated workflows easier to scale, including workflows used for harmful purposes; efficiency and task accuracy therefore need to be assessed alongside the intended use.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state the step-level, receiver-relative framework, its formal results, and the selector; Sections 2–4 develop and test them, and Appendix E states the scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Appendix E, referenced from Section 5, discusses the step-level scope, the extension to end-to-end task success, the regimes where the analysis-guided selector helps most, and the dependence of gain and tax on the bounded decoder class.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Each result in Sections 2 and 3 states its assumptions locally (log-loss, proper Bayes-sufficient compression, bottleneck factorization, single crossing, shared carry sequences), and Appendix B gives the complete proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4.1 and Appendix C specify the benchmarks, sender and receiver models, message budget, paired contrasts, scorer targets, replay protocol, and selector calibration; Table 6 records the support used by each numerical family.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Code and the stored per-step records are released at `https://github.com/Yydc/Communication`, with slice manifests and scripts that recompute the reported table cells from the records; the repository README lists the components it does not include, such as the full benchmark-running harness.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The work is inference-only; Section 4.1 gives the default models, message budget, contrasts, and metrics, and Appendices C and C.6 give per-benchmark supports, decoding settings, calibration splits, and selector weights.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: Table 8 reports instance-clustered percentile bootstrap 95% intervals (5,000 resamples) and exact McNemar tests for the paired effects; the OpenAI extension reports Wilson 95% intervals (Tables 18 and 19), and the sender threshold reports a bootstrap interval (Section 4.5).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments are inference-only with no model training; open-weight Qwen models were served on a 2×NVIDIA H200 node, OpenAI models were queried through the API at a cumulative cost of approximately 300 USD, and Appendix C reports the evaluated step counts that determine reproduction cost.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper evaluates communication protocols on existing public benchmarks through model APIs and open-weight models; it involves no human subjects, deception, private data, or high-risk artifacts.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The broader-impacts paragraph in Appendix E discusses lower inference cost and latency as well as the risks of propagating incorrect directives, exposing sensitive context, and scaling harmful workflows, together with mitigations.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases no pretrained model, generative system, or scraped dataset; the released records are model outputs on existing public benchmarks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The six benchmark families are cited at first use in Section 4.1 and used under their published terms for research; models are identified by their public names and accessed through their providers' APIs or open-weight releases under the corresponding terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released code and per-step records are documented in the repository README and data card and are distributed under the MIT License.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The work does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve human subjects, so IRB approval is not required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are the object of study: the experiments measure communication between sender and receiver LLMs from the Qwen, OpenAI, DeepSeek, Llama, GLM, and MiniMax families, whose identities, decoding settings, and prompts are reported in Section 4.1 and Appendix C.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.