

DuplexAgent-RSI: Recursive Harness Improvement for Full-Duplex Voice Agent Collaboration

Yingda Shen* Yuxiang Wang* Kunyu Feng* Qinke Ni* Jiaqi Li*[†]
Minghao Hsu* Junan Zhang* Dekun Chen* Yutong Bian*[†] Zhizheng Wu*[†]

* The Chinese University of Hong Kong, Shenzhen

[†] Amphion Technology Co., Ltd

[Project Page](#) [GitHub](#)

Abstract

Voice agents are converging on a collaboration pattern: a full-duplex interaction model stays on the live channel as the entry to the conversation, while search, reasoning, and coding are handled through asynchronous delegation. A duplex model supports continuous listening and speaking, but complex reasoning and tool use may exceed its capabilities. A coding agent can plan and execute extended tasks, but its sequential interface is a poor fit for live conversation. Combining them requires a harness that coordinates task acceptance, progress, cancellation, replacement, and result delivery while keeping the conversation responsive. Existing harnesses often rely on coupled heuristics, making them difficult to improve systematically from evidence. We present DuplexAgent, a full-duplex collaboration system whose harness expresses this workflow as six editable modules, and *Duplex-Harness-RSI*, a closed loop that revises them from interaction traces. A simulator automatically generates timed test conversations, runs the system, and produces failure traces that identify the collaboration modules requiring repair. Reasoning LLMs and coding agents in the delegation pool also serve the improvement loop: the Exam Planner selects the next tests from observed weaknesses and the repair archive, and the Harness Editor proposes targeted module changes. The capabilities that serve the user thus also improve the system’s coordination. Experiments on intelligence, agentic, and duplex benchmarks show that DuplexAgent combines continuous interaction with difficult reasoning and complex task execution, achieving stronger spoken-knowledge and executable-tool scores than the compared delegated systems while maintaining strong interruption response. A harness ablation further shows that this modular, verifiable loop outperforms the initial harness and repeated editing that lacks its diagnosis and repair archive.



Figure 1: Benchmark results for DuplexAgent (blue), compared with other collaboration systems (gray). Higher is better on every panel. Intelligence reports MMSU accuracy, agentic tool use reports BFCL parallel-multiple and FDB-v3 Pass@1, and duplex reports FDB-v1 interrupt TOR. DuplexAgent is stronger on all three capabilities. All collaboration systems use the same delegation LLM. See Tables 4 and 5.

1 Introduction

As language-model agents move from answering questions to carrying out work such as search, tool use, multi-step plans, and code edits (Yao et al., 2023), users need to supervise that work as it proceeds: they refine the goal, ask for status, interrupt an answer, or start a second task before the first has finished. Real-time speech provides a low-friction interface for this supervision, but it replaces orderly text turns with overlap, backchannels, asides, and changes of mind. Continuous interaction and complex task completion are usually handled by different model families. A full-duplex interaction model keeps listening and speaking on the live timeline, supporting turn-taking, barge-in, and backchannels (Défossez et al., 2024; Wang et al., 2024). A long search, difficult reasoning pass, or repository edit, however, requires deliberate work beyond the conversation. Tool-using and coding agents can perform that work (Yao et al., 2023), but a sequential agent cannot remain attentive to the live channel while it runs.

Recent systems bridge this division by keeping the interaction model on the live channel and assigning demanding work to a delegation pool through asynchronous handoff (Deng et al., 2026; Ant Group (Venus Team) and Tsinghua University, 2026). This combines continuous interaction with the reasoning ability of LLMs and the long-horizon execution of coding agents, but does not determine how an active task should be updated, cancelled, or returned. Suppose a user requests last night’s match result, then says “actually, the women’s match” while the first search is running, and begins another utterance just as the original result returns. The system must revise the active task, reject the obsolete result, and wait for a valid speaking opportunity. A fixed coordinator may instead launch a second independent search, speak the first result over the user, or later cancel the wrong task. Existing systems implement useful instances of handoff, acknowledgement, and delivery control (Deng et al., 2026; Ant Group (Venus Team) and Tsinghua University, 2026; Orantqing et al., 2026; Lin et al., 2025c), but these decisions are generally coupled in system-specific heuristics. When the same failure recurs, a trace may not clearly identify whether delegation, task interpretation, background execution, or delivery should change. A bounded repair is then difficult to isolate, evaluate, and transfer.

We present *DuplexAgent*, a full-duplex collaboration system whose harness makes this control explicit in six editable modules: Policy decides whether a turn is answered, delegated, or treated as task control; Intake interprets the request and the live task it concerns; Hold maintains the conversation while work is pending; Worker selects the delegate and the result contract; Deliver controls when and how a result is returned; and Memory preserves the context shared by the conversation and the background work. The interaction model remains the conversational entry, while a *delegation pool* of reasoning LLMs and coding agents carries delegated work. A common harness coordinates their exchange through model-specific adapters, supporting both open-weight and hosted interaction models and one or several delegates.

Separating these modules gives each failure a place to be investigated and provides the basis for *Duplex-Harness-RSI*. In the match-query example, the trace can distinguish a task that was not updated from an obsolete result that was delivered. The diagnosis points to Intake, Deliver, or a repair involving both, rather than a change to the whole harness. The proposed change is then specific enough to evaluate against the failed conversation and other tests.

Each test is a timed conversation assembled from a request type and an interaction pattern. It states the expected behaviour, and the trace attributes a failure to the responsible module. A reasoning LLM from the delegation pool acts as the *Exam Planner*: it reads attributed failures and the repair archive, then adjusts the next exam. The simulator generates the conversations. A coding agent from the same pool acts as the *Harness Editor* and proposes changes to the modules named by the diagnosis. Evaluation compares each candidate with the current harness on paired runs and checks a disjoint protection set before retention.

Section 2 reviews related work. Section 3 presents the six modules, test generation, failure attribution, and the improvement loop. Section 4 describes the session engine, the adapters, and the RSI configuration used in the experiments. Section 5 reports spoken knowledge, tool use, duplex timing, and harness improvement with two interaction models. Figure 2 summarises the system and improvement loop.

Contributions.

1. *DuplexAgent*. A complete collaboration system that attaches a range of full-duplex interaction models as the conversational entry and a range of reasoning LLMs and coding agents as the delegation pool, and that expresses the collaboration workflow as six editable harness modules.

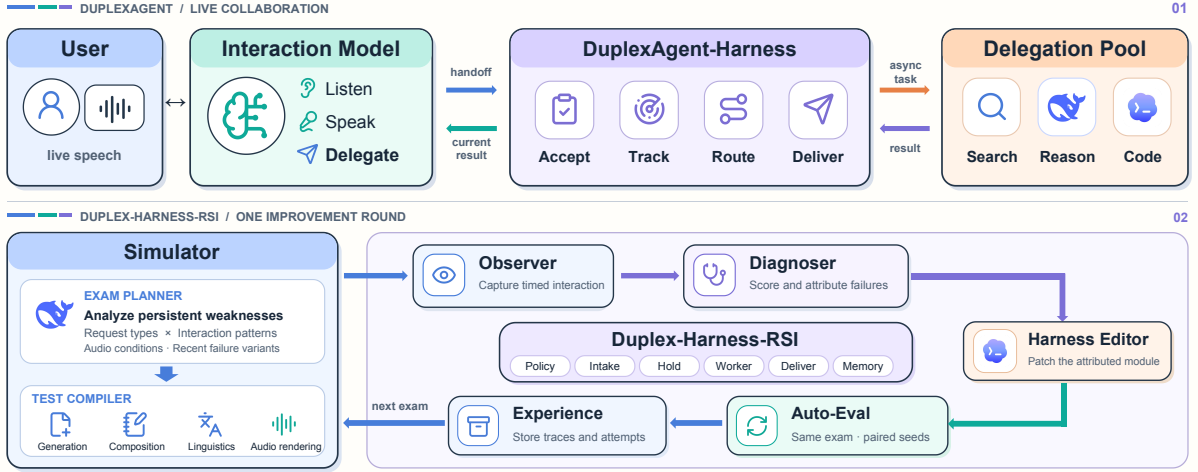


Figure 2: Schematic of DuplexAgent and Duplex-Harness-RSI.

2. *Duplex-Harness-RSI*. A recursive improvement loop that reuses the delegation pool. A reasoning LLM acts as Exam Planner, selecting tests from attributed failures and the repair archive. A coding agent acts as Harness Editor, revising the relevant modules. Simulation and verifiable evaluation connect these revisions to observed outcomes, so the capabilities that serve the user also improve the harness.
3. *A reusable collaboration simulator*. An automatic pipeline for content generation, temporal composition, speech rendering, and evaluation, with coverage organized by nineteen request types and twenty-two interaction patterns. It supplies the tests and failure traces used by Duplex-Harness-RSI and can generate training or evaluation conversations for other interaction systems.

2 Related work

Full-duplex models and delegated agents. Full-duplex speech research has explored both end-to-end and modular routes. Models such as Moshi (Défossez et al., 2024), Freeze-Omni (Wang et al., 2024), SALMONN-omni (Yu et al., 2024), and MiniCPM-o 4.5 (Cui et al., 2026) address streaming speech understanding and generation, while modular systems study controllable components, acoustic-semantic separation, and voice or role conditioning (Liu et al., 2025; 2026; Roy et al., 2026; Chen et al., 2025). A second line of work connects the live channel to background computation: MoshiRAG (Chien et al., 2026) adds asynchronous retrieval, Qwen-Audio-Agent (Deng et al., 2026) delegates background work, Realtime-Venus (Ant Group (Venus Team) and Tsinghua University, 2026) coordinates asynchronous returns, and Gander (Orantqing et al., 2026) connects interaction to long-horizon execution. Recent systems and benchmarks also emphasize multi-round state and realistic overlap rather than isolated turns (Lin et al., 2025a; He et al., 2026; Wang et al., 2026). DuplexAgent builds on this architectural direction and studies how its coordination mechanisms can be diagnosed and revised from experience.

Evaluation of spoken agents. VoiceBench (Chen et al., 2024) evaluates voice assistants across spoken tasks and input conditions. Full-Duplex-Bench (Lin et al., 2025b) measures turn-taking, with later versions extending overlap handling and tool use (Lin et al., 2025a; 2026). BFCL (Patil et al., 2025) evaluates structured function calling, while τ -Voice (Ray et al., 2026) studies voice agents on service tasks. These benchmarks provide external measures of knowledge, tool execution, and interaction. Our simulator complements them with explicit task transitions and module-attributed failures for harness revision. External benchmark scores and simulator collaboration scores answer different questions and are reported separately.

Recursive harness improvement. Automated agent design and harness optimization treat prompts, tools, or executable coordination as objects that can be searched and revised (Hu et al., 2025; Lee et al., 2026). Self-harness and recursive improvement systems extend this idea by using traces, scores, and prior attempts to guide later changes (Zhang et al., 2026; Tan et al., 2026). Related work also explores modular repair and training-time optimization of agent workflows (Wu et al., 2026; Luo et al., 2025). Duplex-Harness-RSI applies harness revision to live spoken collaboration, where correctness includes task identity, conversational continuity, and the timing of delivery. Its Exam Planner adapts the test distribution, and its Harness Editor changes the harness modules under a fixed evaluation boundary.

3 Method

A deployed DuplexAgent system is

$$S = (F, B, H), \quad (1)$$

where F is the full-duplex interaction model, B is the delegation pool, and H is the collaboration harness. This decomposition separates the models that produce an answer from the harness that decides how that answer enters the live conversation. F listens and speaks on the user-facing channel, members of B carry out longer jobs asynchronously, and H maintains the identity, timing, and context of the work between them. Duplex-Harness-RSI improves H while keeping the attached models fixed. Its Exam Planner and Harness Editor are reasoning and coding roles drawn from the same pool B that serves user requests, so the improvement loop is part of the deployed system.

3.1 The collaboration harness

The harness exists because a delegated result is useful only when it still answers the user’s current request and can be delivered without breaking the conversation. Figure 3 makes this requirement concrete: the user changes the game while the first task is running, so the system must update the active goal, retire the obsolete work, and wait for an appropriate moment to speak.

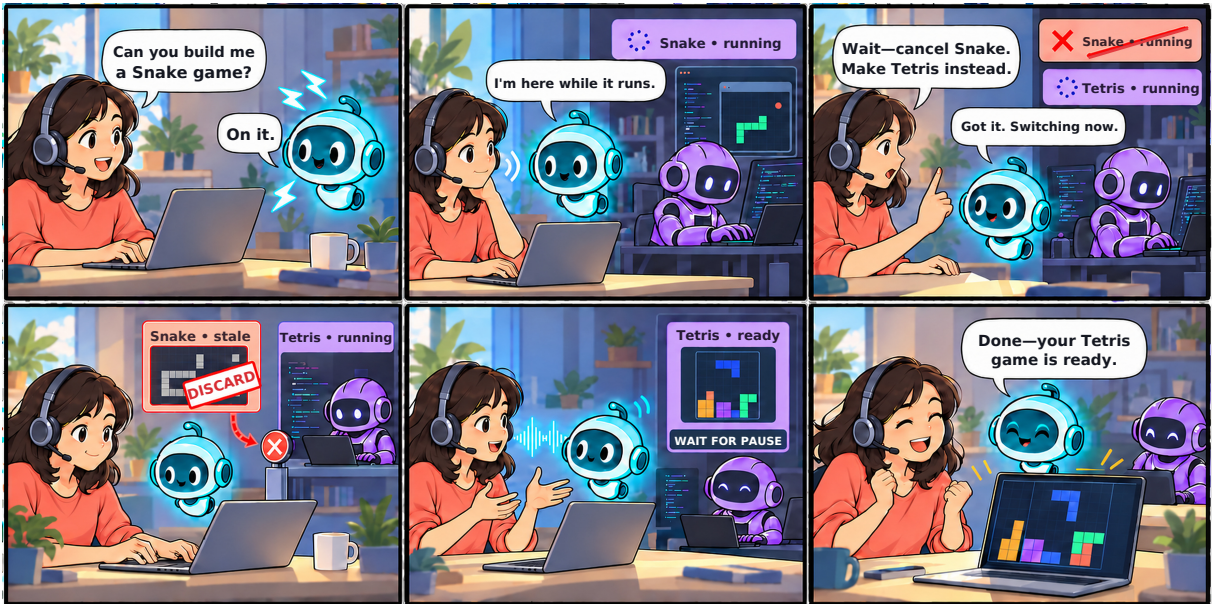


Figure 3: A full-duplex collaboration session. The user requests Snake, then cancels it and asks for Tetris while work is running. The obsolete Snake result is discarded. The current Tetris result returns through the interaction model after the user pauses.

The harness comprises

$$H = (\text{Policy}, \text{Intake}, \text{Hold}, \text{Worker}, \text{Deliver}, \text{Memory}). \quad (2)$$

Table 1 defines the decision owned by each module and the surface on which it can be changed. Policy chooses the route: answer, delegate, or task control. Intake identifies the operation and the live task it concerns. Hold and Deliver protect the conversational rhythm. Worker governs the handoff to a delegate, and Memory keeps the relevant context available to both the live conversation and the background work. Each module has a clear input, decision, and observable effect, so a failure can be attributed to a bounded part of the harness and a later revision can be checked against the same behaviour. An interaction-model adapter carries these decisions to the attached model without changing their meaning.

The six modules are supported by two pieces of shared state. A session engine keeps a task ledger with the task identity, version, status, and delivery history. A task can be accepted, run, completed, delivered, cancelled, or replaced; a late delegate return is still recorded, but it cannot make an obsolete task current again. Memory holds the conversational context needed to interpret the latest request and to give a delegate enough information to act. The adapter maps the harness decisions to the attached interaction model and delegate interfaces. Thus task control has a precise effect: it changes the ledger, and the ledger determines which background work remains eligible for delivery.

Table 1: Six editable modules of the DuplexAgent harness.

Module	Collaboration decision	Editable surface
Policy	Answer, delegate, or task control	Instructions, delegation schema, routing examples
Intake	Operation expressed, and which live task	Parsing, control cues, task relations, resolution rules
Hold	What the user hears while a task is pending	Receipts, progress timing, control replies
Worker	Delegate, context, and result contract	Routing, task instructions, result schemas, executable overrides
Deliver	When and how a current result returns	Speaking window, waiting, success or failure rendering
Memory	Context shared by conversation and background work	Turn assembly, retention, compaction, reset recovery

The Snake-to-Tetris session in Figure 3 follows one task through this protocol. Policy decides that implementation should be delegated and Memory retains the goal; Intake opens the task, Hold acknowledges it, and Worker sends the request and relevant context to a coding agent. When the user changes the goal, Intake resolves the cancellation and replacement against the ledger, Memory updates the context, and Hold acknowledges the change. A late Snake result is recorded but cannot be delivered. When Tetris becomes ready, Deliver waits for a speaking window and returns the current result through the interaction model. The six modules therefore cooperate over time, rather than acting as independent stages.

3.2 Composing test conversations

The simulator turns the harness contract into executable conversations. It separates what the user wants from what happens while that request is being handled. A *request type* specifies the requested capability and expected initial route; an *interaction pattern* specifies events such as refinement, cancellation, overlap, or delayed completion. Linguistic and acoustic variation are added after these semantic expectations are fixed. The same control operation can therefore be tested across different capabilities while preserving the behaviour that the evaluator expects.

The bank contains nineteen request types. For exposition, we group their content into *Conversational* requests such as social exchange and preferences; *Information* requests such as factual or live-information queries; *Reasoning* requests such as calculation and structured explanation; and *Action* requests such as tool use and long-horizon execution. These groups describe content, not a blanket delegation rule. A familiar fact may be answered directly, a casual request may need retrieval, and an ambiguous or unsupported request may permit either a direct response or delegation. Each item records its allowed route: direct, delegate, or either.

The twenty-two patterns cover *Nominal* completion; *Task control* including refinement, replacement, cancellation, and status; *Concurrent work* with new requests before existing work finishes; *Full-duplex overlap* including barge-in, result arrival during user speech, and backchannels; and *Continuity and recovery* including delay, failure, follow-up, and reset. Only compatible request-pattern pairs are composed. The grid organizes coverage without requiring every cell to be instantiated.

With a compatible pair chosen, the pipeline first generates a request, reference answer when one is available, and natural follow-up or control turns. Quality checks test semantic fit, naturalness, answer consistency, and duplication; arithmetic references can be computed directly. Items are assigned to development, protection, and sealed sets before temporal composition, so alternate forms of one case cannot cross an evaluation boundary. A fixed core of development cases is repeated in every exam so that later rounds can be compared on the same cases. A deterministic composer then places the turns on a timeline, specifies delegate delays or failures, records the required task transitions, and defines the speaking windows in which a result is admissible. The resulting expectations are executable: answer checks may use exact, contains, regular-expression, option-letter, or semantic criteria, while tasks without a stable answer are judged by their interaction behaviour.

Only after these expectations are fixed do we vary the spoken surface. Hesitations, paraphrases, distractors, spoken numbers, and spelled identifiers change how a request sounds without changing what the system must do. Speech rendering and acoustic augmentation then synthesize the turns and add ambient noise or channel effects. Speaker, style, language, and pace can vary across the resulting audio, while clean and augmented versions retain the same semantics. Live evaluation streams each waveform on a realtime clock.

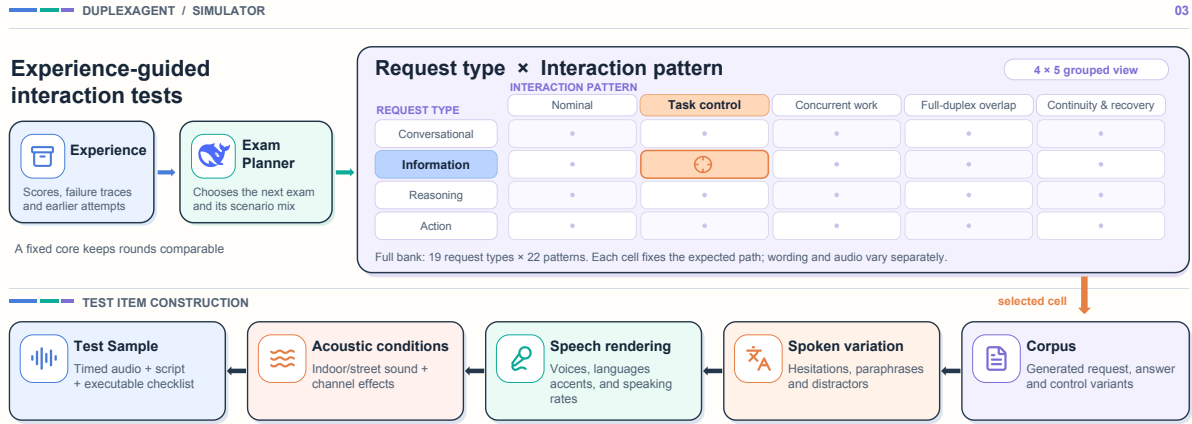


Figure 4: Test-conversation construction. The Exam Planner selects coverage over request types, interaction patterns, and input conditions. Content generation and deterministic temporal composition produce a corpus with explicit expectations. Speech rendering and acoustic augmentation turn it into timed audio input.

For the running example, an Action request is combined with replacement and result-arrival overlap. The composer schedules the goal change before Snake completes and requires only Tetris to be returned. Input variation may make cancellation hesitant or acoustically degraded without changing the requirement. References stay in the evaluator and never enter the interaction model’s prompt.

3.3 Trace-based diagnosis and collaboration score

A simulator case is useful for Duplex-Harness-RSI only when a failed run can be explained. In Figure 2, the Observer records the behaviour of a run as a trace: speech activity, transcripts, delegation signals, module decisions, delegate returns, task transitions, and delivery events. The Diagnoser aligns that trace with the expected behaviour and assigns the failure signature. It also groups related outcomes and selects representative traces for the Exam Planner and the Harness Editor.

Each failed run receives a failure signature

$$\varphi(\tau) = (\text{cause}, \text{module}, \text{pattern}). \quad (3)$$

The Diagnoser first identifies failure flags and then selects one terminal cause in a fixed priority order. A default mapping assigns that cause to a module, and intermediate decisions and returned payloads can revise the assignment. A wrong spoken answer belongs to Deliver when the delegate payload was correct, and a missing result belongs to Worker when execution failed. Several flags may be present on one run; the terminal cause is the target that later rounds compare.

In a failed Snake-to-Tetris session, an obsolete Snake result may be spoken during the user’s new request. The trace distinguishes incorrect task-control resolution from delivery of a result already marked obsolete. This determines whether Intake, Deliver, or a coupled repair should change; the audible error alone is insufficient.

The collaboration score summarizes these attributions. A conversation succeeds when it satisfies the

Table 2: Default failure attribution, refined using intermediate decisions and returned payloads.

Failure family	Default attribution	Trace evidence
Perception	Outside harness	Requested turn not recovered well enough for the harness to act
Routing	Policy	Missed or extra delegation, or a reply to non-directed speech
Interpretation	Intake	Malformed request, or control applied to the wrong task
Continuity	Hold	Receipts conflict with speech, handoffs repeat, or pending work lacks progress
Execution	Worker	Execution fails, content is wrong, or the result contract is violated
Delivery	Deliver	Missing, stale, duplicated, late, unspoken, spoken over the user, or failure shown as success
Context	Memory	Question content or established context lost on delegation, follow-up, or reset

item’s expected behaviour. Runs in which the interaction model fails to perceive the input are excluded, because that failure is outside the harness revision. We report

$$Q = \text{percentage of successes, excluding perception failures.} \quad (4)$$

We compute Q on the fixed core and on the protection set, which keep the same cases across rounds. Q measures whether the coordination is correct; the failure signature and the trace explain the score.

3.4 Duplex-Harness-RSI

Duplex-Harness-RSI uses the diagnosis from one exam to decide what the next exam should reveal. The Exam Planner reads attributed failures and the repair archive, then shifts emphasis among request types, interaction patterns, and input conditions. It keeps the fixed core and representative coverage, and it bounds how far the sampling weights may move. Each exam adds a weighted development sample and may include a limited number of fresh variants derived from recent failures. These variants are archived with the round rather than added to the bank permanently. Cases that retest an accepted repair stay in the plan, and repeatedly unsuccessful targets do not dominate the next exam. The planner changes what is tested, and leaves the scoring rules and the protection set as they are.

The planned exam is first run with the incumbent, the harness version currently in use. Its trace, score, failure signature, representative cases, and the visible part of the repair archive form the evidence given to the Harness Editor. The editor must turn that evidence into a testable hypothesis: it identifies the decision that produced the failure, proposes a coherent change on the declared editable surfaces, and records what the change is expected to improve. The simulator and the evaluator stay fixed, so the candidate is judged by the behaviour it changes. The excerpts below state each role’s information and obligations, taken from the released contracts.

Exam Planner: select the next exam

Role. Choose what the next exam should test.

Inputs. Read the request-type and interaction-pattern taxonomy, available test inventory, per-cell outcomes and attributed failures, the previous exam plan and its results, and the history of accepted and rejected repairs.

Tasks. Assess whether the previous emphasis exposed useful failures. Set sampling weights over request types, interaction patterns, and input conditions. Allocate fresh variants to recent failures the harness can address, and add cases that protect newly repaired behaviours.

Requirements. Preserve the fixed core, maintain representative coverage, and respect the limit on added cases. Use valid taxonomy combinations. Rotate away from repeatedly unsuccessful targets; do not change scoring rules or the protection set.

Output. Return one structured exam plan containing the sampling emphasis, coverage constraints, any added cases, the previous evaluation summary, and a short rationale for the allocation.

Harness Editor: propose a targeted repair

Role. Improve the collaboration harness from the supplied evidence, using only its declared editable surfaces. **Inputs.** Read the incumbent harness, module files and adapter contracts, current metrics, actionable failure signatures with representative traces, and visible prior edits, rationales, and evaluation verdicts.

Tasks. Locate the decision that produced each failure and group related cases by mechanism. Check which changes have already been tried. State a testable hypothesis and create candidate harness copies with coherent repairs, possibly spanning several modules. Use visible evaluation feedback to refine rejected proposals.

Requirements. Preserve task validity and continuous interaction. Edit only declared surfaces in candidate copies; keep the incumbent, evaluator, and simulator unchanged. Do not inspect the protection set or insert scenario-specific answers.

Output. For each candidate, return the revised files and a `rationale.json` describing the primary module, failure analysis, hypothesis, change summary, expected gain, risks, and addressed signatures. Fixed evaluation determines retention.

The evaluator checks six invariants: delegation originates from the interaction model; the harness does not mute or interrupt its speech; acceptance precedes completion; only a current result may be delivered, at most once; results return through the interaction model; and test reference answers do not enter its prompt. The session engine, adapter contracts, simulator, and scoring rules are outside the editor’s write surfaces. These constraints keep the meaning of a successful revision fixed while the harness modules change.

The incumbent and the candidate are evaluated on the same exam. A *win* is an incumbent failure that the candidate makes acceptable; a *loss* is the reverse. A run that fails only because the interaction model did

not perceive the input is not counted as a harness loss. This is the same exclusion used in Q .

The development gate requires more wins than losses, preserved invariants, and cause regressions within a fixed tolerance. It reports which signatures changed, so the editor can compare its hypothesis with the measured effect. Candidates are ranked by their paired wins and losses. A candidate that passes is checked on the protection set in separate executions. The protection gate requires at least as many wins as losses and preserved invariants. Protection traces stay hidden from the editor. A rejected candidate does not replace the incumbent.

The repair archive keeps exam plans, traces, candidate changes, rationales, and visible verdicts for both accepted and rejected attempts. In Figure 2, experience is this archive carried into the next round. The Exam Planner and the Harness Editor use it to avoid repeating unsuccessful changes and to build on repairs that have already helped.

Let g_0 denote the campaign’s initial harness and g_k its k th retained version. A round checkpoint r_t reports the incumbent after t completed improvement rounds. Rejection keeps the current version, and a repair that touches several modules still creates one retained version.

The loop plans an exam, runs and diagnoses it, edits the harness, applies the development and protection gates, and writes the repair archive. The reasoning and coding capabilities come from the delegation pool already used by DuplexAgent. The interfaces, test expectations, and gates stay fixed while that pool revises the harness.

4 Implementation

The session engine consumes events for user and agent speech, partial and final transcripts, and delegation or control signals. It sends instructions, tool definitions, context updates, and returned results through an adapter. Each adapter declares its capabilities, including native function calling, textual delegation markers, context injection, and the treatment of interim tool outputs. The harness uses these declarations to choose a compatible receipt and delivery path. This separates harness decisions from the transport details of an open-weight or hosted realtime interaction model.

Worker selects a delegate according to the requested capability and supplies the current goal, relevant context, and a result contract. In the reported setup, DeepSeek-v4-flash handles reasoning and general delegated work, while Codex supplies coding capability. During Duplex-Harness-RSI, DeepSeek-v4-flash is reused as the Exam Planner and Codex as the Harness Editor. The planner runs offline, and the editor runs in an isolated workspace that contains the incumbent harness and the visible part of the repair archive.

A harness version consists of a manifest and its declared module files, including instructions, configuration, and executable overrides. The evaluator derives a patch from the editor’s filesystem changes and rejects undeclared paths. The editor cannot change scoring rules, import the simulator into the harness, or insert scenario-specific answers.

The experiments use the default configuration of the released implementation, summarized in Table 3. Each round runs the live interaction model and the delegation pool. The six harness modules are revised, and the model weights stay fixed.

Table 3: Configuration used in the RSI experiments.

Component	Experimental setting
Content bank	Content split before temporal composition into development, protection, and sealed sets (60/20/20).
Exam Planner	DeepSeek-v4-flash; bounded weights over request types, interaction patterns, and input conditions, with request-coverage floors.
Adaptive exam	Fixed core share 40%; fresh-item budget of 30 per round, with weighted sampling of the remaining cases.
Harness Editor	Codex using gpt-5.6-sol; three candidates and up to two refinement attempts per round.
Development gate	More paired wins than losses, preserved collaboration invariants, and cause regressions within a fixed tolerance.
Protection gate	A separate protection set; at least as many paired wins as losses and preserved invariants.
Repair archive	Attributed outcomes, candidate changes, and visible verdicts for accepted and rejected attempts.

5 Experiments

We evaluate the complete system on external tasks and examine harness revision separately on simulator conversations. External comparisons measure the combined effect of delegation and the harness. The ablation then compares ways of revising that harness. Percentage differences below are stated in percentage points.

5.1 Evaluation setup

Spoken knowledge. We use OpenBookQA (OBQA) and Massive Multi-task Spoken Understanding (MMSU) from VoiceBench (Chen et al., 2024). Both present spoken multiple-choice questions. OBQA emphasizes elementary science, whereas MMSU spans a wider range of subjects. The response is parsed for an option letter and compared with the reference.

Executable tool use. The BFCL evaluation (Patil et al., 2025) uses spoken requests from five single-turn subsets in the v3 release: simple, multiple, parallel, parallel-multiple, and irrelevance. The first four vary the number of available functions and required calls; irrelevance tests whether the system abstains from calling an unsuitable function. Calls are scored against the benchmark’s function and argument requirements. We report each subset and their unweighted mean.

FDB-v3 (Lin et al., 2026) evaluates tool use under naturally disfluent speech. Selection F1 measures agreement between expected and selected calls. Argument accuracy uses strict character-level exact-string matching, and Pass@1 requires the correct tool set with every argument matching exactly.

Duplex timing. We use FDB-v1 (Lin et al., 2025b) pause handling, turn taking, and user interruption. Input is streamed on a realtime clock; output speech is aligned to the evaluation windows. Takeover rate (TOR) is the fraction of samples with a substantive response rather than silence or a brief backchannel. Lower pause TOR means fewer premature responses during a hesitation. Higher turn TOR and interrupt TOR mean the system responds more often after a completed turn or an interruption. We report the synthetic and Candor pause conditions separately.

Systems and comparison scope. Tables 4 and 5 include standalone interaction models and systems that add asynchronous delegation. MoshiRAG adds retrieval to Moshi. Gander connects an interaction model to long-horizon execution. Realtime-Venus-Omni is the Realtime-Venus system attached to Realtime-Omni-9B. Qwen-Audio-Agent and DuplexAgent are each run on VoiceChat and on Qwen-Realtime. Throughout the tables and figures, VoiceChat denotes NVIDIA-NemotronLabs-VoiceChat-11B (Balam et al., 2026) and Qwen-Realtime denotes Qwen-Audio-3.0-Realtime. In the Qwen-Realtime block, Qwen-Audio-Agent and DuplexAgent delegate reasoning to DeepSeek-v4-flash. Comparisons within a block hold the interaction model fixed. Comparisons across blocks also change the interaction model.

Simulator and improvement setup. DeepSeek-v4-flash generates test content and serves as the Exam Planner; Codex serves as the Harness Editor. Qwen3-TTS renders the composed conversations as speech, varying speaker, language, and speaking style, and acoustic augmentation adds noise and channel effects. The simulator follows Section 3.2. External benchmarks keep their own inputs.

VoiceChat and Qwen-Realtime are revised in separate campaigns with the configuration in Table 3. Model weights stay fixed. The fixed core is repeated in every exam so that collaboration can be compared across revisions. The exams used for editing also include adaptively chosen development cases and fresh variants.

5.2 External task performance

5.2.1 Executable tool use

Executable calls require a precise handoff. Table 4 separates two kinds of tool behaviour. A standalone interaction model that can bind the benchmark schema keeps a usable BFCL score: Realtime-Omni-9B averages 68.8, VoiceChat 53.5, and Qwen-Realtime 82.0. Moshi and MiniCPM-o 4.5 have no such binding, so their four calling subsets are 0.0 and only irrelevance, which rewards an empty call list, reaches 100.0. Adding a background delegate does not create that binding, and where the interaction model already emits calls, the added system can replace them with a description. MoshiRAG, Gander, Realtime-Venus-Omni, and Qwen-Audio-Agent therefore leave executable accuracy at or below the matching interaction model. The drop is largest for VoiceChat and Realtime-Omni-9B, the interaction models that were already calling.

Table 4: Executable tool use on BFCL and FDB-v3. DuplexAgent scores higher than the standalone interaction model and Qwen-Audio-Agent on both attachments. Scores are percentages. * denotes VoiceChat and † denotes Qwen-Realtime.

Method	Delegate	BFCL-v3						FDB-v3		
		Average	Simp.	Mult.	Para.	P-M	Irre.	F1	Argu.	Pass@1
Moshi	✗	20.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
MoshiRAG	✓	20.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
MiniCPM-o 4.5	✗	20.0	0.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0
Gander	✓	17.7	0.0	0.0	0.0	0.0	88.3	0.0	0.0	0.0
Realtime-Omni-9B	✗	68.8	74.0	57.0	71.0	53.0	89.2	7.0	4.2	2.0
Realtime-Venus-Omni	✓	46.5	48.5	41.0	35.0	22.0	85.8	5.1	3.3	1.0
VoiceChat	✗	53.5	54.5	60.0	36.0	22.0	95.0	60.2	26.2	16.0
Qwen-Audio-Agent*	✓	24.9	3.5	2.0	12.0	7.0	100.0	19.2	13.5	5.0
DuplexAgent*	✓	75.8	77.0	78.0	77.0	57.0	90.0	97.4	57.0	47.0
Qwen-Realtime	✗	82.0	88.0	86.0	85.0	66.0	85.0	87.0	47.8	39.0
Qwen-Audio-Agent†	✓	80.8	88.0	85.0	77.0	58.0	95.8	75.5	42.7	30.0
DuplexAgent†	✓	90.8	92.0	88.0	96.0	78.0	100.0	84.8	55.0	49.0

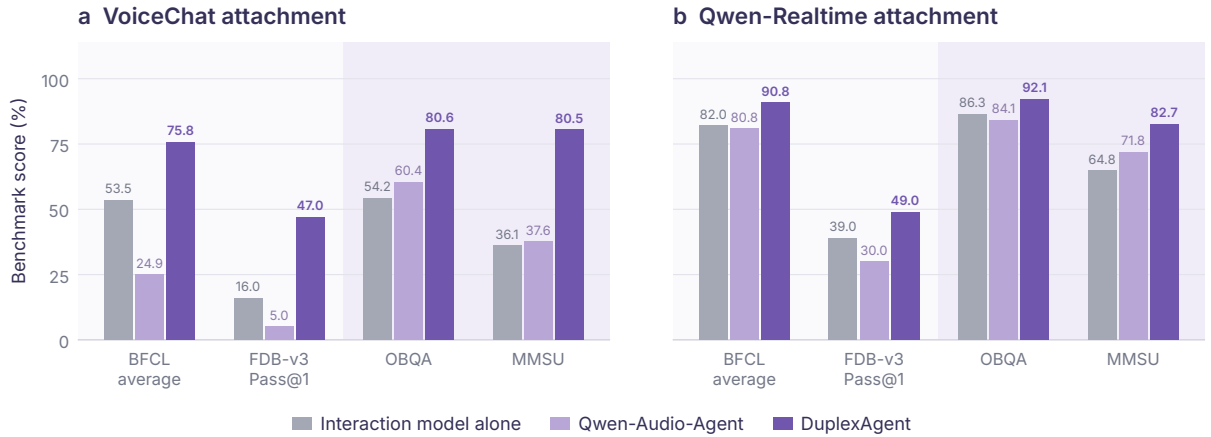


Figure 5: DuplexAgent scores above both its standalone interaction model and Qwen-Audio-Agent on the same attachment.

Those systems hand the delegate a description rather than a benchmark call. Gander’s interaction model opens work with task_start, task_send, and task_resolve; the argument of task_start is a short task title, and a coding agent then acts with its own file and command tools. Realtime-Venus-Omni writes the request as prose inside a <delegate> span and dispatches that sentence. Qwen-Audio-Agent answers in speech when it can, and otherwise calls spawn_thinking with a natural-language restatement of the objective. MoshiRAG retrieves passages for the spoken reply. A request such as “implement this feature” or “draft this section” can succeed under that contract, because the delegate chooses its own actions from a fuzzy objective. BFCL and FDB-v3 ask for something narrower. The scored object is the benchmark function name together with its required arguments: one call, several calls, or no call, and on FDB-v3 an exact argument string recovered from disfluent speech.

The recorded handoffs match that gap. Gander’s calls are task titles, so the BFCL calling subsets and all three FDB-v3 numbers stay at 0.0, and titles emitted on items that should abstain lower irrelevance from 100.0 to 88.3. Realtime-Omni-9B already fills the function on clean BFCL speech; FDB-v3 is the harder surface, and Realtime-Omni-9B withholds the call on nearly every item (Pass@1 2.0). Realtime-Venus-Omni then substitutes a sentence for the call the interaction model had produced. BFCL average falls

from 68.8 to 46.5, parallel from 71.0 to 35.0, and parallel-multiple from 53.0 to 22.0, because one prose span must be parsed back into several typed arguments. FDB-v3 Pass@1 stays at 1.0. Qwen-Audio-Agent on VoiceChat withholds calls the interaction model was making: simple accuracy falls from 54.5 to 3.5, FDB-v3 Pass@1 from 16.0 to 5.0, and irrelevance rises to 100.0. On Qwen-Realtime the interaction model already calls accurately, and the same restatement still trims parallel accuracy from 85.0 to 77.0, parallel-multiple from 66.0 to 58.0, and FDB-v3 Pass@1 from 39.0 to 30.0.

DuplexAgent keeps the decision to delegate separate from the content of the handoff. Policy leaves a direct reply on the interaction model and delegates the request that needs a tool or a harder computation. Worker receives the official schema and the arguments carried by the user turn, and Memory retains the identifiers and constraints that a title or a short restatement drops. Deliver returns only the current result, in a speaking window. Those rules are the harness after Duplex-Harness-RSI (Section 5.3), scored on the same waveforms, schemas, and checks as the other rows. On VoiceChat, BFCL average rises from 53.5 to 75.8 and FDB-v3 from 60.2 / 26.2 / 16.0 to 97.4 / 57.0 / 47.0. Irrelevance falls from 95.0 to 90.0: recovering required calls also admits some unnecessary ones. On Qwen-Realtime, BFCL average rises from 82.0 to 90.8, parallel-multiple from 66.0 to 78.0, and irrelevance from 85.0 to 100.0. Selection F1 moves from 87.0 to 84.8, while argument accuracy rises from 47.8 to 55.0 and Pass@1 from 39.0 to 49.0, so the completed calls are more exact even where selection is slightly less complete.

5.2.2 Spoken knowledge and duplex timing

Table 5: Spoken knowledge and duplex timing. DuplexAgent raises spoken-knowledge accuracy on both attachments, while duplex timing stays near the attached interaction model. Marks follow Table 4.

Method	Delegate	Intelligence		FDB-v1			
		OBQA	MMSU	Pause Synthetic ↓	Pause Candor ↓	Turn TOR ↑	Interrupt TOR ↑
Moshi	✗	23.8	24.2	1.000	0.991	1.000	0.94
MoshiRAG	✓	30.8	28.1	1.000	0.981	1.000	0.83
MiniCPM-o 4.5	✗	69.6	55.0	0.132	0.222	0.898	0.85
Gander	✓	82.4	62.7	0.000	0.019	0.390	0.72
Realtime-Omni-9B	✗	84.1	64.5	0.000	0.019	0.864	0.68
Realtime-Venus-Omni	✓	41.4	27.4	0.000	0.019	0.797	0.78
VoiceChat	✗	54.2	36.1	0.162	0.343	0.983	1.00
Qwen-Audio-Agent*	✓	60.4	37.6	0.206	0.324	0.966	0.99
DuplexAgent*	✓	80.6	80.5	0.117	0.241	0.961	0.965
Qwen-Realtime	✗	86.3	64.8	0.074	0.278	0.898	0.89
Qwen-Audio-Agent [†]	✓	84.1	71.8	0.103	0.176	0.915	0.92
DuplexAgent[†]	✓	92.1	82.7	0.147	0.231	0.966	0.97

The same handoff lifts a spoken choice and can drop the call. Table 5 is the other half of the pattern in Table 4. A delegate that receives a description can still write a better spoken answer. MoshiRAG raises OBQA from 23.8 to 30.8 and MMSU from 24.2 to 28.1 while its calling subsets stay at 0.0, because retrieval supplies text for speech and never a benchmark function. Gander raises OBQA from 69.6 to 82.4 and MMSU from 55.0 to 62.7: the coding agent answers the question, and the interaction model speaks that answer, while the scored tool channel still carries only a task title. Qwen-Audio-Agent on VoiceChat raises the same cells from 54.2 to 60.4 and from 36.1 to 37.6, the small gain available when the contract prefers to answer in speech. On Qwen-Realtime the same agent lowers OBQA from 86.3 to 84.1 and raises MMSU from 64.8 to 71.8. In each of these rows, spoken knowledge and executable accuracy move apart because the handoff is a description.

Realtime-Venus-Omni shows the failure mode in which the description also displaces the answer. OBQA falls from 84.1 to 41.4 and MMSU from 64.5 to 27.4. Realtime-Omni-9B states the option letter; after the harness, the transcript that reaches the scorer is often the delegate span or a receipt, so the letter is no

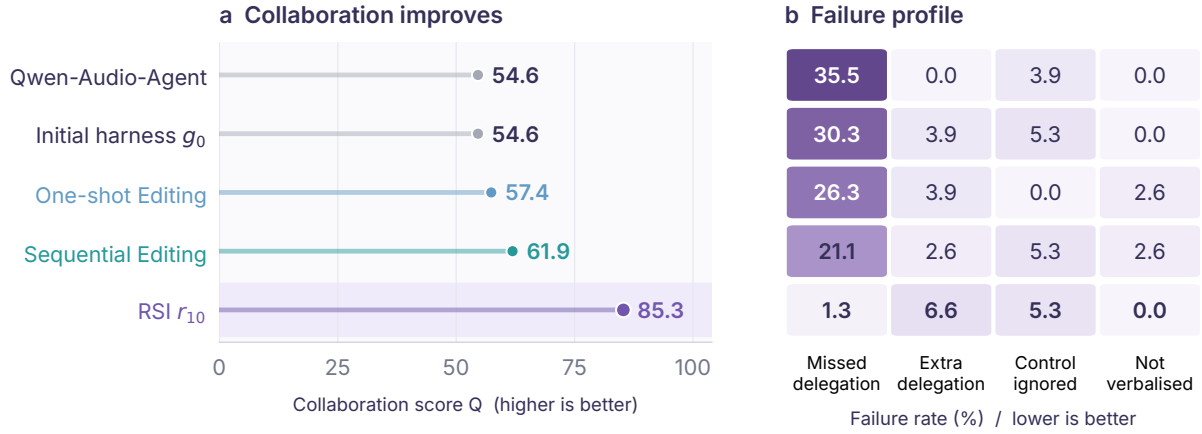


Figure 6: VoiceChat harness ablation on the fixed core. The left panel is the collaboration score Q , and the right panel is the rate of each diagnostic failure. Duplex-Harness-RSI scores higher than one-shot or sequential editing.

longer there to be parsed. Tool accuracy falls on the same attachment. A background model therefore improves a live system only when the harness decides which requests stay with the interaction model and which details travel with a delegated request.

DuplexAgent makes that separation on both attachments. A direct question remains a spoken answer; a tool request remains an executable call whose arguments were passed through. OBQA rises from 54.2 to 80.6 on VoiceChat and from 86.3 to 92.1 on Qwen-Realtime; MMSU rises from 36.1 to 80.5 and from 64.8 to 82.7. The larger gain on the weaker interaction model is the delegate supplying reasoning the interaction model does not have. The gains combine access to the delegate with the harness that selects the route, packages the handoff, and returns the result. Section 5.3 isolates how that harness was revised.

Timing remains a separate constraint. A description-only handoff also changes who holds the floor. Gander’s pause TOR falls to 0.000 / 0.019, and its turn TOR falls from 0.898 to 0.390, so many finished user turns receive no reply while a titled task is open. Realtime-Venus-Omni keeps the Realtime-Omni-9B pause TOR and moves turn TOR from 0.864 to 0.797. Moshi and MoshiRAG speak through the user’s hesitation (pause TOR at or near 1.0). DuplexAgent stays near the interaction model it is attached to. With VoiceChat, synthetic and Candor pause TOR fall from 0.162/0.343 to 0.117/0.241, while turn TOR and interrupt TOR decline slightly from 0.983/1.000 to 0.961/0.965. With Qwen-Realtime, turn TOR rises from 0.898 to 0.966 and interrupt TOR from 0.890 to 0.970. Candor pause TOR improves from 0.278 to 0.231, but synthetic pause TOR worsens from 0.074 to 0.147.

Better task performance therefore coexists with a less favorable result on one pause condition, while the live channel remains responsive.

5.3 Harness improvement

Pooled editing, sequential editing, and RSI. We compare three ways of using the same exam material to improve the initial VoiceChat harness g_0 :

1. *One-shot Editing* pools all test cases encountered during the ten-round RSI campaign and gives them to the coding agent for one harness revision.
2. *Sequential Editing* gives the same coding agent the exams from those ten rounds in their original order and performs ten successive revisions. It omits the per-round diagnosis and the repair archive used by Duplex-Harness-RSI.
3. *Duplex-Harness-RSI* performs ten rounds. Each round plans an exam, runs and diagnoses it, edits the harness, and applies the development and protection gates. The Exam Planner and the Harness Editor use that round’s diagnosis and the repair archive. We report the incumbent at checkpoint r_{10} .

The exams used for editing extend beyond the fixed core. One-shot Editing receives all of them in one pool. Sequential Editing replays them in the original order for the same number of rounds. Figure 6 compares the resulting harnesses on the fixed core.

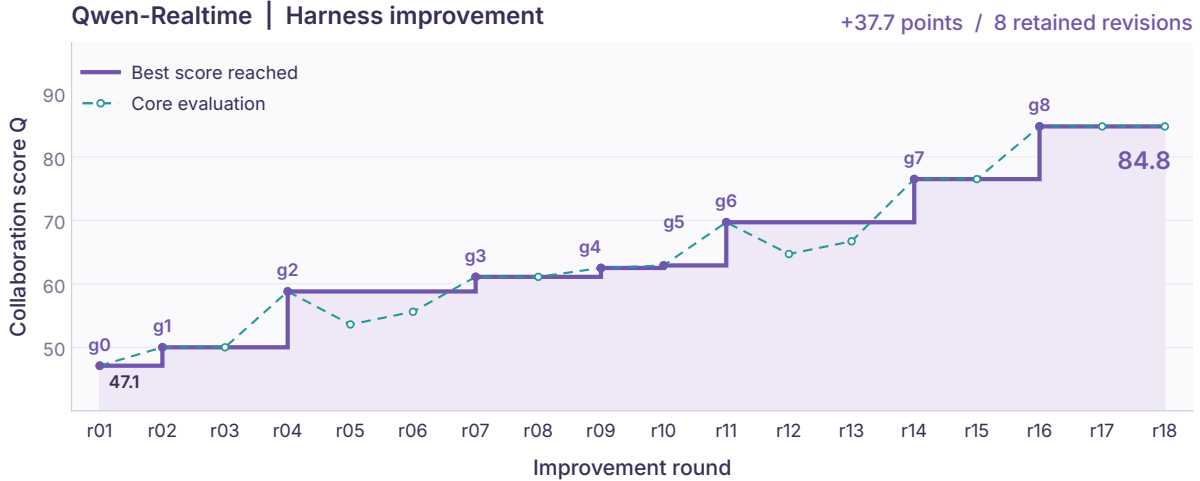


Figure 7: Duplex-Harness-RSI on Qwen-Realtime. Retained versions raise the collaboration score on the fixed core from the initial harness g_0 to g_8 .

Qwen-Audio-Agent and g_0 both score 54.6, with different rates of missed and extra delegation. One-shot Editing reaches 57.4 and Sequential Editing reaches 61.9. The ten-round checkpoint r_{10} reaches 85.3, improving on g_0 by 30.7 points and on Sequential Editing by 23.4 points. Repeated access to the same exams improves the harness. The full loop, which adds diagnosis and the repair archive, produces a substantially larger gain.

Missed delegation falls from 30.3% of items in the fixed core under g_0 to 1.3% under Duplex-Harness-RSI. Extra delegation rises from 3.9% to 6.6%, and ignored control stays at 5.3%. The main gain is recovered delegation, with a remaining tradeoff in unnecessary calls and unresolved control.

Version lineage and progress. Figure 7 follows the Qwen-Realtime campaign. It is separate from the VoiceChat checkpoint r_{10} : each attachment has its own initial harness and its own sequence of exams. Here the initial harness g_0 scores 47.1. Eight retained versions later, g_8 scores 84.8, a gain of 37.7 points. As defined in Section 3.4, g_k counts retained versions and r_t counts completed rounds, so a round can finish without producing a new g_k .

The dashed series records the fixed-core score after each round, including proposals that were not kept, so the line can fall. The solid staircase follows the best retained score. It steps up only when a candidate passes the development gate and the protection gate, and it stays flat when a round retains nothing, leaving the current harness in place. The late plateau is a run of rounds in which further proposals no longer passed the gates. Together with the VoiceChat ablation, the curve shows the same accumulation on a second interaction model. A repair is added only after the paired comparison supports it.

What the modules learned. Table 6 shows the decision behind each retained repair. On Qwen-Realtime, Policy was delegating conversational turns and familiar facts too broadly. Stating the direct-answer categories explicitly, while still delegating requests that need additional capability, reduced extra calls by 50%. On VoiceChat, Intake could treat a late cancellation as work that had already completed. Resolving it against live task state, and applying the same handling to replacement preambles, reduced ignored cancellations by 81%. Replacement failures then fell by 65%, and later by 83.3%. Hold began reporting progress earlier and kept receipts brief, so missing progress updates fell by 75%. Worker kept the requested units and precision, calculated when the request required it, and retained verified result content, so wrong answers fell by 66.7%. Deliver was changed on both attachments: a result is spoken only after the user is silent, and an unsuccessful execution is distinguished from a successful answer. Overlaps and concealed failures both fell by 100%. Memory is the complementary case. Delegation was already correct, but the delegate received an incomplete question or missing context. Assembling the complete turn and retaining that context reduced those failures by 60%. These percentages come from the local exam used to test each repair. The fixed-core scores combine the repairs that the gates retained.

Rejected attempts show how a broad repair can regress. A broad delivery timeout can introduce other speech conflicts, and a broad direct-answer rule can miss required delegation. The development gate therefore keeps a repair only when the hypothesis is narrow and the paired comparison supports it.

Table 6: Representative harness repairs. Each change reduces the targeted failures on the local exam used to test it. Marks follow Table 4.

Module	Rule before repair	Rule after repair	Recorded local evidence
Policy [†]	Conversational and familiar-fact turns are routed too broadly.	State the direct-answer categories explicitly, while still delegating requests that need additional capability.	Extra calls reduced by 50%.
Intake [*]	A late cancellation can be treated as work that has already completed.	Resolve the cancellation against live task state, and apply the same control handling to replacement preambles.	Ignored cancellations reduced by 81%. Replacements reduced by 65%, then by 83.3%.
Hold [†]	Pending work does not receive a timely progress update.	Report progress earlier, and keep receipts brief and compatible with the adapter.	Missing progress updates reduced by 75%.
Worker [*]	Conversions skip the required calculation, or reformulate output that was already verified.	Preserve the requested units and precision, calculate when the request requires it, and retain verified result content.	Wrong answers reduced by 66.7%.
Deliver [*]	Completion can start speech during the user’s turn, and an inability to execute is reported as success.	Speak only once the user is silent, and distinguish unsuccessful execution from a successful answer.	Overlaps reduced by 100% on both attachments. Concealed failures reduced by 100%.
Memory [*]	Delegation is correct, but the delegate receives an incomplete question or missing context.	Assemble the complete user turn, and retain the context that a delegated request needs.	Missing-context failures reduced by 60%.

6 Discussion and limitations

The results support a practical division of responsibility: the interaction model maintains the live conversation, the delegation pool supplies task capability, and the harness coordinates the two. The retained repairs make routing, task control, context, and the timing of delivery more precise, and spoken knowledge, executable tool use, and fixed-core collaboration rise with that precision.

These gains stop at the harness. Duplex-Harness-RSI rewrites module instructions and executable rules, and the interaction-model weights stay fixed. A repair can require silence before speech, or a complete question before delegation, while the model’s perception of overlapped speech and its own judgement that a turn has ended remain as they were. When the gates accept or reject a candidate, this remainder is still in the trace. The Observer has aligned speech, delegation, and delivery with the expected behaviour, and the Diagnoser has named the cause, the module, and the pattern. The Exam Planner already turns those signatures into the next timed conversations. The record thus shows which failures a module edit can absorb, and which remain in the interaction model’s own perception and timing. Agentic reinforcement learning for full-duplex collaboration continues the loop from there: the simulator keeps supplying the exams, the traces, and the failure signatures, and the update extends from the harness to the interaction model on the live timeline.

7 Conclusion

DuplexAgent keeps a full-duplex interaction model on the live channel and a delegation pool for reasoning and longer work, and writes their exchange as six editable modules: Policy, Intake, Hold, Worker, Deliver, and Memory. Duplex-Harness-RSI reuses the pool that serves the user. The Exam Planner chooses the next conversations from attributed failures and the repair archive, the Harness Editor revises the modules the diagnosis names, and paired checks on the fixed core and the protection set decide what is kept. The simulator makes each revision checkable, turning request types and interaction patterns into timed tests and traces. On VoiceChat and on Qwen-Realtime, the retained harness improves spoken knowledge, executable tool use, and collaboration on the fixed core, while the model weights remain those the campaign started with. The revisions show how far coordination can be improved from evidence. The same traces still contain the perception and timing that a module edit does not reach. Agentic reinforcement learning for full-duplex collaboration can take up that remainder, training the interaction model in the simulator that already evaluates the harness.

References

- Ant Group (Venus Team) and Tsinghua University. Realtime-venus: A full-duplex interaction system with asynchronous delegation. *arXiv preprint arXiv:2609.13814*, 2026.
- Jagadeesh Balam, Travis Bartley, Edresson Casanova, Sanjay Chauhan, Chen Chen, Zhehuai Chen, et al. NemotronLabs VoiceChat: An open full-duplex speech-to-speech model with tool calling capabilities. *arXiv preprint arXiv:2609.21967*, 2026. URL <https://arxiv.org/abs/2609.21967>. Model card: <https://huggingface.co/nvidia/NVIDIA-NemotronLabs-VoiceChat-11B>.
- Junjie Chen, Yao Hu, Junjie Li, Kangyue Li, Kun Liu, Wenpeng Li, Xu Li, Ziyuan Li, Feiyu Shen, Xu Tang, et al. FireRedChat: A pluggable, full-duplex voice interaction system with cascaded and semi-cascaded implementations. *arXiv preprint arXiv:2509.06502*, 2025.
- Yiming Chen, Xianghu Yue, Chen Zhang, Xiaoxue Gao, Robby T. Tan, and Haizhou Li. VoiceBench: Benchmarking LLM-based voice assistants. *arXiv preprint arXiv:2410.17196*, 2024.
- Chung-Ming Chien, Manu Orsini, Eugene Kharitonov, Neil Zeghidour, Karen Livescu, and Alexandre Défossez. MoshiRAG: Asynchronous knowledge retrieval for full-duplex speech language models. *arXiv preprint arXiv:2604.12928*, 2026.
- Junbo Cui et al. MiniCPM-o 4.5: Towards real-time full-duplex omni-modal interaction. *arXiv preprint arXiv:2604.27393*, 2026.
- Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: A speech-text foundation model for real-time dialogue. *arXiv preprint arXiv:2410.00037*, 2024.
- Chong Deng, Yunjie Ji, Yuxiang Kong, Xiangang Li, Xu Li, Binbin Zhang, Haina Zhu, and Jianheng Zhuo. Qwen-Audio-Agent technical report. *arXiv preprint arXiv:2609.25195*, 2026. URL <https://arxiv.org/abs/2609.25195>.
- Zhang He, Wenqian Cui, Haoning Xu, Xiao-Hui Li, Lei Zhu, Haoli Bai, Ma Shaohua, and Irwin King. MTR-DuplexBench: Towards a comprehensive evaluation of multi-round conversations for full-duplex speech language models. In *Findings of ACL*, pp. 5334–5351, 2026.
- Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *Proc. ICLR*, pp. 21344–21377, 2025.
- Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.
- Guan-Ting Lin, Shih-Yun Shan Kuan, Qirui Wang, Jiachen Lian, Tingle Li, Shinji Watanabe, and Hung-yi Lee. Full-duplex-bench v1.5: Evaluating overlap handling for full-duplex speech models. *arXiv preprint arXiv:2507.23159*, 2025a.
- Guan-Ting Lin, Jiachen Lian, Tingle Li, Qirui Wang, Gopala Anumanchipalli, Alexander H. Liu, and Hung-yi Lee. Full-duplex-bench: A benchmark to evaluate full-duplex spoken dialogue models on turn-taking capabilities. *arXiv preprint arXiv:2503.04721*, 2025b.
- Guan-Ting Lin, Chen Chen, Zhehuai Chen, and Hung-yi Lee. Full-duplex-bench-v3: Benchmarking tool use for full-duplex voice agents under real-world disfluency. *arXiv preprint arXiv:2604.04847*, 2026.
- Yueqian Lin, Zhengmian Hu, Jayakumar Subramanian, Qinsi Wang, Nikos Vlassis, Yiran Chen, et al. AsyncVoice agent: Real-time explanation for LLM planning and reasoning. In *Proc. ASRU*, pp. 1–4, 2025c.
- Zhanxun Liu, Yifan Duan, Mengmeng Wang, Pengchao Feng, Haotian Zhang, Xiaoyu Xing, Yijia Shan, Haina Zhu, Yuhang Dai, Chaochao Lu, et al. X-Talk: On the underestimated potential of modular speech-to-speech dialogue systems. *arXiv preprint arXiv:2512.18706*, 2025.
- Zhenyu Liu, Xuanyu Zhang, Yunxin Li, Qixun Teng, Shenyuan Jiang, Haolan Chen, Mingjun Zhao, Fanbo Meng, Yu Xu, Yancheng He, et al. Hierarchical acoustic-semantic modeling: Modality separation and semantic coherence for full-duplex SLMs. In *Proc. ACL*, pp. 9264–9280, 2026.
- Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K Qiu, and Yuqing Yang. Agent lightning: Train any AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

- Orantqing, Shengpeng Ji, Junlong Tong, Jialong Zuo, et al. Multimodal duplex interaction agent. *arXiv preprint arXiv:2609.08977*, 2026.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 48371–48392, 2025. URL <https://proceedings.mlr.press/v267/patil25a.html>.
- Soham Ray, Keshav Dhandhania, Victor Barres, and Karthik Narasimhan. τ -voice: Benchmarking full-duplex voice agents on real-world domains, 2026. URL <https://arxiv.org/abs/2603.13686>.
- Rajarshi Roy, Jonathan Raiman, Sang-gil Lee, Teodor-Dumitru Ene, Robert Kirby, Sungwon Kim, Jaehyeon Kim, and Bryan Catanzaro. PersonaPlex: Voice and role control for full duplex conversational speech models. In *Proc. ICASSP*, pp. 16137–16141, 2026.
- Zihan Tan, Leixin Sun, Zitong Shi, Yitao Liu, Jiajun Wu, et al. MetaRSI / RSI²: A meta-recursive self-improving system for recursive self-improving systems themselves. *arXiv preprint arXiv:2609.06396*, 2026.
- Chengyou Wang, Hongfei Xue, Guojian Li, Zhixian Zhao, Shuiyuan Wang, Shuai Wang, Xin Xu, Hui Bu, and Lei Xie. Full-duplex interaction in spoken dialogue systems: A comprehensive study from the ICASSP 2026 HumDial challenge. *arXiv preprint arXiv:2604.21406*, 2026.
- Xiong Wang, Yangze Li, Chaoyou Fu, Yunhang Shen, Lei Xie, Ke Li, Xing Sun, and Long Ma. Freeze-omni: A smart and low latency speech-to-speech dialogue model with frozen LLM. *arXiv preprint arXiv:2411.00774*, 2024.
- Siwei Wu, Jincheng Ren, Yizhi Li, Haau-Sing Li, Chengran Yang, Yuxuan Zhang, Weicheng Gu, Jian Yang, Riza Batista-Navarro, Chuanyi Zhang, Xianglong Liu, Ming Zhou, Bryan Dai, and Chenghua Lin. ModularRSI: Modular and generalizable recursive harness self-improvement. *arXiv preprint arXiv:2609.14857*, 2026.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. Re-Act: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.
- Wenyi Yu, Siyin Wang, Xiaoyu Yang, Xianzhao Chen, Xiaohai Tian, Jun Zhang, Guangzhi Sun, Lu Lu, Yuxuan Wang, and Chao Zhang. SALMONN-omni: A codec-free LLM for full-duplex speech understanding and generation. *arXiv preprint arXiv:2411.18138*, 2024.
- Hangfan Zhang, Shao Zhang, Kangcong Li, Chen Zhang, Yang Chen, Yiqun Zhang, Lei Bai, and Shuyue Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026.