REAL-TIME AUDIO TOOL BENCH: EVALUATING TOOL USE IN MULTI-TURN VOICE INTERACTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Voice-native models now call tools directly from speech, yet how reliably they do so in live, multi-turn conversation is largely unmeasured. Real-time voice raises problems that text tool-calling benchmarks cannot expose: arguments must be heard correctly, details arrive over several turns, and users interrupt, sometimes while a call is still pending. We introduce **Real-Time Audio Tool Bench**, a benchmark of 1040 tasks over 44 tools in 7 domains, each run live over a provider's streaming API. Three suites cover explicit multi-turn requests (Reactive), implicit intent (Proactive), and mid-conversation changes of mind (Interruption). Deterministic checks grade which tools are called, with what arguments, and when, with no LLM judge; each failure is attributed to perception (e.g., a misheard name or number) or to reasoning (e.g., a missing, premature, or unwarranted call). Across six models from four providers, the best passes 38.7% of tasks, and each suite exposes a different bottleneck: text transcripts in place of audio raise Reactive pass rates by 30 to 65 percentage points; models act on nearly all Strong-intent Proactive tasks but complete at most 21% of them; and when a pending call's result never returns, no model recovers on more than 10.3% of tasks. None of the static test-time harnesses we test improves all three suites at once. We release the benchmark, the evaluation framework, and all evaluation traces.

1 INTRODUCTION

Voice assistants are turning into agents. Voice-native models such as OpenAI Realtime (OpenAI, 2024), Gemini Live (Solovyev & Basu Mallick, 2024), Grok Voice (xAI, 2026), and GLM-Realtime (Zeng et al., 2024) take speech directly as input and call tools mid-conversation, without the traditional cascade of speech recognition, a text LLM, and speech synthesis. These models already ship through production APIs, yet there is no systematic way to measure how reliably they call tools during real-time voice interaction.

Real-time voice tool calling is not text tool calling with an audio front end. First, arguments must be heard: a misheard name or a dropped digit yields a call that is well formed but books the wrong thing. Second, requests unfold over several turns while audio streams in, so the model must judge when it has heard enough to act; a call made too early commits incomplete arguments. Third, users interrupt to correct a detail, cancel, or slip in an unrelated request, both while the model is speaking and while a tool call is still pending. Existing benchmarks miss most of this. Text benchmarks such as BFCL (Patil et al., 2025) and ToolBench (Qin et al., 2024) assume a clean written request. Audio benchmarks add speech but not live interaction: AU-Harness (Surapaneni et al., 2025) converts text tasks into synthesized speech for a pipeline evaluation, CAVA (Held et al., 2025) matches function names without checking argument values, and VoiceAgentBench (Jain et al., 2025) evaluates function calling without streaming. None of them separates mishearing from acting too early or from failing to recover after an interruption, so when a call fails, they cannot say why.

We introduce **Real-Time Audio Tool Bench**, a benchmark for tool calling in real-time, multi-turn voice interaction (Figure 1). Its 1040 tasks span 44 tools in 7 domains and form three suites, each built around a common user behavior. In **Reactive** (200 tasks), the user makes explicit requests whose arguments arrive over several turns. In **Proactive** (600 tasks), the user's goal is only implied, at three levels of intent strength; at the weakest level the user is only reminiscing, and the correct behavior is not to act. In **Interruption** (240 tasks), the user changes their mind while the model is

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

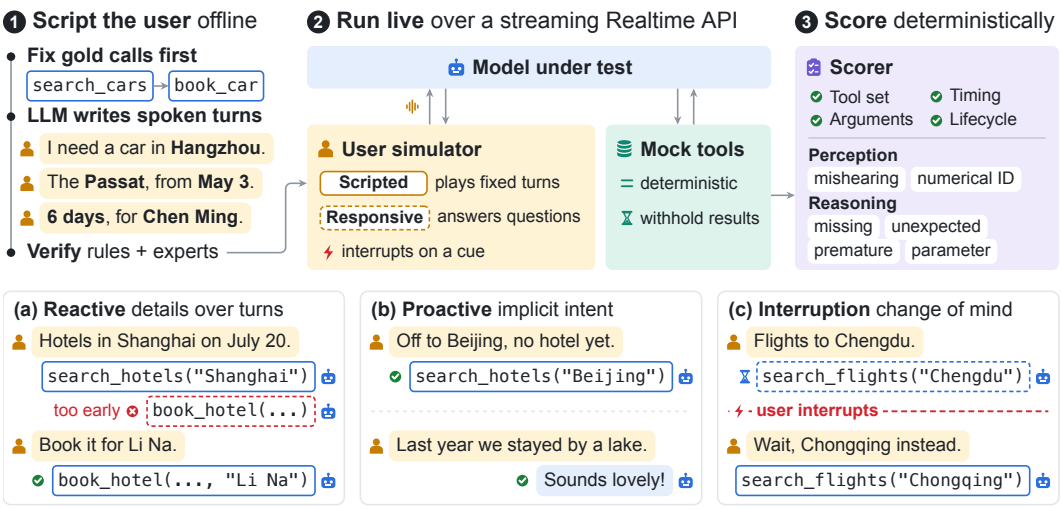


Figure 1: **Overview. (1)** Tasks are scripted offline. The gold tool calls and their argument values are fixed from an executable tool chain, and an LLM writes spoken user turns that reveal these values a few at a time. Rule checks verify every task, and experts audit a stratified sample. **(2)** Each task runs live over the provider's streaming API. The user simulator streams pre-rendered audio and interrupts on a cue: a keyword in the model's speech, or a call whose result the mock tools withhold. All reported results use the scripted mode; an optional responsive mode also answers the model's follow-up questions from the task goal, without seeing the gold calls. **(3)** Deterministic checks score each trace, and each failure is attributed to perception or reasoning. *Bottom:* one example per suite. (a) A call issued before its arguments are complete is premature. (b) Implicit intent should trigger a call; Weak intent (below the dashed line) should not. (c) The user changes their mind while a call is pending.

speaking or while a tool call is pending, leaving a *dangling call* whose result arrives late or never. User turns are spoken in voices cloned with ElevenLabs from real human recordings in Mozilla Common Voice (Ardila et al., 2020), and every task runs live over the provider's streaming API.

Three deterministic checks score each tool call: whether the right tools are called, whether their arguments are correct, and whether each call waits until its arguments have been given. No LLM judge is involved. Each failure is then assigned to one of six categories: two *perception* errors, in which the model mishears an entity or a number, and four *reasoning* errors, in which it misses a call, makes an unexpected one, calls too early, or reasons incorrectly about an argument value. The split shows whether a failure starts in the audio or in the decision about what to call.

We evaluate six voice-native models from four providers. The best, gpt-realtime-1.5, passes 38.7% of tasks, and each suite exposes a different bottleneck. On Reactive tasks, perception is the limit: replacing the audio with text transcripts raises pass rates by 30 to 65 percentage points. On Proactive tasks, models recognize implicit intent, acting on at least 95% of Strong-intent tasks, but complete at most 21% of them correctly. On Interruption tasks, models lose track of pending calls: when a dangling call's result never returns, no model recovers on more than 10.3% of tasks. Static test-time harnesses that each target one of these failures raise some suites and lower others, and none improves all three at once.

Our contributions are (1) a benchmark of live, multi-turn voice tool-calling tasks, including interruptions that leave a dangling call (§3); (2) deterministic scoring with a failure attribution that separates perception from reasoning and agrees with human labels at 0.87 macro-F1 (§4); and (3) an evaluation of six models that finds a distinct bottleneck in each suite, with a harness study in which every static fix for one suite costs accuracy on another (§5). We release the tasks with audio, the evaluation framework, and all evaluation traces.

2 Related Work

Tool-calling benchmarks. Tool-calling evaluation has matured on clean text. BFCL (Patil et al., 2025) is the de facto standard, and later benchmarks extend it to multi-turn, stateful, and agentic settings (Yao et al., 2024; Lu et al., 2024; Waqas et al., 2026; Cuadron et al., 2025), to nested and long-context calls (Qin et al., 2024; Guo et al., 2024; Basu et al., 2024; Ye et al., 2025; Zhong et al., 2025), and to multilingual requests (Kulkarni et al., 2025). All of them present the request as text, so they cannot expose misheard arguments, details that arrive over several spoken turns, or interruptions.

Speech and voice-agent evaluation. Speech benchmarks measure comprehension, dialogue dynamics, and turn-taking (Wang et al., 2025; Chen et al., 2024; Huang et al., 2024; Lin et al., 2025b; Arora et al., 2025; Lin et al., 2025a) but do not involve tool execution, which leaves untested the assumption that voice is text plus speech recognition. Meanwhile, end-to-end voice models with native tool calling have shipped (OpenAI, 2024; Solovyev & Basu Mallick, 2024; Gemini Team, Google, 2023; xAI, 2026; Zeng et al., 2024; Xu et al., 2025; Défossez et al., 2024), and new architectures interleave streaming perception with partial-turn, asynchronous, or speculative tool calls under sub-second latency (Chiang et al., 2025; Feng et al., 2026; Hooper et al., 2026).

Voice tool-calling benchmarks. The closest benchmarks test tool calling from speech, but each covers only part of the live setting. Pipeline benchmarks convert text benchmarks into synthesized speech and reuse the text gold labels (Surapaneni et al., 2025; Laskar et al., 2026). Audio-native datasets test single-turn function calling (Jain et al., 2025; Held et al., 2025; Pahwa et al., 2026; Wu et al., 2025). Concurrent full-duplex benchmarks measure the gap between text and voice agents on 278 grounded tasks (Ray et al., 2026) and multi-step tool use under disfluent real human speech (Lin et al., 2026). None of them attributes failures to the separate capabilities that streaming multi-turn dialogue with real tools demands: perception, timing of the call, and tracking of state across interruptions. Our benchmark isolates each of these, adds implicit-intent tasks and interruptions during pending calls, and scores every call deterministically.

3 Real-Time Audio Tool Bench

Each task is a scripted multi-turn conversation paired with a gold set of tool calls and their exact argument values. The three suites differ in how the user expresses what they want.

3.1 Task Suites

Reactive (200 tasks). The user makes explicit requests ("help me search flights from New York to LA") and supplies the required arguments over several turns, at most three per turn. The model must wait until all required arguments of a call have been given; calling earlier commits incomplete or guessed arguments. A task needs 1 to 8 tools, arranged as a single call, a chain linked by IDs (e.g., `search_flights` → `book_flight`), or a DAG with parallel branches, such as searching for and booking both a flight and a hotel.

Proactive (600 tasks). The user never gives a direct command, so the intent to act must be inferred. Tasks fall into three bands of intent strength. In *Strong* tasks (240), the user presents an actionable request without a direct command ("If The Wandering Earth 3 is playing, I'm totally down for it"). In *Medium* tasks (240), the intent is vaguer ("it'd be kind of nice to just unwind with a movie or something"), and a confirmation or comparison turn typically precedes the action. In both bands, the model passes only if it completes the action within the dialogue; confirming or asking a clarifying question first is not penalized. In *Weak* tasks (120), the user reminisces, quotes someone, or speculates ("I remember my buddy Chen Ming went to see The Wandering Earth 3"), so task-relevant entities appear without any request. Acting on them would be a false positive under common deployment policies, so the model passes only by making no tool call.

Interruption (240 tasks). The user changes their mind at one of two moments (Appendix Figure 6). In *speech-phase* tasks (90), the tool calls have executed and the model is delivering its spoken reply when the user corrects an argument, cancels the operation, or inserts a new request before returning

to the original one. In *tool-phase* tasks (150), the model has issued a call whose result has not yet returned, a dangling call, when the interruption arrives; the framework creates this window by executing the call but withholding its result. The user then cancels the pending call (*Cancel*), changes its arguments (*Correction*), or inserts an urgent request and later asks to resume. In these three *Priority* subtypes, the withheld result never returns, so the model must re-issue the call (*No-return*); returns after the inserted task, so the model must use it instead of calling again (*Late-return*); or returns during the inserted task, so the model must interleave the two (*Early-return*).

3.2 Task Construction and Validation

Tasks are built in four steps, detailed in Appendix C.

Executable tool chains. A dependency graph over the 44 tools records which search supplies the ID that a booking consumes, which bookings constrain each other (a flight's arrival sets the hotel check-in), and which tools are interchangeable (flights and trains). Tool chains are sampled from this graph, so every booking follows its search. All argument values are fixed from a deterministic mock-data store before any dialogue is written, and ID arguments such as `hotel_id` come from actually running the upstream search. Every task is therefore executable end to end, and every tool returns the same result for the same arguments.

User turns. GPT-5.2 writes natural spoken turns that reveal the fixed values over 4 to 7 turns, at most three per turn, in the style of the target suite and band (Appendix C.2). We record the turn in which each value first appears, which fixes the earliest turn at which each call is allowed (Appendix C.4).

Audio. We clone the voices of N speakers from their Mozilla Common Voice recordings with ElevenLabs (ElevenLabs, 2026) and speak every user turn in one of these voices. Each task is rendered once and cached without added noise, so every model receives byte-identical audio and differences between models cannot come from the input.

Verification. A rule checker confirms the schema, that every ID chain executes against the mock data, and that every argument value is expressed in the dialogue; an LLM checker flags dialogues whose implied tool set disagrees with the gold calls (Appendix C.3). Three human studies validate the result (Appendix E). Two experts who did not author tasks solved 193 of 200 stratified tasks from the audio alone. Two annotators labeled the intent band of 200 Proactive tasks with a Cohen's κ (Cohen, 1960) of 0.88 between them, and every disagreement fell on the Strong/Medium boundary. On 200 failed traces stratified by failure category, the automatic attribution (§4) agrees with an independent annotator at 0.87 macro-F1.

The resulting benchmark has 1040 tasks over 44 tools: 41 tools in 7 domains (travel, entertainment, life services, healthcare, education, finance, and transportation) and 3 general-purpose tools, such as web search, that no task requires and that serve as distractors. Appendix B lists every tool, the dependency graph, and the task distribution.

4 Evaluation

Live execution. Each task runs in its own WebSocket session with the provider's streaming API, which detects the end of each user turn with server-side voice activity detection (VAD). The model receives only the audio and the 44 tool schemas. A user simulator streams each turn's cached audio in order, and every tool call executes against the mock backend, which returns its result to the session. Interruptions fire on a cue: in the speech phase, when a task-specific keyword appears in the model's streamed transcript; in the tool phase, when the model issues the call whose result is withheld. The framework then cancels the model's response and sends the interruption audio (Appendices D.3 and D.4).

Scoring. A task passes only if all of its tool calls satisfy three checks. (1) *Tool set*: every gold tool is called, and no other tool is. Extra calls count as failures because many tools have side effects; an unrequested booking cannot simply be undone. (2) *Arguments*: every argument matches

Table 1: Pass rate (%) with 95% bootstrap confidence intervals; best per column in **bold**. *Overall* is the unweighted mean of Reactive, Proactive (mean of Strong, Medium, and Weak), and Interruption (mean of the speech and tool phases). On Weak tasks, a pass means making no tool call.

			Proactive			Interruption	
Model	**Overall**	**Reactive**	**Strong**	**Medium**	**Weak**	**Speech**	**Tool**
gpt-realtime-1.5	**38.7** ±3.3	**41.9** ±7.1	**20.9** ±5.3	**19.7** ±5.0	**89.0** ±5.5	**32.2** ±10.0	**30.0** ±7.3
gpt-realtime	27.5 ±3.0	28.8 ±6.1	6.3 ±3.1	11.3 ±4.0	75.4 ±7.6	24.4 ±8.9	20.7 ±6.3
gpt-realtime-mini	23.0 ±3.0	27.3 ±6.3	8.4 ±3.6	7.6 ±3.6	51.7 ±8.9	20.0 ±8.3	18.7 ±6.3
qwen3.5-omni-plus	21.5 ±2.5	13.1 ±4.8	9.2 ±3.6	5.5 ±2.7	86.4 ±6.4	24.4 ±8.9	10.7 ±5.0
grok-voice-think	21.5 ±2.5	16.7 ±5.3	12.2 ±4.0	12.2 ±4.2	81.4 ±7.2	15.6 ±7.2	9.3 ±4.7
gemini-3.1-flash	7.3 ±1.8	9.1 ±3.8	0.8 ±1.0	0.8 ±1.0	29.7 ±8.4	3.3 ±3.9	1.3 ±1.7

its gold value under type-aware rules that tolerate formatting differences ("138-1234-5678" vs. "13812345678") but not content errors. A hotel booked in the wrong city cannot be fixed by the spoken reply, so this check is strict. If a tool is called more than once, the last call counts, which credits self-correction. (3) *Timing*: no call is issued before all of its required arguments have been given. If `book_restaurant` needs a contact name given in turn 5 and a phone number given in turn 6, it may not be called before turn 6. This is only a lower bound. The model may confirm or ask a clarifying question first, and a late call passes as long as it arrives before the task ends. Suite-specific rules extend these checks. Medium's timing bound is relaxed to the turn at which clarification completes, Weak passes only if no tool is called, and the Interruption subtypes add lifecycle constraints on the pending call, such as not re-issuing a cancelled call or a call whose result has already returned (Appendix D).

Failure attribution. Each failed call receives one of six categories. A tool-set violation is a *missing call* or an *unexpected call*, and a timing violation is a *premature call*. An argument mismatch is classified by its edit distance to the gold value. Within distance 2, it is a perception error: a *numerical-ID error* for digit strings such as phone numbers, and an *entity mishearing* otherwise (e.g., "Li Na" → "Lina"). Beyond distance 2, it is a *parameter-reasoning* error (e.g., shortening "RIDE123456" to "123456"). Edit distance only assigns the category; it never turns a mismatch into a pass.

Metrics. We report task pass rates with 95% bootstrap confidence intervals. The Proactive and Interruption scores average their bands and phases, and the overall score averages the three suites. We also measure two latencies from user-audio commit, the moment the client finishes sending a user turn, including the silence padding that server-side VAD requires: *TTFT*, the time to the model's first audio, and *TTFC*, the time to its first complete tool call. Both include end-of-turn detection, so they measure the delay a user experiences rather than inference time alone.

5 RESULTS

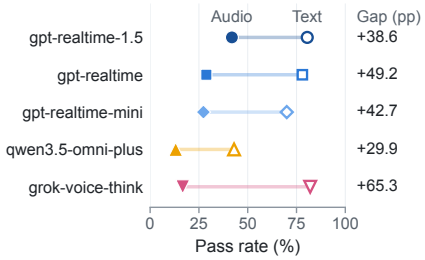
Table 1 reports pass rates for the six models. The best, gpt-realtime-1.5, passes 38.7% of tasks overall. On every column that requires a tool call, no model exceeds 42%; every model stays below 21% on the Strong and Medium bands, and gemini-3.1-flash stays below 10%. The only high scores are on Weak tasks, where passing means not acting. Confidence intervals are wide on the smaller suites (up to ±10 pp on the 90 speech-phase tasks), so we rely on large or consistent gaps rather than on small differences between models.

Table 2 shows where the failures come from, and the answer differs by suite. On Reactive, perception errors are the largest category for four of the six models. On Proactive, missing calls are the largest category for every model. On Interruption, premature and missing calls together account for 41–67% of failed calls. The next three subsections examine each suite in turn (§5.1–5.3); we then report latency (§5.4) and test whether static harnesses close the gaps (§5.5).

Table 2: Share of failed calls (%) by failure category in each suite. *Perc.* = perception (entity mishearing and numerical-ID error), *Prem.* = premature call, *Param.* = parameter reasoning, *Miss.* = missing call. Unexpected calls (at most 5.8%) are omitted, so rows sum to slightly less than 100. The largest category per model and suite is in **bold**. Appendix Table 24 gives the full breakdown.

	Reactive				Proactive				Interruption			
Model	Perc.	Prem.	Param.	Miss.	Perc.	Prem.	Param.	Miss.	Perc.	Prem.	Param.	Miss.
gpt-realtime-1.5	**55.7**	8.0	19.5	16.7	23.4	10.6	18.1	**46.0**	**43.3**	27.6	13.4	13.4
gpt-realtime	**39.0**	25.5	20.3	15.2	12.8	21.2	25.4	**37.9**	27.3	**42.0**	12.6	16.1
gpt-realtime-mini	**43.8**	13.4	22.3	20.5	17.1	15.6	20.2	**41.3**	**34.2**	27.7	15.5	21.9
qwen3.5-omni-plus	10.4	16.2	30.2	**43.2**	8.1	5.1	23.8	**61.7**	17.4	**36.5**	17.4	25.7
grok-voice-think	22.0	1.8	11.1	**65.1**	28.1	4.3	20.5	**44.4**	23.2	5.0	14.4	**54.7**
gemini-3.1-flash	**35.6**	2.8	29.0	32.6	12.0	9.4	23.3	**50.2**	13.3	16.4	17.1	**51.0**

(a) Pass rate: audio vs. text input



(b) Median latency

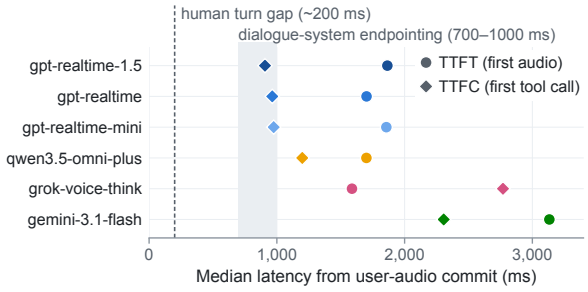


Figure 2: Reactive tasks. **(a)** Pass rate with audio and with text input; labels give the gap. Gemini is omitted because its API does not accept text-only turns. **(b)** Median latency from user-audio commit to the first audio (TTFT, circles) and the first tool call (TTFC, diamonds). The dashed line marks the ≈200 ms gap between human turns and the shaded band the 700–1000 ms endpointing thresholds of spoken dialogue systems.

5.1 Reactive: Perception Is the Bottleneck

Sending the user turns as text transcripts instead of audio isolates perception. On the Reactive tasks, text input raises pass rates by 29.9 to 65.3 pp for all five models that accept it (Figure 2a); gpt-realtime-1.5 nearly doubles, from 41.9% to 80.5%. The models can combine arguments across turns when they read them, and lose much of that ability when they must hear them.

The failure attribution agrees. Perception errors are the largest Reactive failure category for the three GPT models and Gemini, accounting for 36–56% of their failed calls (Table 2). They follow two patterns seen in all six models (Appendix F.2): entity mishearing, such as "Li Na" → "Lina" or "Zhou Jun" → "Zhao Jun", and numerical-ID errors, such as a digit dropped from an 11-digit phone number. The errors concentrate on person names and phone numbers, short strings in which a single wrong character changes the value, whereas city names are rarely misheard (Appendix F.5). The models also fail on largely the same tasks: pairwise Jaccard similarity between their sets of failed Reactive tasks is 59–84% (Appendix F.6), which points to a shared limitation rather than to weaknesses of individual models. Two models fail mostly through missing calls instead: qwen3.5-omni-plus (43% of failed calls) and grok-voice-think (65%). Yet grok-voice-think gains the most from text input (16.7% → 82.0%), so for this model the audio condition also causes calls to be dropped, not just misheard.

Errors compound because a task passes only if every call is right. Graded call by call, gpt-realtime-1.5 scores 84.9%, yet it passes 41.9% of tasks and only 14.6% of tasks with seven or more tools (Appendix Tables 34 and 21).

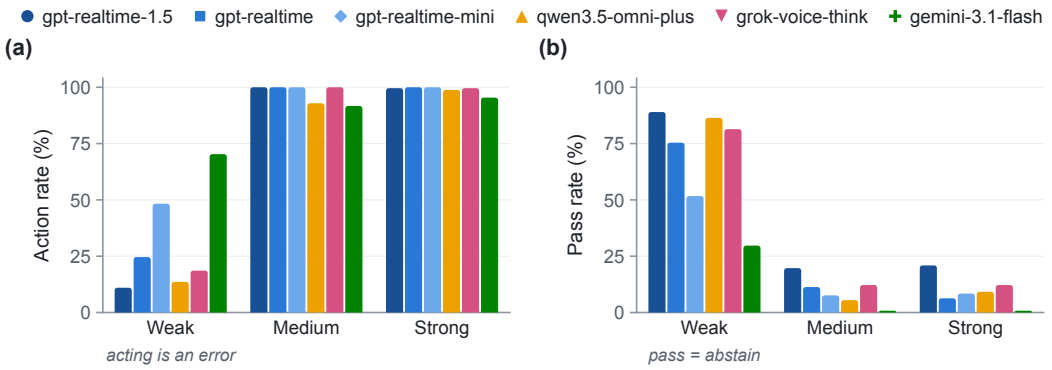


Figure 3: Proactive tasks by intent band. **(a)** Action rate, the fraction of tasks with at least one tool call, rises with intent strength for every model. **(b)** Pass rate stays low on Strong and Medium tasks despite near-universal action. On Weak tasks passing means abstaining, so a high Weak action rate is an error.

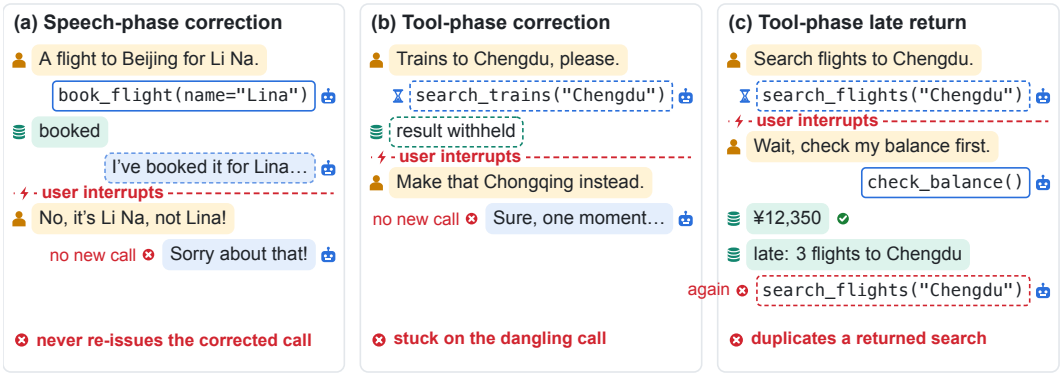


Figure 4: Three interruption failures. **(a)** Speech-phase correction: the model apologizes but never re-issues the corrected call. **(b)** Tool-phase correction: the model keeps waiting on the dangling call instead of issuing the corrected search. **(c)** Late-return: the model re-issues a search whose result has already arrived.

5.2 Proactive: Models Recognize Intent but Fail to Execute

Figure 3 separates deciding to act from acting correctly. Action rate rises with intent strength for every model and reaches at least 95% on Strong tasks, so models do register implicit intent. Yet no model passes more than 20.9% of Strong tasks, an *execution gap* of 78.7 to 94.6 pp between acting and acting correctly (Appendix Table 22). Missing calls are the largest Proactive failure category for every model (38–62% of failed calls; Table 2): models typically start the task but do not issue every call it requires. The opposite error appears on Weak tasks, where passing requires abstention: gemini-3.1-flash calls a tool on 70.3% of them and gpt-realtime-mini on 48.3%.

5.3 Interruption: Models Lose Track of Pending Calls

Table 3 breaks down the Interruption results by phase and subtype. No model passes more than a third of the tasks in either phase. Two failure patterns recur (Figure 4), and both appear in the attribution: premature calls, which here include acting on a superseded request, and missing calls, such as never re-issuing a corrected or unanswered call, together account for 41–67% of failed calls (Table 2).

Corrections do not override the original intent. After a speech-phase correction, models often apologize but do not re-issue the corrected call (Figure 4a). After a tool-phase cancellation or

Table 3: Interruption pass rate (%) by phase and tool-phase subtype; best per column in **bold**. In the Priority subtypes, the user inserts an urgent task while a call is pending, and the call's result never returns (No-ret.), returns after the inserted task (Late-ret.), or returns during it (Early-ret.). Cancel is the only subtype on which any model exceeds 50%.

		Tool-phase subtypes					
Model	**Speech**	**Cancel**	**Corr.**	**No-ret.**	**Late-ret.**	**Early-ret.**	**Tool**
gpt-realtime-1.5	**32.2**	**58.1**	20.0	**10.3**	**26.7**	**33.3**	**30.0**
gpt-realtime	24.4	38.7	**26.7**	3.4	16.7	16.7	20.7
gpt-realtime-mini	20.0	41.9	13.3	3.4	20.0	13.3	18.7
qwen3.5-omni-plus	24.4	19.4	10.0	0.0	6.7	16.7	10.7
grok-voice-think	15.6	12.9	13.3	0.0	6.7	13.3	9.3
gemini-3.1-flash	3.3	6.5	0.0	0.0	0.0	0.0	1.3

Table 4: Per-suite pass rate (%) of gpt-realtime-1.5 with each harness on the full benchmark. Parentheses give the change from no harness, with gains in green and losses in red. No harness improves every column.

		Proactive			Interruption	
Harness	**Reactive**	**Strong**	**Medium**	**Weak**	**Speech**	**Tool**
No harness	41.9	20.9	19.7	89.0	32.2	30.0
Transcript rewrite	46.0 (+4.1)	22.5 (+1.6)	17.5 (−2.2)	95.0 (+6.0)	18.9 (−13.3)	46.2 (+16.2)
Audio replay	49.0 (+7.1)	17.5 (−3.4)	22.5 (+2.8)	90.0 (+1.0)	27.0 (−5.2)	44.0 (+14.0)
Interruption lock	47.5 (+5.6)	14.7 (−6.2)	17.5 (−2.2)	94.7 (+5.7)	29.2 (−3.0)	57.7 (+27.7)

correction, they acknowledge the change but keep waiting on the dangling call or reuse its old argument (Figure 4b). Once an intent is established, an interruption does not reliably replace it.

Pending calls are not tracked. The Priority subtypes require the model to set the dangling call aside, complete the inserted task, and then resolve the original call. This is where performance collapses. When the result never returns, no model passes more than 10.3% of tasks. When it returns after the inserted task, models often re-issue a call whose result they already have (Figure 4c). The speech phase shows the same weakness: its hardest subtype is redirection, which also requires resuming a suspended task (at most 16.7%; Appendix Table 23). Models appear to overwrite their context rather than keep track of which calls are still pending.

5.4 Latency

Every model responds more slowly than conversation requires (Figure 2b). Median TTFT ranges from 1.6 s (grok-voice-think) to 3.1 s (gemini-3.1-flash), above both the ≈200 ms gap between human turns (Stivers et al., 2009) and the 700–1000 ms endpointing thresholds of current spoken dialogue systems (Skantze, 2021). The fastest median TTFC, 907 ms for gpt-realtime-1.5, is still far from seamless tool use (Appendix F.7).

5.5 No Static Harness Improves All Three Suites

Test-time harnesses substantially improve agent performance on other benchmarks (Yang et al., 2024; Xia et al., 2024). We test whether they can close the gaps above with three harnesses, each supplying a signal the model does not observe directly (Appendix G). *Transcript rewrite* restores truncated IDs and phone numbers in booking, cancellation, and payment calls from the API's running transcript of the conversation. *Audio replay* applies the same repair to a second transcript, obtained by re-transcribing the user's audio with a batch transcriber. *Interruption lock* switches the session to a cautious prompt after the user's first interruption keyword (e.g., "scratch that"); the prompt makes the latest statement authoritative and holds back committing calls until their arguments are restated. Two further harnesses that act on intent cues show the same pattern (Appendix G.6).

Table 4 reports the results for gpt-realtime-1.5. Every harness improves Reactive (by 4–7 pp) and tool-phase Interruption (by 14–28 pp), and every harness loses ground elsewhere: transcript rewrite costs 13 pp on speech-phase Interruption, audio replay 5 pp there, and interruption lock 6 pp on Strong. No harness matches or beats the baseline on every column.

Each harness applies one global policy, which is right for the failure it targets and wrong elsewhere. The transcript rewrite repairs misheard arguments, but the same rule overwrites arguments that the user has just corrected or redirected, which hurts speech-phase Interruption. The interruption lock catches tool-phase interruption keywords, but the same keywords also occur when users commit to an action on Strong tasks, where the cautious prompt then holds the call back. Choosing the right policy requires understanding the dialogue in context, which is the capability the benchmark measures. Stronger harnesses replace static rules with learned controllers or auxiliary orchestration (Liu et al., 2026; Bholani, 2026; Chen et al., 2025), but these change the system under test from the model to the model plus an auxiliary system, and training-time methods (Liu et al., 2025; Wang et al., 2026) fall outside the test-time setting. Within that setting the trade-off is structural, which suggests that the remaining gap calls for better models rather than better wrappers.

6 Discussion

Implications. The three bottlenecks call for different remedies. For perception, what matters is the recognition of short, high-information strings such as names, phone numbers, and IDs, not overall word error rate: one wrong character in such a string yields a call that is well formed but wrong. Re-checking these values before a committing call helps, as the 7.1 pp Reactive gain from audio replay shows. For Proactive tasks, models already detect intent, so progress depends on follow-through; training and evaluation should reward completing every call that a request implies, not merely acting on it. For interruptions, the interruption lock's 27.7 pp gain on tool-phase tasks shows that making an interruption explicit helps a model resolve a pending call, and its losses elsewhere show that a fixed rule cannot decide when to apply it. Tracking pending calls explicitly, in the model or in the streaming protocol, is a direct target for the failures we observe.

Limitations. The tasks are controlled scenarios rather than open-ended deployments. User turns are clean, scripted speech in cloned voices, so robustness to spontaneous speech, disfluency, and background noise is untested. The tasks are in English, with Chinese personal names as the main stress test for proper-noun recognition; other languages, accents, and code-switching are left to future work. Each task is run once per model, and providers do not offer snapshot-level version pinning, so reruns may differ as models are updated. Provider APIs also differ: to call tools in audio sessions, Gemini needed renamed functions, simplified parameter types, and thinking disabled (Appendix D.4), which may understate its capability. Finally, the harness study covers only gpt-realtime-1.5.

7 Conclusion

We introduced Real-Time Audio Tool Bench, a benchmark that evaluates tool calling in live, multi-turn voice interaction with deterministic scoring and failure attribution. Across six voice-native models, each suite exposes a different weakness: perception limits Reactive tasks, execution rather than intent recognition limits Proactive tasks, and failure to track pending calls limits recovery from interruptions. Static harnesses shift errors between suites rather than removing them. We release the benchmark, the evaluation framework, and all evaluation traces to support work on reliable real-time voice agents.

AI USE STATEMENT

In this work, we used ElevenLabs voice cloning to render the user audio in voices cloned from Mozilla Common Voice speakers (Appendix D.4). The audio-replay harness uses `gpt-4o-transcribe` as part of the method under study (Appendix G.4). Every task passes rule-based verification before inclusion, experts completed a stratified 200-task subset under the live interface (§3.2), and scoring is fully rule-based, with no LLM judge. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Rosana Ardila, Megan Branson, Kelly Davis, Michael Kohler, Josh Meyer, Michael Henretty, Reuben Morais, Lindsay Saunders, Francis Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference (LREC)*, pp. 4218–4222, 2020.

Siddhant Arora, Zhiyun Lu, Chung-Cheng Chiu, Ruoming Pang, and Shinji Watanabe. Talking turns: Benchmarking audio foundation models on turn-taking dynamics. In *International Conference on Learning Representations (ICLR)*, 2025.

Kinjal Basu, Ibrahim Abdelaziz, Kiran Kate, Mayank Agarwal, Maxwell Crouse, Yara Rizk, Kelsey Bradford, Asim Munawar, Sadhana Kumaravel, Saurabh Goyal, Xin Wang, Luis A. Lastras, and Pavan Kapanipathi. NESTFUL: A benchmark for evaluating LLMs on nested sequences of API calls. *arXiv preprint arXiv:2409.03797*, 2024.

Neeraj Bholani. Graph-based self-healing tool routing for cost-efficient LLM agents. *arXiv preprint arXiv:2603.01548*, 2026.

Arthur Chen, Zuxin Liu, Jianguo Zhang, Akshara Prabhakar, Zhiwei Liu, Shelby Heinecke, Silvio Savarese, Victor Zhong, and Caiming Xiong. Grounded test-time adaptation for LLM agents via environment interaction. *arXiv preprint arXiv:2511.04847*, 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.

Cheng-Han Chiang, Xiaofei Wang, Linjie Li, Chung-Ching Lin, Kevin Lin, Shujie Liu, Zhendong Wang, Zhengyuan Yang, Hung yi Lee, and Lijuan Wang. Shanks: Simultaneous hearing and thinking for spoken language models, 2025. URL `https://arxiv.org/abs/2510.06917`.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi: 10.1177/001316446002000104.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks, 2025. URL `https://arxiv.org/abs/2502.08235`.

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.

ElevenLabs. AI voice cloning. `https://elevenlabs.io/voice-cloning`, 2026.

Guangyu Feng, Huanzhi Mao, Prabal Dutta, and Joseph E. Gonzalez. Concurrency without model changes: Future-based asynchronous function calling for llms, 2026. URL `https://arxiv.org/abs/2605.15077`.

Gemini Team, Google. Gemini: A family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models. In *Findings of the Association for Computational Linguistics: ACL*, 2024.

Will Held, Michael J. Ryan, Aditya Shrivastava, Ali Sartaz Khan, Caleb Ziems, Ella Li, Martijn Bartelds, Michael Sun, Tan Li, Woody Gan, and Diyi Yang. CAVA: Comprehensive assessment of voice assistants. https://talkarena.org/cava, 2025.

Coleman Hooper, Minwoo Kang, Suhong Moon, Nicholas Lee, Eric Wen, John Wawrzynek, Michael W. Mahoney, Yakun Sophia Shao, Amir Gholami, and Kurt Keutzer. Speculative interaction agents: Building real-time agents with asynchronous i/o and speculative tool calling, 2026. URL https://arxiv.org/abs/2605.13360.

Chien-yu Huang, Ke-Han Lu, Shih-Heng Wang, Chi-Yuan Hsiao, Chun-Yi Kuan, Haibin Wu, Siddhant Arora, Kai-Wei Chang, Jiatong Shi, Yifan Peng, Roshan Sharma, Shinji Watanabe, Bhiksha Ramakrishnan, Shady Shehata, and Hung-yi Lee. Dynamic-SUPERB: Towards a dynamic, collaborative, and comprehensive instruction-tuning benchmark for speech. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024.

Dhruv Jain, Harshit Shukla, Gautam Rajeev, Ashish Kulkarni, Chandra Khatri, and Shubham Agarwal. VoiceAgentBench: Are voice assistants ready for agentic tasks? *arXiv preprint arXiv:2510.07978*, 2025.

Mayank Kulkarni, Vittorio Mazzia, Judith Gaspers, Chris Hench, and Jack FitzGerald. MASSIVE-agents: A benchmark for multilingual function-calling in 52 languages. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 20193–20215, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-emnlp.1099. URL https://aclanthology.org/2025.findings-emnlp.1099/.

Md Tahmid Rahman Laskar, Xue-Yong Fu, Seyyed Saeed Sarfjoo, Quinten McNamara, Jonas Robertson, and Shashi Bhushan TN. From text to voice: A reproducible and verifiable framework for evaluating tool calling llm agents, 2026. URL https://arxiv.org/abs/2605.15104.

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*, 2025a.

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, Qinsi Wang, Yudong Liu, Hengfan Zhang, Jayakumar Subramanian, Nikos Vlassis, Hai Helen Li, and Yiran Chen. Voice evaluation of reasoning ability: Diagnosing the modality-induced performance gap. *arXiv preprint arXiv:2509.26542*, 2025b.

Boyan Liu, Gongming Zhao, and Hongli Xu. Utility-guided agent orchestration for efficient LLM tool use. *arXiv preprint arXiv:2603.19896*, 2026.

Licheng Liu, Zihan Wang, Linjie Li, Chenwei Xu, Yiping Lu, Han Liu, Avirup Sil, and Manling Li. A simple "try again" can elicit multi-turn LLM reasoning. *arXiv preprint arXiv:2507.14295*, 2025.

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.

OpenAI. Introducing the Realtime API. https://openai.com/index/introducing-the-realtime-api/, 2024.

Ramit Pahwa, Apoorva Beedu, Parivesh Priye, Rutu Gandhi, Saloni Takawale, Aruna Baijal, and Zengli Yang. Audio2tool: Bridging spoken language understanding and function calling. *arXiv preprint arXiv:2604.22821*, 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 (ICML)*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

Soham Ray, Keshav Dhandhania, Victor Barres, and Karthik Narasimhan. τ-Voice: Benchmarking full-duplex voice agents on real-world domains. *arXiv preprint arXiv:2603.13686*, 2026.

Gabriel Skantze. Turn-taking in conversational systems and human-robot interaction: A review. *Computer Speech & Language*, 67:101178, 2021. doi: 10.1016/j.csl.2020.101178.

Ivan Solovyev and Shrestha Basu Mallick. Gemini 2.0: Level up your apps with real-time multimodal interactions. `https://developers.googleblog.com/en/gemini-2-0-level-up-your-apps-with-real-time-multimodal-interactions/`, 2024.

Tanya Stivers, N. J. Enfield, Penelope Brown, Christina Englert, Makoto Hayashi, Trine Heinemann, Gertie Hoymann, Federico Rossano, Jan Peter de Ruiter, Kyung-Eun Yoon, and Stephen C. Levinson. Universals and cultural variation in turn-taking in conversation. *Proceedings of the National Academy of Sciences*, 106(26):10587–10592, 2009. doi: 10.1073/pnas.0903616106.

Sidharth Surapaneni, Hoang Nguyen, Jash Mehta, Aman Tiwari, Oluwanifemi Bamgbose, Akshay Kalkunte, Sai Rajeswar, and Sathwik Tejaswi Madhusudhan. AU-Harness: An open-source toolkit for holistic evaluation of audio LLMs. *arXiv preprint arXiv:2509.08031*, 2025.

Bin Wang, Xunlong Zou, Geyu Lin, Shuo Sun, Zhuohan Liu, Wenyu Zhang, Zhengyuan Liu, AiTi Aw, and Nancy F. Chen. AudioBench: A universal benchmark for audio large language models. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025.

Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, Yiping Lu, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN-2: Reasoning collapse in agentic RL. *arXiv preprint arXiv:2604.06268*, 2026.

Daud Waqas, Aaryamaan Golthi, Erika Hayashida, and Huanzhi Mao. Assertion-conditioned compliance: A provenance-aware vulnerability in multi-turn tool-calling agents, 2026. URL `https://arxiv.org/abs/2512.00332`.

Boyong Wu, Chao Yan, Chen Hu, Cheng Yi, Chengli Feng, Fei Tian, Feiyu Shen, Gang Yu, Haoyang Zhang, Jingbei Li, et al. Step-audio 2 technical report. *arXiv preprint arXiv:2507.16632*, 2025.

xAI. Grok voice think fast 1.0. `https://x.ai/news/grok-voice-think-fast-1`, 2026.

Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Agentless: Demystifying LLM-based software engineering agents. *arXiv preprint arXiv:2407.01489*, 2024.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, Bin Zhang, Xiong Wang, Yunfei Chu, and Junyang Lin. Qwen2.5-Omni technical report. *arXiv preprint arXiv:2503.20215*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. ToolHop: A query-driven benchmark for evaluating large language models in multi-hop tool use. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2025.

Aohan Zeng, Zhengxiao Du, Mingdao Liu, Kedong Wang, Shengmin Jiang, Lei Zhao, Yuxiao Dong, and Jie Tang. GLM-4-Voice: Towards intelligent and human-like end-to-end spoken chatbot. *arXiv preprint arXiv:2412.02612*, 2024.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. ComplexFuncBench: Exploring multi-step and constrained function calling under long-context scenario. *arXiv preprint arXiv:2501.10132*, 2025.

A CODEBASE

The codebase is at: https://anonymous.4open.science/r/audio-tool-bench-FC72/

B BENCHMARK SPECIFICATION

This section gives the static benchmark specification: the suite-level task distribution (Appendix B.1), the complete tool catalog (Appendix B.2), and the tool dependency graph (Appendix B.3).

B.1 TASK DISTRIBUTION

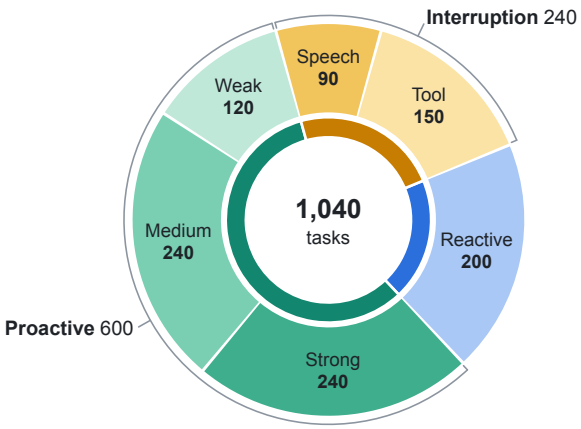


Figure 5: Task distribution across the 1040-task benchmark: suites in the inner ring, and Proactive bands and Interruption phases in the outer ring, labelled with task counts.

B.2 COMPLETE TOOL CATALOG

Real-Time Audio Tool Bench includes 44 functions across 7 domains. All tools follow the OpenAI function-calling JSON Schema and return fixed mock data for reproducibility. Tables 5–12 list every tool with its complete parameter definitions.

Table 5: Travel Booking tools (12 functions).

Tool	Description	Parameters
`search_flights`	Search flights between cities	`origin`* (str), `destination`* (str), `date` (str), `cabin_class` (str), `max_price` (num)
`book_flight`	Book a flight	`flight_no`* (str), `passenger_name` (str), `date`* (str)
`search_trains`	Search train tickets between cities	`origin`* (str), `destination`* (str), `date` (str), `train_type` (str), `seat_class` (str)
`book_train`	Book a train ticket	`train_no`* (str), `date`* (str), `passenger_name`* (str), `seat_class`* (str), `id_number` (str)
`search_hotels`	Search hotels in a city	`city`* (str), `checkin_date` (str), `nights` (int), `location` (str), `min_stars` (int), `max_price` (num)
`book_hotel`	Book a hotel room	`hotel_name`* (str), `checkin_date`* (str), `nights`* (int), `guest_name`* (str)
`search_restaurants`	Search restaurants	`city`* (str), `cuisine` (str), `location` (str)
`book_restaurant`	Make a restaurant reservation	`restaurant_name`* (str), `date`* (str), `time`* (str), `party_size`* (int)
`search_cars`	Search rental cars	`city`* (str), `car_type` (str), `pickup_date` (str), `days` (int), `max_price` (num)
`book_car`	Book a rental car	`car_model`* (str), `pickup_date`* (str), `days`* (int), `renter_name`* (str), `with_insurance` (bool)
`search_attractions`	Search tourist attractions	`city`* (str), `attraction_type` (str), `max_price` (num)
`book_attraction_ticket`	Book attraction tickets	`attraction_name`* (str), `date`* (str), `time_slot`* (str), `visitor_name`* (str), `ticket_count`* (int), `id_number` (str)

* = required parameter.

B.3 TOOL DEPENDENCY GRAPH

Tool dependencies form a directed graph with three edge types. *Resource* edges (15) encode the strongest constraint: each search tool returns an entity ID that its paired book tool consumes as input,

Table 6: Entertainment tools (6 functions).

Tool	Description	Parameters
search_movies	Search movies and showtimes	city* (str), movie_name (str), cinema (str), genre (str), date (str)
book_movie_ticket	Buy movie tickets	movie_name* (str), cinema* (str), date* (str), showtime* (str), seat_count* (int), seat_preference (str), ticket_type (str)
search_shows	Search concerts and theater	city* (str), show_name (str), show_type (str), venue (str), date (str), max_price (num)
book_show_ticket	Buy show tickets	show_name* (str), venue* (str), date* (str), seat_zone* (str), ticket_count* (int), buyer_name* (str), phone* (str)
search_sports_events	Search sports events	city* (str), sport_type (str), team (str), venue (str), date (str), max_price (num)
book_sports_ticket	Buy sports event tickets	event_name* (str), venue* (str), date* (str), seat_zone* (str), ticket_count* (int), buyer_name* (str), phone* (str), id_number* (str)

* = required parameter.

Table 7: Life Services tools (5 functions).

Tool	Description	Parameters
search_restaurants_delivery	Search delivery restaurants	city* (str), cuisine (str), max_delivery_time (int), max_delivery_fee (num)
place_food_order	Place a delivery order	restaurant_name* (str), items* (str), delivery_address* (str), phone (str), note (str)
track_package	Track package delivery status	tracking_number* (str), company (str)
search_home_services	Search home services	city* (str), service_type (str), max_price (num), time_preference (str)
book_home_service	Book a home service	service_name* (str), date* (str), time_slot* (str), address* (str), phone (str), special_requirements (str)

* = required parameter.

Table 8: Healthcare tools (3 functions).

Tool	Description	Parameters
search_doctors	Search doctors	city* (str), hospital (str), department (str), specialty (str), date (str)
book_appointment	Book a medical appointment	doctor_name* (str), hospital* (str), department* (str), date* (str), time_slot* (str), patient_name (str), patient_phone (str), id_number (str), symptoms (str)
search_medicine	Search medicine	medicine_name (str), category (str), indication (str), max_price (num), prescription_required (bool)

* = required parameter.

Table 9: Education tools (5 functions).

Tool	Description	Parameters
search_courses	Search courses	keyword (str), category (str), instructor (str), max_price (num), level (str), has_vacancy (bool)
enroll_course	Enroll in a course	course_id* (str), student_name (str), student_phone (str), student_email (str), payment_method (str)
search_books	Search library books	keyword (str), category (str), author (str), only_available (bool)
reserve_book	Reserve a book	book_id* (str), reader_id (str), reader_name (str), pickup_date (str)
renew_book	Renew a borrowed book	loan_id* (str)

* = required parameter.

Table 10: Financial tools (5 functions).

Tool	Description	Parameters
check_balance	Check account balance	account_number* (str)
get_transaction_history	Query transaction history	account_number* (str), start_date (str), end_date (str), transaction_type (str), limit (int)
transfer_money	Transfer money	from_account (str), to_account (str), amount* (num), payee_name* (str), remark (str), transfer_type (str)
list_bills	Query bill list	bill_type (str), status (str), start_date (str), end_date (str)
pay_bill	Pay a bill	bill_id* (str), payment_account (str), auto_pay (bool)

* = required parameter.

Table 11: Transportation tools (5 functions).

Tool	Description	Parameters
request_ride	Request a ride	pickup_location* (str), dropoff_location* (str), car_type* (str), passenger_count (int), scheduled_time (str), passenger_phone (str), special_requirements (str)
check_ride_status	Query ride status	ride_id* (str)
cancel_ride	Cancel a ride	ride_id* (str), reason (str)
search_parking	Search parking lots	city* (str), location (str), max_price_per_hour (num), spot_type (str), only_available (bool), max_distance (num)
reserve_parking_spot	Reserve a parking spot	parking_id* (str), parking_lot_name (str), start_time* (str), duration_hours* (num), license_plate* (str), spot_type (str), driver_phone* (str)

* = required parameter.

Table 12: Utility tools (3 functions). These general-purpose tools are registered alongside domain tools during evaluation but are never targeted by any task; they serve as distractors that increase tool-selection difficulty.

Tool	Description	Parameters
web_search	Search the web	query* (str), num_results (int)
fetch_url	Fetch webpage content	url* (str)
extract_info	Extract structured info from text	text* (str), fields* (str)

* = required parameter.

so the book call must follow the search. *Temporal* edges (8) capture logical ordering without ID passing, e.g., booking a flight before a hotel to determine check-in time. *Alternative* edges (2 pairs, bidirectional) connect functionally equivalent pairs such as flights and trains.

Table 13: Resource edges. Each search tool returns an entity ID consumed by its paired book tool.

Search Tool	Book Tool	ID Field
search_hotels	book_hotel	hotel_id
search_flights	book_flight	flight_id
search_trains	book_train	train_id
search_restaurants	book_restaurant	restaurant_id
search_restaurants_delivery	place_food_order	delivery_id
search_movies	book_movie_ticket	movie_id
search_shows	book_show_ticket	show_id
search_attractions	book_attraction_ticket	attraction_id
search_sports_events	book_sports_ticket	event_id
search_cars	book_car	car_id
search_doctors	book_appointment	doctor_id
search_home_services	book_home_service	service_id
search_courses	enroll_course	course_id
search_books	reserve_book	book_id
search_parking	reserve_parking_spot	parking_id

Table 14: Temporal edges. Logical ordering constraints without ID passing.

Source	Target	Rationale
book_flight	book_hotel	Flight arrival determines hotel check-in
book_train	book_hotel	Train arrival determines hotel check-in
book_hotel	search_restaurants	Find restaurants near the hotel
book_hotel	search_attractions	Find attractions near the hotel
book_movie_ticket	search_restaurants	Dine before or after a movie
book_show_ticket	search_restaurants	Dine before or after a show
book_car	search_parking	Find parking after renting a car
book_appointment	search_medicine	Buy medicine after a doctor visit

Table 15: Alternative edges (bidirectional). Functionally equivalent tool pairs.

Tool A	Tool B	Rationale
`search_flights`	`search_trains`	Both are intercity transport
`search_restaurants`	`search_restaurants_delivery`	Dine-in vs. delivery

C Task Construction Pipeline

This section describes the full task-construction pipeline, from graph sampling (Appendix C.1) and transcript synthesis (Appendix C.2) to validation (Appendix C.3), timing metadata (Appendix C.4), and concrete examples (Appendix C.5).

C.1 Graph Sampling Strategies

The first stage of task generation samples a valid tool chain from the dependency graph. Sampling relies on two core algorithms and a post-processing step.

Chain Walk. A directed random walk on the dependency graph produces a linear tool chain. The walk starts from a search tool (75% probability) or a standalone tool (25%), never from a book tool. At each step, the walker queries downstream neighbors along Resource and Temporal edges, filters out visited nodes, and randomly selects one. If no downstream neighbor remains, an unvisited search tool is spliced in as a new starting point. The walk continues until the chain reaches the target length.

Template Sampling. Six predefined scenario templates provide DAG-structured tool combinations (Table 16). Each template consists of several branches, where each branch is a search–book pair or a standalone tool. The sampling process selects a template with enough branches, randomly picks the required number, and flattens them into a tool list. For odd target counts, an additional standalone or search tool is appended.

Table 16: Chain templates for DAG sampling.

Template	Branches
business_trip	[search_flights → book_flight], [search_hotels → book_hotel], [search_restaurants → book_restaurant]
travel	[search_trains → book_train], [search_hotels → book_hotel], [search_attractions → book_attraction_ticket]
date	[search_movies → book_movie_ticket], [search_restaurants → book_restaurant]
entertainment	[search_shows → book_show_ticket], [search_restaurants → book_restaurant]
driving	[search_cars → book_car], [search_parking → reserve_parking_spot]
healthcare	[search_doctors → book_appointment], [search_medicine]

Dispatch by tool count. The algorithm selection depends on the target number of tools: 1 tool uses direct random selection (70% search, 30% standalone); 2 tools selects a search–book pair via Resource edges; 3 tools splits evenly between Chain Walk and Template Sampling; 4–8 tools favors Template Sampling (70%) with Chain Walk as fallback (30%). Batch generation draws the target tool count uniformly from 1 to 8.

Post-processing. Regardless of the algorithm used, the final tool list is validated: if any book tool's required search tool is missing or appears after it, the search tool is automatically inserted. This enforces Resource dependency constraints for generated chains.

C.2 Schema-Grounded Transcript Synthesis

Parameter values for every task are first sampled from the deterministic mock-data store along the tool dependency graph; the transcript is then synthesized to carry exactly those pre-fixed values. We use GPT-5.2 (temperature 0.7) for the surface realization step, conditioned on the sampled tool chain and the fixed parameter set: given the parameters, the model writes the natural multi-turn user utterances that would deliver those values across 4–7 turns, with at most 3 new parameters per turn.

The synthesis is constrained by the upstream schema and parameters; subsequent validation checks for invented entities, IDs, or scenarios. The prompt enforces five constraints:

1. **Natural spoken language.** Users talk to a voice assistant, not type commands. Parameter names and field names must never appear; instead, values are embedded in natural phrases (e.g., "Book me a table at Chuanfu Laozao" rather than "restaurant_name: Chuanfu Laozao").
2. **Self-contained references.** Entities must be mentioned by name. Vague references such as "that one" or "the first result" are prohibited, since the transcript must be interpretable without access to search results.
3. **Additive phrasing.** When parameters are spread across turns, subsequent turns add information rather than modify it. Phrases like "for 3 nights" and "checking in March 15th" are preferred over "update the check-in date to March 15th," which implies changing an existing record.
4. **Gradual information disclosure.** Parameters are distributed across turns to simulate realistic multi-turn information gathering, with at most 3 new parameters per turn.
5. **Search before book.** Search parameters (city, origin, destination) must appear in earlier turns than book parameters (entity name, buyer name), reflecting the natural flow of first searching then booking.

Proactive variants. Proactive tasks use band-specific transcript styles that vary the explicitness of user intent:

- **Strong:** The user expresses clear intent with explicit confirmation phrases ("go ahead and book," "lock it in," "reserve it under my name"). All required parameters are provided naturally across turns. The model should recognize the intent and execute the requested tools.
- **Medium:** The user expresses moderate, casual intent. Information is embedded in conversational phrasing ("been meaning to get around to," "it'd be nice to," "kinda tempted to") with eventual confirmation. The model should act on the request.
- **Weak:** The user employs narrative framing (reminiscence, hypothetical, third-person storytelling). Tool-relevant entities and parameter values may appear as distractors, but no second-person verbs directed at the AI ("can you," "help me") and no execution verbs ("book it," "go ahead") are present. The model should *not* call any tools.

C.3 Two-Layer Verification

Every generated task passes through two verification layers before inclusion in the benchmark.

Rule Checker (6 sequential checks). Each check short-circuits on failure.

1. *JSON format*: required fields (`task_id`, `transcript`, `expected_tools`) exist and are non-empty.
2. *Parameter completeness*: every required parameter for each expected tool (per its JSON Schema) is present and non-null.
3. *ID chain integrity*: every book tool's upstream search tool appears in `expected_tools` at a strictly earlier position.
4. *Mock data consistency*: every entity ID in book tool parameters (e.g., `hotel_id`, `flight_id`) exists in the mock data store. This check covers all 15 Resource edge pairs.
5. *Date logic*: `book_hotel.nights` > 0, `book_car.days` > 0, etc.
6. *Search execution*: all search tools are actually executed against mock data. The check verifies that (a) each search returns non-empty results, and (b) every entity ID referenced by a downstream book tool appears in those results. This is the strictest check for search-then-book ID-chain feasibility in the mock data.

Model Checker (optional). The full transcript is sent to GPT-4o-mini, which predicts which tools should be called. The predicted set is compared against `expected_tools` via set equality. Disagreement flags potential ambiguity in the transcript.

C.4 COMPUTING `INFO_COMPLETE_TURN`

The `info_complete_turn` field marks the earliest turn at which all user-provided parameters for a tool have appeared, serving as the ground truth for premature-call detection. The computation proceeds in three steps:

1. **Identify tracked parameters.** For each tool, include a parameter only if its value is non-null, it is not a Resource ID field (e.g., `hotel_id`, which comes from search results rather than user speech), and it is not boolean-typed. Boolean fields are excluded from `info_complete_turn` because, in this tool catalog, they are optional filters with implicit defaults rather than required user-spoken slots; when a boolean is present in the expected call, its value is still checked during parameter matching.
2. **Scan the transcript.** Iterate over turns and read the `contains_params` annotation (e.g., `book_hotel.guest_name=Li Na`). Record the first turn in which each tracked parameter appears.
3. **Take the maximum.** $\texttt{info_complete_turn}[\text{tool}] = \max(\text{first-appearance turns of all tracked parameters})$.

For example, if `book_restaurant` requires `contact_name` (first appears in turn 5) and `phone` (turn 6), then $\texttt{info_complete_turn} = 6$. Any call to `book_restaurant` before turn 6 is a premature call.

C.5 TASK EXAMPLES

This section presents one representative example from each task suite. All examples are drawn from the actual generated task data.

C.5.1 REACTIVE TASK

The user issues explicit commands and provides parameters across 6 turns. The model must call tools only after all required information has been gathered.

> **Turn 1:** "Search for movies playing in Beijing."
> **Turn 2:** "I want to see The Wandering Earth 3. Book 1 ticket at Bona International for the 21:50 show."
> **Turn 3:** "Put the movie ticket under the name Chen Ming."
> **Turn 4:** "Now find restaurants in Shanghai."
> **Turn 5:** "Reserve a table at Chuanfu Laozao on 2026-03-02 at 11:30 for 4 people."
> **Turn 6:** "Use the contact name Li Na, and the phone number is 13851574217."

Expected tools: `search_movies`(city=Beijing) → `book_movie_ticket`(cinema=Bona International, showtime=21:50, seat_count=1, buyer_name=Chen Ming) → `search_restaurants`(city=Shanghai) → `book_restaurant`(date=2026-03-02, time=11:30, party_size=4, contact_name=Li Na, phone=13851574217).

info_complete_turn: search_movies=1, book_movie_ticket=3, search_restaurants=4, book_restaurant=6. Resource IDs (`movie_id`, `restaurant_id`) come from search results and are excluded from info_complete_turn computation.

C.5.2 PROACTIVE STRONG

The user expresses clear intent with explicit confirmations. The model should recognize the intent and call the requested tools.

> **Turn 1:** "I'm in Beijing tonight and kinda feel like catching a movie instead of going straight home."

> **Turn 2:** "If The Wandering Earth 3 is playing, I'm totally down for it."
> **Turn 3:** "Bona International works for me, and 21:50 is basically perfect."
> **Turn 4:** "Yeah, go ahead and lock in 1 seat under Chen Ming."
> **Turn 5:** "Also, find me restaurants in Shanghai. I'll be there tomorrow."
> **Turn 6:** "Chuanfu Laozao, 2026-03-02, 11:30, party of 4. Contact name Li Na, phone 13851574217."

Expected tools: `search_movies`(city=Beijing) → `book_movie_ticket`(cinema=Bona International, showtime=21:50, seat_count=1, buyer_name=Chen Ming) → `search_restaurants`(city=Shanghai) → `book_restaurant`(date=2026-03-02, time=11:30, party_size=4, contact_name=Li Na, phone=13851574217).

Scoring: The model passes if it calls all four expected tools with correct parameters.

C.5.3 PROACTIVE MEDIUM

The user expresses moderate, casual intent; information is embedded in conversational phrasing with eventual confirmation. The model should act on the request.

> **Turn 1:** "I've been wiped out lately. I'll be in Beijing for a bit, and it'd be kind of nice to just unwind with a movie or something."
> **Turn 2:** "For the movie, if anything like The Wandering Earth 3 is on, I'd totally be up for it, something around 21:50."
> **Turn 3:** "Yeah, that works. Go ahead and lock in 1 seat at Bona International, under Chen Ming."
> **Turn 4:** "Also, I'm heading to Shanghai right after, and I keep thinking we should do a lunch thing there… Chuanfu Laozao, 4 of us, 2026-03-10 is probably the day, around 11:30."
> **Turn 5:** "Use Li Na as the contact, phone 13851574217."

Expected tools: `search_movies`(city=Beijing) → `book_movie_ticket` → `search_restaurants`(city=Shanghai) → `book_restaurant`.

Scoring: The model passes if it calls all expected tools with correct parameters. The casual tone does not reduce the user's intent, and eventual confirmations ("go ahead," "that works") signal genuine requests.

C.5.4 PROACTIVE WEAK

The transcript contains tool-relevant parameter values embedded in past-tense reminiscence. No tools should be called.

> **Turn 1:** "Last year I was bouncing between cities a lot. Beijing was the one where we ended up talking about what movies were even playing at the time."
> **Turn 2:** "I remember my buddy Chen Ming went to see The Wandering Earth 3 at Bona International, and he was raving about the screen the next day."
> **Turn 3:** "He said the showtime was 21:50, and he just grabbed 1 seat since nobody else could make it."
> **Turn 4:** "Then a couple weeks later I was in Shanghai, and everyone kept arguing about where to eat there like it was a sport."

Expected tools: none. The user is recounting past experiences in third person, and no request or action is implied. The speech contains the same entities and parameter values as a Strong task (Beijing, The Wandering Earth 3, Bona International, 21:50, Chen Ming, Shanghai) but framed as reminiscence. Any tool invocation is a false positive.

C.5.5 INTERRUPTION: CORRECTION

The user interrupts the model mid-response to correct a parameter value. The framework monitors the model's streaming transcript for the keyword "D231" and sends `response.cancel` when detected.

> **Turn 1:** "Search trains from Beijing to Shanghai."
> **Turn 2 `:** "Wait, not D231, I meant D321."

Turn 3: "Book train D321 on 2026-03-18 for Zhang Wei."
Turn 4: "First Class seat."
Turns 5–8: (continue with attraction and hotel bookings)

Interrupt trigger: keyword "D231," timeout 5s. The model must discard D231 and use the corrected D321. If the model still books D231 after being interrupted, this is classified as a premature call on the cancelled intent.

C.5.6 INTERRUPTION: CANCELLATION

The user cancels a previously announced task.

Turn 1: "Search trains from Beijing to Shanghai. Also, later help me with an attraction plan."
Turns 2–6: (complete train and hotel bookings)
Turn 7 `: "Actually forget the attraction. I don't need it."
Turn 8: "Proceed with only the train booking and the Hangzhou hotel booking."

Interrupt trigger: keyword "attraction." **Expected tools:** only `search_trains`, `book_train`, `search_hotels`, `book_hotel`. **Cancelled tools:** `search_attractions`, `book_attraction_ticket`. The model must suppress the cancelled tools despite the user having explicitly announced the intention earlier.

C.5.7 INTERRUPTION: REDIRECTION

The user inserts a new task mid-conversation, completes it, and returns to the original task.

Turn 1: "Search trains from Beijing to Shanghai."
Turn 2: "Book train G1 for Zhou Jun."
Turn 3: "Set the seat class to First Class."
Turn 4 `: "Wait, before that, search movies in Chengdu, then book movie YOLO at Wanda Cinema Jinhua."
Turn 5: "Set the showtime to 20:30 and seat count to 3."
Turn 6: "Use buyer name Li Na and pay bill BILL20260215003."
Turn 7: "OK, back to the train. Continue the booking."

Expected tools (in execution order): `search_movies` → `book_movie_ticket` → `pay_bill` → `search_trains` → `book_train`. The inserted task (movie + bill) executes first; the original task (train) resumes afterward. The model must maintain two concurrent parameter sets and correctly restore the original context when the user says "back to the train."

D EVALUATION, SCORING, AND RUNTIME

This section defines the evaluation mechanics used by the main results: parameter matching (Appendix D.1), Proactive scoring (Appendix D.2), Interruption triggering and recovery logic (Appendix D.3), and cross-provider runtime infrastructure (Appendix D.4).

D.1 PARAMETER MATCHING RULES

Comparing model-generated tool-call arguments against expected values requires type-aware matching. Voice interaction introduces natural format variation (e.g., a user says "138-1234-5678" but the expected value is "13812345678"). Our matching rules tolerate format differences but reject content errors. These rules define pass/fail; edit distance is applied only afterward to classify a failed parameter match as perception-side or reasoning-side.

D.1.1 MATCHING RULES BY PARAMETER TYPE

The matching function dispatches on parameter name. All comparisons apply after stripping leading and trailing whitespace.

Rule 1: Null handling. If either the expected or actual value is null, matching succeeds only when both are null.

Rule 2: Digit-string parameters (`phone`, `account_number`, `tracking_number`, `id_number`, `license_plate`). All non-digit characters are stripped from both values before comparison. If either value contains no digits after stripping, the comparison falls back to case-insensitive string matching. This normalizes formats like "138-1234-5678" and "13812345678" to the same representation, while still catching missing or extra digits.

Rule 3: Time parameters (`time_slot`, `time`, `departure_time`, `arrival_time`, `showtime`). Exact match, or the actual value starts with the expected value. This handles the case where the model returns a time range (e.g., "09:00–12:00") when only the start time ("09:00") is specified.

Rule 4: List parameters (`items`). Both values are split on " and " or ",", stripped, lowercased, sorted, and rejoined with commas. This makes "Kung Pao Chicken, Rice" equal to "Rice and Kung Pao Chicken."

Rule 5: Default (all other parameters). Case-insensitive exact string comparison after whitespace stripping.

D.1.2 Multiple Calls

If the model calls the same tool multiple times (e.g., retrying after a parameter mismatch), evaluation uses the **last call**'s arguments. This "last-call-wins" policy gives the model credit for self-correction.

D.1.3 Perception-Error Detection

Perception-error detection is used for failure attribution (§4) and does not affect pass/fail scoring. For each mismatched parameter, the Levenshtein edit distance between the expected and actual values is computed (for digit-string parameters, non-digit characters are stripped first).

- Edit distance ≤ 2: classified as a **perception-side error**, called an ASR error in the appendix tables. It is a *numerical-ID error* for the digit-string parameters of Rule 2 and an *entity mishearing* otherwise (e.g., "Li Na" $\rightarrow$ "Lina," distance 1).
- Edit distance > 2: classified as **parameter reasoning** (e.g., "Li Na" $\rightarrow$ "Zhang Wei," distance 7).

The threshold of 2 is based on empirical observation: common ASR errors (space merging, single-character substitution, trailing truncation) produce edit distances ≤ 2, while model reasoning errors (entirely wrong values) produce much larger distances.

D.2 Proactive Scoring Details

Proactive tasks use fully rule-based scoring. Each task specifies an `intent.strength` field (strong, medium, or weak) that determines the expected model behavior and pass criteria. No LLM judge is required.

D.2.1 Pass Criteria by Band

Strong and Medium tasks include a non-empty `expected_tools` list. The model passes if it calls *all* expected tools with parameters matching the ground truth (using the matching rules in Section D.1) and respects the timing lower bound (§C.4). Medium's timing check is relaxed to the turn at which clarification completes.

The model fails if:

- Any expected tool is missing from the model's tool calls.
- A forbidden/distractor tool is called.
- Tool parameters do not match expected values (per Section D.1).

- A tool is called before the timing lower bound (premature call).

Weak tasks have an empty `expected_tools` list. The user expresses no actionable intent (reminiscence, hypothetical, third-person narrative). The model passes if and only if it calls *no* tools. Any tool invocation, including read/search tools, is a false positive.

D.2.2 Parameter Matching

For Strong and Medium tasks, each expected tool's parameters are matched against the model's actual call using the rules in Section D.1. A tool call passes only if *all* expected parameter values match. The "last-call-wins" policy applies: if the model calls the same tool multiple times, the last call's arguments are used.

D.2.3 Aggregate Metrics

Four metrics are reported:

1. **Strong Pass Rate**: fraction of Strong tasks (240) that pass.
2. **Medium Pass Rate**: fraction of Medium tasks (240) that pass.
3. **Weak Pass Rate**: fraction of Weak tasks (120) that pass.
4. **Overall Proactive Accuracy**: mean of the three per-band pass rates.

D.3 Interruption Trigger Mechanics

Interruption tasks require precisely timed intervention: the framework must cancel the model's response mid-generation and inject the user's interruption audio. Figure 6 illustrates the two interruption phases. This section describes the trigger mechanism, fallback strategies, and race-condition handling used to make interruption timing reproducible.

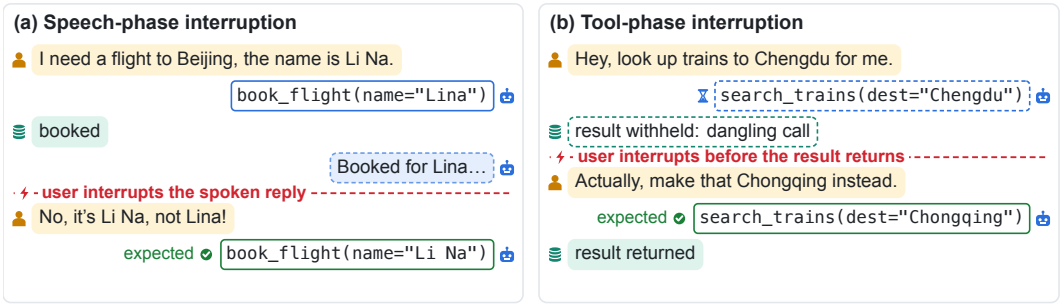


Figure 6: **Interruption phases.** (a) Speech phase: the tool call has executed and the model is delivering its spoken reply when the user corrects an argument. (b) Tool phase: the model has issued a tool call whose result has not yet returned when the user changes the request.

D.3.1 Overall Flow

Each interruption task contains one transcript turn marked `is_interruption: true`. At runtime, the framework:

1. Sends the preceding turn's audio normally (PCM16, 24 kHz, in 4800-byte chunks with 50 ms spacing).
2. Calls `create_response()` and enters an event-monitoring loop.
3. Monitors three trigger conditions in parallel (Section D.3.2).
4. On any trigger, sends `response.cancel` to abort the model's response.
5. Drains residual events until the server confirms cancellation (`response.done`).
6. Sends the interruption audio and continues the conversation.

D.3.2 TRIGGER CONDITIONS

Three conditions are monitored concurrently. Any one is sufficient to trigger the interruption.

Condition 1: Keyword detection (primary). On each `response.audio_transcript.delta` event, the new text is appended to the accumulated transcript. If the keyword appears as a substring (case-insensitive), the interruption fires. For example, in a Correction task with keyword "D231," the interruption triggers the moment the model mentions the incorrect train number. This ensures the interruption occurs at a semantically appropriate moment.

Condition 2: Audio byte fallback (2-second forced interruption). If the model has output 2 seconds of audio ($2.0 \times 24000 \times 2 = 96{,}000$ bytes for 24 kHz 16-bit mono) without the keyword appearing, the interruption fires regardless of transcript content. This handles cases where the model skips the keyword or transcript events are delayed.

Condition 3: Wall-clock timeout (5-second cap). If more than 5 seconds have elapsed since the model began speaking (first `response.audio.delta` or `response.audio_transcript.delta` event), the interruption fires. Before speech begins (e.g., during tool-call execution), the timeout is set to 30 seconds to avoid premature triggering.

In practice, the vast majority of interruptions are triggered by keyword detection, a small number by the audio byte fallback, and the wall-clock timeout is rarely reached.

D.3.3 TOOL-CALL PASS-THROUGH

The model may execute tool calls before beginning its spoken response (e.g., calling `search_trains` before announcing results). During this phase, tool calls proceed normally: results are returned to the model, a new `create_response()` is issued, and the tool-round counter increments (up to 5 rounds). Interruption triggers are inactive during this phase because the `speech_started` flag remains false until audio output begins.

D.3.4 TOOL-PHASE RESULT HOLDING

Tool-phase interruption tasks override the pass-through behavior for one selected call. When the model emits the trigger tool call, the harness executes it against the deterministic mock backend but withholds the result from the Realtime session, creating a stable dangling-call state before the user interruption is sent. The held result is then handled according to the subtype (the main text calls these subtypes Cancel, Correction, No-return, Late-return, and Early-return): `tool_cancel` and `tool_correction` discard the stale result after the interruption, `priority_no_return` never releases it, `priority_late_return` releases it after the inserted task completes, and `priority_early_return` releases it while the inserted task is still active. This server-side result holding makes the timing window deterministic rather than dependent on provider latency.

D.3.5 CANCELLATION AND DRAINING

After sending `response.cancel`, the framework drains the event pipeline:

1. Read up to 200 residual events (10-second timeout per event).
2. Continue accumulating transcript text (to record what the model said before being interrupted).
3. Wait for a `response.done` event confirming the server has stopped generation.
4. After `response.done`, wait an additional 0.5 seconds to catch any trailing error event.

D.3.6 RACE-CONDITION HANDLING

Response already finished. If the server finishes its response before `response.cancel` arrives, it returns a `response_cancel_not_active` error. The framework silently ignores this error, since the interruption is no longer needed.

Tool-call event before cancellation. The model may emit a tool-call event before the cancel takes effect. These calls are executed normally (they are already in the event pipeline), but during evaluation, tool calls made before the interruption turn are not penalized as premature calls (§D.1).

Audio and cancel sequencing. The framework sends the preceding turn's audio, commits the buffer, and calls `create_response()` before entering the monitoring loop. This ensures that cancellation only occurs during the response-generation phase, never overlapping with audio transmission.

D.3.7 Return Metadata

Each interruption step records: `interrupted` (bool), `interrupt_trigger_type` ("keyword," "forced," or "timeout"), and `partial_transcript` (the model's output before interruption). These fields are stored in the trace for post-hoc analysis.

D.4 Evaluation Infrastructure and Reproducibility

D.4.1 Model Versions and Session Configuration

Table 17 lists the exact API model identifier, output voice, input transcription model, and VAD configuration for each evaluated model. No explicit temperature, top_p, or top_k is set; all models use provider defaults. Model IDs are API-level identifiers; none of the four providers offers snapshot-level version pinning. Each task is evaluated in a single run; run-to-run variance is not measured (acknowledged in §6).

Table 17: Per-model evaluation configuration. VAD columns report `server_vad` parameters; "—" indicates Gemini uses its own endpoint detection.

Provider	Model ID	Voice	Transcription	Silence (ms)	Prefix (ms)
OpenAI	`gpt-realtime` / `-1.5` / `-mini`	alloy	whisper-1	800	300
xAI	`grok-3-fast`	Ara	grok-voice-listen-fast-1.0	200	300
Alibaba	`qwen3-omni-flash-realtime`	Ethan	built-in	800	500
Google	`gemini-2.5-flash-native-audio-latest`	Puck	built-in	—	—

Grok additionally appends 3 s of silence padding and forces a commit to compensate for its shorter silence threshold.

D.4.2 Cross-Provider Schema Normalization

All providers receive the same 44 tool definitions. Provider-specific adaptations:

- **OpenAI.** Native function calling; tool schemas are flattened from the Chat Completions format into the Realtime API's per-tool structure.
- **Grok.** Inherits the OpenAI wire format; replaces the input transcription model with `grok-voice-listen-fast-1.0`.
- **Qwen.** Inherits the OpenAI wire format; `prefix_padding_ms` is raised to 500 ms to match DashScope defaults.
- **Gemini.** Three workarounds are required: (1) function names are converted from `snake_case` to `camelCase` because underscores trigger a 1011 protocol error—names are converted back before scoring; (2) enum, array, and boolean parameter types are simplified to `string` to avoid 1008 schema-validation errors; (3) `thinking_budget` is set to 0 because the thinking mode disables function calling in native-audio sessions.

D.4.3 User Audio Pipeline

User-turn audio is pre-rendered with ElevenLabs voice cloning (ElevenLabs, 2026). The voice bank holds N speakers from Mozilla Common Voice (Ardila et al., 2020), and each speaker's voice is cloned from their Common Voice recordings (TODO(authors): ElevenLabs model ID and voice settings; how speakers were selected and assigned to tasks). The audio is stored as PCM16 at 24 kHz mono and cached in `data/audio/`. Caching ensures byte-identical audio across evaluation runs. During playback, audio is streamed to the model in 4 800-byte chunks at 50 ms intervals.

D.4.4 Connection and Retry

- **WebSocket connect**: 3 attempts with exponential back-off (5 s, 10 s); `open_timeout`=30 s.

- **Response timeout**: 60 s per model response; tool-phase `recv_timeout`=30 s.
- **Race conditions**: the client silently ignores `conversation_already_has_active_response`, `response_cancel_not_active`, and `response_already_done` events, which arise from asynchronous cancel/response sequencing.
- **Gemini**: SDK-level retry is disabled (`HttpRetryOptions(attempts=1)`) to prevent silent re-execution of completed turns.

D.4.5 Trace Format and Released Artifacts

Each task produces a JSON trace containing: `task_name`, `start_time`, `end_time`, a `steps` array, `summary`, and `metadata` (provider, model ID, input mode, VAD mode, user voice). Each step records `step_id`, `timestamp`, `assistant_response`, `llm_calls` (with `finish_reason` and `tool_calls`), and `tool_executions` (with `tool_name`, `arguments`, `result`, `latency_ms`). The anonymous artifact repository is available at `https://anonymous.4open.science/r/audio-tool-bench-FC72/`.

Released artifacts.

- 1040 task JSONs with transcripts, expected tool calls, parameter annotations, and `info_complete_turn` metadata.
- Pre-rendered user audio in the cloned voices (PCM16, 24 kHz) for all 1040 tasks.
- 7240 evaluation traces: 6 models $\times$ 1040 tasks (voice mode) + 5 models $\times$ 200 tasks (text mode).
- Evaluation framework source: model clients for all four providers, audio runner, mock tool executor, and scorer.
- Deterministic mock-data store (fixed tool responses).
- Analysis and plotting scripts for all tables and figures.

E Human Validation

This appendix details the human validation studies summarized in §3.2: task solvability (Appendix E.1), intent-label stability (Appendix E.2), automatic attribution reliability (Appendix E.3), and disagreement analysis (Appendix E.4).

E.1 Task Solvability Study

Protocol. We sample 200 tasks stratified by suite and subtype, and recruit two annotators who are familiar with the tool catalog but were not involved in task authoring. Annotators complete the sampled tasks under the same conditions as model evaluation: listening to the pre-rendered task audio, viewing the tool catalog, and issuing tool calls through the annotation interface. We report task-level completion over the 200-task sample. A task is marked *solvable* if the annotator can identify all expected tools and provide correct parameters from the audio alone.

Results. Human annotators successfully completed 193 of 200 tasks (96.5%). Among the seven failed tasks, one representative example is a *priority_no_return* tool-phase interruption scenario (tint_000047) where the user interrupts a dangling search call with a higher-priority request, the original tool result never returns, and the user later says "OK, continue with the flight search," requiring the annotator to re-issue the dangling tool call. This failure reflects genuine behavioral complexity (state tracking across an interruption boundary) rather than a task design flaw: the same failure mode dominates model evaluations on tool-phase Interruption tasks.

E.2 Proactive Intent Agreement

Protocol. We sample 200 Proactive tasks stratified by intent band (80 Strong, 80 Medium, 40 Weak), matching the full Proactive suite distribution. Each rater sees the first two user utterances and the available tool list, which is the same information window models receive before committing to an action. Raters independently label each task as Strong, Medium, or Weak. They receive a one-page guideline defining the three bands with examples but no access to the ground-truth labels. We recruit two human annotators.

Table 18: Proactive intent agreement on 200 tasks. κ = Cohen's kappa.

Comparison	Agreement	κ
Annotator 1 vs. Gold	80.0%	0.700
Annotator 2 vs. Gold	88.5%	0.825
Annotator 1 vs. Annotator 2	92.0%	0.876

Results. Table 18 reports pairwise agreement.

Human–human Cohen's κ (Cohen, 1960) $= 0.876$ (92.0% agreement, 184/200). All 16 disagreements fall on the Strong $\leftrightarrow$ Medium boundary; neither annotator ever confuses Strong with Weak or Medium with Weak.

Interpretation. Both annotators achieve substantial-to-strong agreement with the gold labels ($\kappa = 0.70$ and 0.83), and their mutual agreement is higher still ($\kappa = 0.88$). The Strong/Weak boundary is stable in this sample: both annotators identify all 40 Weak items correctly. Disagreements concentrate exclusively on the Strong $\leftrightarrow$ Medium boundary, reflecting genuine pragmatic ambiguity in how explicitly users signal intent through conversational phrasing.

E.3 Failure Attribution Reliability

Protocol. We validate the rule-based six-class attribution (§4) against independent human labels. From 1,940 failed traces, we draw a stratified sample of 200 across the six attribution classes, over-sampling rare classes for statistical power. The resulting rule-based strata contain 21 entity mishearing, 23 numerical-ID error, 57 missing call, 35 premature call, 44 parameter reasoning, and 20 unexpected call traces. The human annotator receives the full trace, expected tool calls, and the six-class taxonomy definitions, then independently assigns a primary failure category without seeing the rule-based label.

The six classes are those of §4. The two perception-side classes, *entity mishearing* (person names, place names, restaurant names) and *numerical-ID error* (phone numbers, ID numbers, flight numbers), are kept separate because their error signatures differ.

Results. Table 19 shows the confusion matrix between rule-based (rows) and human (columns) labels. Table 20 reports per-class precision, recall, and F1.

Table 19: Row-normalized attribution confusion matrix: rule-based labels (rows) vs. human labels (columns).

Rule \ Human	Entity	Num. ID	Missing	Premature	Param.	Unexpected
Entity mish.	**90.5%**	0.0%	0.0%	0.0%	9.5%	0.0%
Numerical ID	0.0%	**87.0%**	0.0%	0.0%	13.0%	0.0%
Missing call	0.0%	0.0%	**87.7%**	12.3%	0.0%	0.0%
Premature call	0.0%	0.0%	5.7%	**85.7%**	0.0%	8.6%
Param. reason.	11.4%	6.8%	2.3%	0.0%	**79.5%**	0.0%
Unexpected call	0.0%	0.0%	0.0%	0.0%	0.0%	**100.0%**

Table 20: Per-class precision, recall, and F1 for rule-based attribution against human labels. Macro-average F1 = 87.0%; overall accuracy = 87.0%.

Class	Precision	Recall	F1
Missing call	87.7%	94.3%	**90.9%**
Premature call	85.7%	81.1%	**83.3%**
Unexpected call	100.0%	87.0%	**93.0%**
Numerical-ID error	87.0%	87.0%	**87.0%**
Param. reasoning	79.5%	87.5%	**83.3%**
Entity mishearing	90.5%	79.2%	**84.4%**
Macro avg	**88.4%**	**86.0%**	**87.0%**

Interpretation. The three *decision-level* classes, namely missing call (F1 = 0.91), premature call (0.83), and unexpected call (0.93), are reliably detected because they have unambiguous mechanical

signatures (tool never called, called before `info_complete_turn`, or called on a no-action task). Numerical-ID error (0.87) is also well-separated: digit truncation and transposition leave clear edit-distance fingerprints. Entity mishearing (0.84) and parameter reasoning (0.83) are the main residual source of confusion. Their boundary is inherently fuzzy: when a name parameter differs by edit distance 2, it may be an ASR slip or a reasoning error depending on phonetic plausibility, which the rule-based system cannot assess. Merged into a single perception-side category, the two perception classes would reach a combined F1 of approximately 0.86.

E.4 Disagreement Analysis

Of the 26 disagreements between rule-based and human labels, most concentrate in a small number of adjacent-category boundaries:

Pattern 1: Parameter reasoning → entity or numerical-ID (8 cases). The rule-based system labels a trace as *parameter reasoning* (edit distance > 2) but the human annotator recognizes the error as a plausible speech mishearing. For example, the guest name "Zhou Jun" transcribed as "Joe Jin" has edit distance 4 (exceeding the threshold of 2) but is phonetically similar, representing a speech-perception error that the edit-distance heuristic misclassifies. Similarly, 3 cases involve numerical IDs where digit transpositions span more than 2 positions but are clearly digit-level ASR errors when examined in context.

Pattern 2: Entity or numerical-ID → parameter reasoning (5 cases). The reverse: the rule-based system flags a parameter mismatch as ASR (edit distance ≤ 2 or digit-string corruption), but the human annotator determines the model used an entirely wrong entity or transformed an identifier through downstream reasoning. For example, a model booked restaurant ID `rst_cd_001` instead of the expected `rst_cd_002`, which is an internal ID mix-up rather than a speech-perception error, despite the low edit distance.

Other patterns. Nine cases involve the missing ↔ premature boundary, where the human annotator weighted incomplete information differently from the `info_complete_turn` heuristic. Three cases involve the premature ↔ unexpected boundary, and one remaining case involves a parameter-reasoning trace that the annotator treated as a missing call.

The disagreement patterns suggest that the edit-distance threshold (2) is a reasonable but imperfect proxy for the ASR-vs-reasoning boundary. Semantic context, which the human annotator has access to but the rule-based system does not, resolves most ambiguous cases. The main-text analyses group entity mishearing and numerical-ID error as perception errors. The two classes are never confused with each other (Table 19), so the residual disagreement lies at the perception/reasoning boundary described above.

F Extended Results and Diagnostics

This section contains detailed result tables (Appendix F.1), ASR diagnostics (Appendix F.2), representative failure traces (Appendix F.3), per-tool and parameter-type analyses (Appendix F.4 and Appendix F.5), cross-model failure overlap (Appendix F.6), latency statistics (Appendix F.7), additional ablations (Appendix F.8), and behavior-vs-task gap analysis (Appendix F.8.2).

F.1 Per-Model Detailed Results

This section provides fine-grained breakdowns of the results summarized in Tables 1–3.

F.1.1 Reactive Tasks by Chain Length

Reactive tasks span a range of chain lengths, all structured as directed acyclic graphs (DAGs) over dependent tools. We partition the 200 tasks into three tiers by expected tool count: *short* ($\leq$4 tools), *medium* (5–6 tools), and *long* ($\geq$7 tools).

Table 21: Reactive pass rate by chain length tier. Tiers are partitioned by expected tool count: *short* (≤4 tools), *medium* (5–6 tools), and *long* (≥7 tools). Best per column in **bold**.

Model	Short	Medium	Long	Overall
gpt-realtime-1.5	**54.7%**	**47.7%**	**14.6%**	**41.9%**
gpt-realtime	40.6%	29.1%	12.5%	28.8%
gpt-realtime-mini	40.6%	25.6%	12.5%	27.3%
qwen3.5-omni-plus	14.1%	16.3%	6.2%	13.1%
grok-voice-think	32.8%	11.6%	4.2%	16.7%
gemini-3.1-flash	12.5%	8.1%	6.2%	9.1%

F.1.2 Proactive Action Rate

Table 22 gives the Strong-band action rates behind Figure 3. Every model acts on at least 95% of Strong tasks, so the execution gap is driven by pass rate rather than by a failure to act.

Table 22: Proactive execution gap on Strong tasks. Action rate is the fraction of tasks with at least one tool call; gap = action rate − pass rate.

Model	Action Rate	Gap
gpt-realtime-1.5	99.6%	**78.7** pp
gpt-realtime	100.0%	93.7 pp
gpt-realtime-mini	100.0%	91.6 pp
qwen3.5-omni-plus	98.8%	89.6 pp
grok-voice-think	99.6%	87.4 pp
gemini-3.1-flash	95.4%	94.6 pp

F.1.3 Speech-phase Interruption by Type

Table 23 breaks down speech-phase interruption pass rates by the three intent subtypes on the combined 90-task speech-phase benchmark.

Table 23: Speech-phase Interruption pass rate by subtype. Best per column in **bold**.

Model	Correction	Cancellation	Redirection
gpt-realtime-1.5	**29.4%**	**50.0%**	12.5%
gpt-realtime	23.5%	31.2%	**16.7%**
gpt-realtime-mini	20.6%	31.2%	4.2%
qwen3.5-omni-plus	20.6%	37.5%	12.5%
grok-voice-think	8.8%	34.4%	0.0%
gemini-3.1-flash	2.9%	6.2%	0.0%

F.1.4 Full Failure Attribution

Table 24 reports failure attribution across Reactive, Proactive, and Interruption tasks. Per-tool counts are summed across all failed tasks in each cell (one failed task may contribute multiple failure labels if it has multiple expected tool calls).

F.1.5 Per-Tool Pass Rate

Different tools exhibit significantly different per-call accuracy. Table 25 reports the average per-call success rate across all models for each tool that appears in `expected_tools` (1040-task benchmark).

F.2 ASR Error Catalog

Perception-side (ASR) errors are the largest Reactive failure category for four of the six models (§5.1). This section catalogs the systematic error patterns observed.

Table 24: Per-suite failure attribution. Pass rate is task-level; remaining columns are per-tool-call primary-category shares. ASR = entity + num-ID; Prem. = premature call; Param = parameter reasoning; Missing = missing call; Unexp. = unexpected call.

Model	Suite	Pass	ASR	Prem.	Param	Missing	Unexp.
gpt-realtime-1.5	Reactive	41.9%	55.7%	8.0%	19.5%	16.7%	0.0%
	Proactive	33.9%	23.4%	10.6%	18.1%	46.0%	1.9%
	Interruption	30.8%	43.3%	27.6%	13.4%	13.4%	2.4%
gpt-realtime	Reactive	28.8%	39.0%	25.5%	20.3%	15.2%	0.0%
	Proactive	22.0%	12.8%	21.2%	25.4%	37.9%	2.8%
	Interruption	22.1%	27.3%	42.0%	12.6%	16.1%	2.1%
gpt-realtime-mini	Reactive	27.3%	43.8%	13.4%	22.3%	20.5%	0.0%
	Proactive	16.6%	17.1%	15.6%	20.2%	41.3%	5.8%
	Interruption	19.2%	34.2%	27.7%	15.5%	21.9%	0.6%
qwen3.5-omni-plus	Reactive	13.1%	10.4%	16.2%	30.2%	43.2%	0.0%
	Proactive	23.0%	8.1%	5.1%	23.8%	61.7%	1.3%
	Interruption	15.8%	17.4%	36.5%	17.4%	25.7%	3.0%
grok-voice-think	Reactive	16.7%	22.0%	1.8%	11.1%	65.1%	0.0%
	Proactive	26.0%	28.1%	4.3%	20.5%	44.4%	2.6%
	Interruption	11.7%	23.2%	5.0%	14.4%	54.7%	2.8%
gemini-3.1-flash	Reactive	9.1%	35.6%	2.8%	29.0%	32.6%	0.0%
	Proactive	6.6%	12.0%	9.4%	23.3%	50.2%	5.0%
	Interruption	2.1%	13.3%	16.4%	17.1%	51.0%	2.1%

F.2.1 Error Patterns

The following patterns recur across multiple models, indicating shared error modes among the evaluated real-time voice systems rather than individual model deficiencies.

Root causes. Name space merging occurs because Chinese pinyin names (e.g., "Li Na") are recognized as single English words ("Lina" is a common English name). Digit truncation affects 11-digit phone numbers and 18-digit ID numbers, which exceed the stable recognition length for continuous digit sequences. Homophone confusion arises from similar pinyin initials and finals (zh/z, ou/ao) that are difficult to disambiguate without semantic context.

F.2.2 Error Rate by Parameter Type

The five parameter types with highest ASR error rate (phone numbers, ID numbers, dates, person names, and numeric values) all share the same underlying characteristic: they are high-information short strings (digits or proper nouns) where a single-character error corrupts the entire value. City and place names, by contrast, are high-frequency English words that ASR has strong priors for. This pattern indicates that improving ASR on proper nouns and digit strings, rather than overall word error rate, is an important lever for voice tool-calling accuracy.

F.2.3 Error Rate by Model

F.3 Failure Mode Trace Examples

This section presents one representative trace for each of the six failure attribution categories (§4). Each example includes the user transcript, the expected and actual tool calls, the attribution category, and a differential diagnosis explaining why the error is not assigned to an adjacent category.

F.3.1 Entity Mishearing

Task: The user books a flight and a hotel, providing the passenger and guest name in the final turn. Model: gpt-realtime-1.5.

> **Turn 6 (user):** "Book the flight for Li Na, and put the hotel reservation under Li Na as well."

Table 25: Per-tool average pass rate across all six models, sorted by easiness. Low-support tools are omitted.

Tool	Avg. Pass Rate	Most Common Failure
search_shows	96.7%	missing_call
search_sports_events	95.3%	missing_call
search_movies	94.4%	missing_call
search_restaurants	93.3%	missing_call
search_restaurants_delivery	92.7%	missing_call
search_parking	92.1%	missing_call
search_cars	92.1%	missing_call
search_attractions	91.4%	missing_call
search_courses	91.1%	missing_call
search_doctors	86.7%	missing_call
search_books	85.9%	missing_call
search_hotels	85.7%	missing_call
transfer_money	85.1%	entity_mishearing
check_ride_status	82.2%	entity_mishearing
list_bills	82.1%	missing_call
search_trains	80.6%	missing_call
get_transaction_history	75.9%	numerical_id_error
enroll_course	71.9%	parameter_reasoning
book_movie_ticket	68.1%	missing_call
search_flights	67.4%	missing_call
book_attraction_ticket	66.6%	premature_call
book_appointment	65.6%	missing_call
cancel_ride	63.3%	parameter_reasoning
book_car	62.0%	missing_call
book_hotel	61.9%	parameter_reasoning
place_food_order	61.4%	numerical_id_error
book_sports_ticket	61.1%	numerical_id_error
book_train	59.9%	parameter_reasoning
book_show_ticket	58.5%	missing_call
check_balance	52.2%	numerical_id_error
book_flight	49.5%	parameter_reasoning
book_restaurant	46.9%	missing_call
search_medicine	41.5%	missing_call
pay_bill	37.4%	entity_mishearing
reserve_parking_spot	25.0%	missing_call
reserve_book	23.9%	parameter_reasoning

Table 26: Systematic ASR error patterns. Edit distances are computed after digit normalization for digit-string parameters.

Error Type	Expected	Actual	Edit Dist.	Affected Parameters
Name space merging	Li Na	Lina	1	passenger_name, guest_name,
	Zhang Wei	Zhangwei	1	contact_name, buyer_name,
	Zhou Jun	Zhoujun	1	visitor_name
Name homophone confusion	Zhou Jun	Zhao Jun	1	all name parameters
	Chen Ming	Cheng Ming	1	
Digit truncation	13812345678	1381234567	1	phone, account_number
	110101…1234	110101…123	1	id_number
ID transposition	D321	D231	2	train_no, flight_no
	G1	G01	1	

Table 27: ASR error rate by parameter type.

Parameter Type	Representative Params	ASR Share
Phone numbers	phone, patient_phone, driver_phone	**90.2%**
ID numbers	id_number, account_number	**82.6%**
Dates	date, checkin_date	**81.8%**
Person names	passenger_name, guest_name, buyer_name	**75.5%**
Numeric values	party_size, ticket_count, seat_count	**75.3%**
Other entities	hotel_name, restaurant_name, etc.	20.7%
City / place	city, origin, destination	6.5%

Table 28: ASR error share by model. ASR errors = `entity_mishearing` + `numerical_id_error`. Share = ASR / total per-tool failures.

Model	ASR Share
gpt-realtime-1.5	84.3%
gpt-realtime	63.8%
gpt-realtime-mini	68.1%
qwen3.5-omni-plus	26.7%
grok-voice-think	52.7%
gemini-3.1-flash	84.4%

Expected: `book_flight(..., passenger_name='Li Na')` and `book_hotel(..., guest_name='Li Na')`
Actual: `book_flight(..., passenger_name='Lina')` and `book_hotel(..., guest_name='Lina')`
Attribution: Entity mishearing: the model collapsed the two-token person name "Li Na" into the single-token name "Lina."
Differential: Not *numerical-ID error*, because the corrupted field is a person name rather than a digit string. Not *parameter reasoning*, because the model used the correct tools at the correct time and preserved the intended referent phonetically; the error is consistent with name recognition rather than semantic parameter selection.

F.3.2 NUMERICAL-ID ERROR

Task: The user searches movies and restaurants, then books both, providing a phone number in the final turn. Model: gpt-realtime-1.5.

> **Turn 6 (user):** "Use the contact name Li Na, and the phone number is 13851574217."

Expected: `book_restaurant(..., phone='13851574217')`
Actual: `book_restaurant(..., phone='1385157217')`
Attribution: Numerical-ID error (one digit dropped from an 11-digit phone number).
Differential: Not *entity mishearing*: the corrupted field is a digit string, not a named entity. The truncation pattern (11 $\rightarrow$ 10 digits) matches the systematic digit-string recognition failure described in Table 26.

F.3.3 PREMATURE CALL

Task: The user books a train, a hotel, and attraction tickets. The hotel guest name ("Chen Ming") is not provided until Turn 6. Model: gpt-realtime-1.5.

> **Turn 5 (user):** "Book Hanting Hotel People's Square, checking in on March 22nd, 2026 for 2 nights."
> **Turn 6 (user):** "Put the hotel booking under the name Chen Ming. Also, search attractions in Beijing."

Expected: `book_hotel(..., guest_name='Chen Ming')` called at or after Turn 6.
Actual: `book_hotel(..., guest_name='Zhang Wei')` called at Turn 5.
Attribution: Premature call: the model invoked `book_hotel` before the user provided the guest name, substituting the name from an earlier train booking.
Differential: Not *parameter reasoning*, as the root cause is timing, not reasoning. The model heard the correct name in Turn 6 but had already committed the booking in Turn 5 with a hallucinated guest name carried over from Turn 3.

F.3.4 PARAMETER REASONING

Task: The user books a flight, a restaurant, and cancels a ride, providing the ride ID "RIDE123456" in the final turn. Model: gpt-realtime-1.5.

> **Turn 6 (user):** "Make the reservation under Zhang Wei, phone number 13867894321, and also cancel my ride RIDE123456."

Expected: `cancel_ride(ride_id='RIDE123456')`
Actual: `cancel_ride(ride_id='123456')`
Attribution: Parameter reasoning: the model stripped the "RIDE" prefix, producing an ID that does not exist in the system.
Differential: Not *numerical-ID error*, since the digit portion "123456" is correct. The model heard the full string but made a reasoning decision to drop the prefix, treating it as a label rather than part of the ID value. The error is semantic (wrong parsing), not perceptual (wrong hearing).

F.3.5 MISSING CALL

Task: The user searches movies, books tickets, searches restaurants, and books a reservation. Model: gpt-realtime-1.5.

> **Turn 5 (user):** "Reserve a table at Shanghai Classic on 2026-03-25 at 11:30."
> **Turn 6 (user):** "It's for 6 people, under the name Zhou Jun, and the phone number is 13867894321."

Expected: `search_movies` → `book_movie_ticket` → `search_restaurants` → `book_restaurant`.
Actual: `search_movies` → `book_movie_ticket` → `search_restaurants`.
Attribution: Missing call: `book_restaurant` was never invoked despite all parameters being provided by Turn 6.
Differential: Not *premature call* or *parameter reasoning*, because the model never attempted the call at all. After completing the restaurant search, the model responded conversationally without proceeding to book, indicating a failure to track that a booking action was still pending.

F.3.6 UNEXPECTED CALL

Task: A Proactive Weak task where the user reminisces about past travel without requesting any action. Model: gpt-5.4 (text mode).

> **Turn 4 (user):** "Back in Beijing, I remember we were chatting about what movies were out around then. Felt like everyone had an opinion."
> **Turn 9 (user):** "... and later in Beijing I'd already bought 2 tickets for Article 20 at CGV Xidan for the 15:30 showtime, also under Li Na."

Expected: No tool calls (Weak intent; all entities are past-tense reminiscence).
Actual: `search_movies(city='Beijing')`.
Attribution: Unexpected call: the model initiated a tool call on a task where no action was warranted.
Differential: Not *premature call*, since premature calls involve correct tools at incorrect timing, whereas here the user never requested any action. The model misread retrospective narrative as a current intent, triggering a false-positive tool invocation.

F.4 Per-Tool Failure Heatmap

Table 29 visualizes per-call pass rate for each tool across all six models as a heatmap. Each cell shows the percentage of correctly-issued calls for a (tool, model) pair, revealing which tools are universally hard and which are model-specific bottlenecks. Tools are sorted by average pass rate (easiest at top); tools with fewer than 50 appearances across the benchmark are omitted.

The pattern is consistent: *search* tools (top of table) have substantially higher per-call success than *book* and standalone-action tools (bottom). The hardest tools systematically involve digit-rich parameters (`book_train`, `book_flight` have low pass rates partly because they require correctly-recognized train/flight numbers and seat selection details) or ambiguous identifiers. Models cluster together on tool difficulty rankings, suggesting tool-level difficulty is largely shared across providers.

F.5 Parameter Type Failure Analysis

Figure 7 visualizes failure modes by parameter slot type, aggregated across all six models on the 1040-task benchmark. Table 30 reports the exact values.

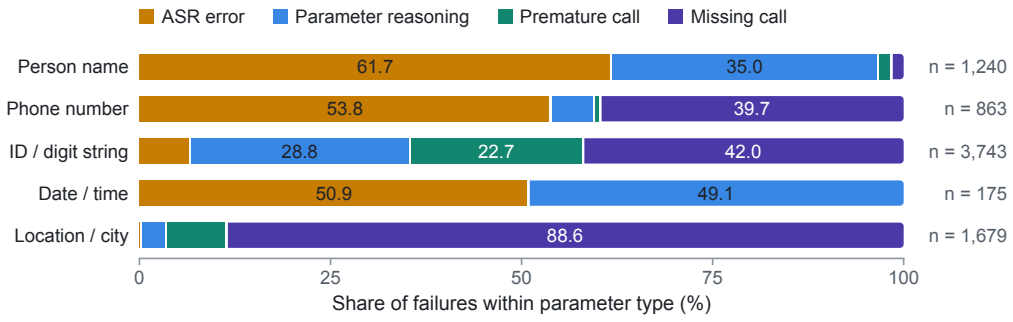


Figure 7: Failure-mode distribution by parameter semantic type. Bars are normalized within each parameter type; labels show high-share segments, and right-side annotations report the total number of failures for that type.

Findings. Person names and phone numbers exhibit the highest ASR error rates (61.7% and 53.8% of all failures respectively), suggesting that proper-noun and digit-string recognition are major perception bottlenecks in this setting. Date / time errors split nearly evenly between ASR (50.9%) and parameter reasoning (49.1%), reflecting both transcription challenges (numeric date formats) and ambiguity in date interpretation. Location / city parameters show very low ASR error (0.2%) but very high missing-call rate (88.6%): cities and place names rarely show ASR errors, but the surrounding tool calls often go unrequested entirely. ID / digit string parameters are dominated by missing calls (42.0%) and parameter-reasoning errors (28.8%) rather than pure transcription, indicating that the difficulty often lies in the model's downstream handling rather than perception alone.

F.6 Cross-Model Failure Consistency

Figure 8 visualizes pairwise Jaccard similarity $J = |A \cap B|/|A \cup B|$ over models' failed Reactive task sets (200 tasks per model). Higher values indicate models fail on the same tasks. Table 31 reports the exact values.

Table 29: Heatmap of per-tool pass rate across all six models. Darker cells indicate higher per-call success. Tools are sorted by average pass rate (easiest at top); low-support tools are omitted. **Bold** = best per row.

Tool	gpt-1.5	gpt	mini	qwen	grok	gemini	Avg
search_shows	99%	99%	**100%**	92%	97%	93%	97%
search_sports_events	96%	95%	96%	94%	**99%**	91%	95%
search_movies	**99%**	**99%**	99%	96%	95%	78%	94%
search_restaurants	**99%**	97%	98%	94%	96%	76%	93%
search_restaurants_delivery	**96%**	95%	95%	88%	89%	93%	93%
search_parking	97%	83%	**99%**	85%	97%	91%	92%
search_cars	**98%**	92%	94%	88%	97%	84%	92%
search_attractions	96%	**97%**	97%	88%	94%	77%	91%
search_courses	**100%**	**100%**	**100%**	97%	94%	56%	91%
search_doctors	83%	**97%**	**97%**	82%	85%	76%	87%
search_books	**97%**	91%	91%	91%	81%	66%	86%
search_hotels	97%	90%	**97%**	75%	92%	63%	86%
transfer_money	**94%**	88%	82%	**94%**	75%	76%	85%
check_ride_status	**90%**	83%	83%	83%	72%	83%	82%
list_bills	**92%**	**92%**	**92%**	69%	69%	77%	82%
search_trains	**96%**	74%	87%	71%	87%	68%	81%
get_transaction_history	72%	**83%**	79%	76%	72%	72%	76%
enroll_course	81%	78%	**84%**	78%	72%	38%	72%
book_movie_ticket	**81%**	78%	69%	61%	71%	48%	68%
search_flights	**85%**	59%	69%	57%	71%	64%	67%
book_attraction_ticket	**78%**	72%	69%	61%	70%	50%	67%
book_appointment	73%	**81%**	68%	56%	60%	55%	66%
cancel_ride	**72%**	64%	60%	64%	64%	56%	63%
book_car	73%	64%	57%	47%	**76%**	55%	62%
book_hotel	**80%**	69%	72%	49%	65%	36%	62%
place_food_order	60%	63%	60%	**65%**	63%	58%	61%
book_sports_ticket	**64%**	63%	60%	62%	59%	59%	61%
book_train	**80%**	67%	65%	46%	61%	41%	60%
book_show_ticket	**68%**	59%	65%	55%	57%	46%	58%
check_balance	**57%**	**57%**	48%	52%	48%	52%	52%
book_flight	**70%**	47%	53%	38%	49%	40%	50%
book_restaurant	**57%**	48%	47%	39%	55%	35%	47%
search_medicine	40%	37%	30%	47%	**53%**	42%	41%
pay_bill	41%	**45%**	34%	34%	34%	34%	37%
reserve_parking_spot	**25%**	**25%**	**25%**	**25%**	**25%**	**25%**	25%
reserve_book	**30%**	23%	27%	23%	23%	17%	24%

Table 30: Failure mode distribution by parameter semantic type, aggregated across all six models. Percentages are per-row (sum to 100% within each parameter type, with rounding).

Param Type	ASR	Param	Premature	Missing
Person name	61.7%	35.0%	1.7%	1.6%
Phone number	53.8%	5.7%	0.8%	39.7%
ID / digit string	6.6%	28.8%	22.7%	42.0%
Date / time	50.9%	49.1%	0.0%	0.0%
Location / city	0.2%	3.3%	7.9%	88.6%

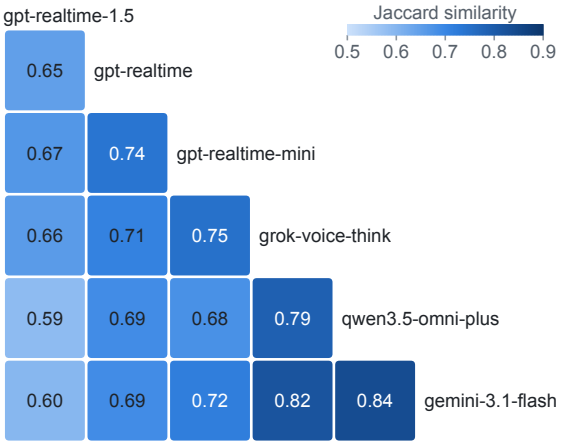


Figure 8: Lower-triangular heatmap of pairwise overlap among models' failed Reactive task sets; each model name labels its row (cells to its left) and its column (cells below it). Models are ordered to group the OpenAI variants and the high-overlap weaker-model cluster; darker cells indicate higher overlap.

Table 31: Pairwise Jaccard similarity $J = |A \cap B|/|A \cup B|$ over models' failed Reactive task sets, reported as percentages. Higher values indicate models fail on the same tasks. Lower-triangular shown.

	gpt-1.5	gpt	mini	qwen	grok
gpt-realtime	65%	–	–	–	–
gpt-realtime-mini	67%	74%	–	–	–
qwen3.5-omni-plus	59%	69%	68%	–	–
grok-voice-think	66%	71%	75%	79%	–
gemini-3.1-flash	60%	69%	72%	84%	82%

Findings. Pairwise Jaccard ranges from 59% to 84%, reflecting substantial overlap in which Reactive tasks each model fails. The highest overlap is between qwen and gemini (84%) and between qwen and grok (79%), indicating that the weaker models share a large common pool of failures. The OpenAI variants (gpt, gpt-mini, gpt-1.5) have moderate pairwise overlap (65–74%) reflecting both shared core difficulties (e.g., proper-noun ASR errors) and differentiation in higher-difficulty reasoning bands. The high overall Jaccard floor (no pair below 59%) is consistent with shared perception-bound hard cases driven by the systematic ASR error patterns documented in §F.2.

F.7 Latency Statistics

Table 32 reports the numerical latency statistics behind Figure 2b. Metrics are timed from user-audio commit and include server-side VAD overhead.

Table 32: Latency statistics (ms). Best per column in **bold**.

	TTFT		TTFC	
Model	**Mean**	**P50**	**Mean**	**P50**
gpt-realtime-1.5	2242	1865	**1203**	**907**
gpt-realtime	**2085**	1702	1258	964
gpt-realtime-mini	2555	1857	1625	975
qwen3.5-omni-plus	2146	1701	1576	1199
grok-voice-think	2151	**1588**	3461	2770
gemini-3.1-flash	4021	3133	3098	2306

F.8 Additional Ablations

F.8.1 Tool Description Richness

We prepare two tool-description variants for all 200 Reactive tasks: *minimal* (function signature only) and *verbose* (detailed description with parameter semantics and examples). We evaluate 3 models under both conditions on the same aligned task set.

Table 33: Reactive pass rate under minimal vs. verbose tool descriptions.

Model	**Minimal**	**Verbose**	Δ
gpt-realtime-1.5	32.5%	37.0%	+4.5 pp
qwen3.5-omni-plus	20.5%	26.0%	+5.5 pp
grok-voice-think	14.0%	18.0%	+4.0 pp

All three models benefit from verbose tool descriptions, with consistent gains of +4–5.5 pp. qwen3.5-omni-plus shows the largest improvement (+5.5 pp), followed by gpt-realtime-1.5 (+4.5 pp) and grok-voice-think (+4.0 pp). The uniformly positive but modest effect suggests that richer documentation helps models select correct tools and parameters, but *tool discoverability does not appear to be the main bottleneck in this ablation*—even with verbose descriptions, pass rates remain below 37%.

F.8.2 Behavior-vs-Task Gap

Table 34 reports the gap between tool-level accuracy (*behavior score*) and task-level pass rate on the combined 200-task Reactive benchmark. Per-tool behavior scores reach 61–85%, but task-level pass rates collapse to 9–42%. The $\sim$43–54 pp gap reflects how a single ASR-corrupted call in a multi-tool chain flips the entire task to fail.

Table 34: Self-correction gap: task-level pass rate vs. average per-tool behavior score. The gap reflects how often a single bad call flips an otherwise good task to fail.

Model	**Pass Rate**	**Behavior Score**	**Gap**
gpt-realtime-1.5	41.9%	84.9%	43.0 pp
gpt-realtime	28.8%	83.1%	54.3 pp
gpt-realtime-mini	27.3%	81.0%	53.7 pp
qwen3.5-omni-plus	13.1%	63.5%	50.4 pp
grok-voice-think	16.7%	65.9%	49.2 pp
gemini-3.1-flash	9.1%	61.1%	52.0 pp

G Harness Implementation Details

This section gives the mechanics of the five harnesses behind Section 5.5: the three protocol-layer harnesses in Table 4 (transcript rewrite, audio replay, interruption lock) and two further intent-layer harnesses.

G.1 Design Principle

Each harness corrects only signals the base model cannot directly observe: ASR errors that the model receives as already-mangled text, and session-level state changes that the Realtime protocol does not

surface. Anything that requires inferring user intent from context is left to the base model. A tool call is considered *commit-class* when its name starts with one of `book_`, `cancel_`, `order_`, `place_`, `pay_`, or `confirm_`; the three protocol-layer harnesses only gate calls in this set.

G.2 Harness Inputs

The harness is implemented as a subclass of the Realtime WebSocket client, so every event between the runner and the API passes through it. This gives the harness two private channels of input beyond what the base model sees.

The first is a running text buffer that concatenates three event streams: finalized user transcripts (`conversation.item.input_audio_transcription.completed`, emitted after the API transcribes the user audio), assistant transcripts (`response.audio_transcript.delta` and `.done`), and tool outputs (truncated to 1000 characters per call). This buffer is the input to the transcript rewrite and to the keyword-based gates. It contains the API's own streaming ASR output, which often preserves identifiers and names that the model truncates in its tool arguments. The buffer typically holds `RIDE123456` while the model emits `ride_id="123456"`, which is exactly the signal the rewrite rule keys on.

The second comes from overriding the audio-send path: every PCM chunk the runner forwards to the API is first copied into a per-turn buffer, snapshotted at `input_audio_buffer.committed`, and kept as the most recent user turn's raw audio. This audio is the input to the audio replay; the streaming ASR inside the Realtime API is tuned for latency over fidelity and tends to drop or mis-segment high-entropy tokens like IDs and phone numbers, so the replay re-runs the same audio through a batch transcriber (`gpt-4o-transcribe`) with a prompt focused on those token types.

G.3 Transcript Rewrite

The transcript rewrite runs immediately before every commit-class tool call once the running transcript exceeds 60 characters. For every string argument whose name contains an identifier token (`_id`, `id_`, `ride`, `booking`, `ticket`, `tracking`, or `account`), the harness scans the transcript for substrings matching `[A-Z]{2,6}\d{4,}`, the prefixed-ID pattern used throughout our task suite. If a candidate from the transcript strictly contains the model's argument value, the candidate replaces it. For arguments whose name contains `phone`, `mobile`, or ends in `_no`, the harness scans for an 11-digit Chinese mobile pattern and replaces when the candidate is longer than the model's digits and includes a four-or-more-digit run from the model's value. The rule only lengthens an argument toward a fully-prefixed reference.

As an example, the model emits `cancel_ride(ride_id="123456")`. The transcript contains "the driver gave me a ride ID, R-I-D-E one-two-three-four-five-six." The harness finds `RIDE123456`, sees that `"123456"` is a strict substring, and overwrites the argument before the call executes.

G.4 Audio Replay

The audio replay applies the rewrite comparison rule to a higher-fidelity transcript. It triggers when the model issues a commit-class tool call, the most recent user turn has buffered audio, and at least one string argument matches the high-risk set (`_id`, `phone`, `name`, `code`, `number`). The harness sends the buffered user audio to `gpt-4o-transcribe` with a focused prompt asking the transcriber to surface IDs, phone numbers, names, and codes verbatim. The returned text becomes a second transcript and the rewrite rule is applied to it: any argument whose model-emitted value is a strict substring of a candidate in the replay text is overwritten by that candidate.

If the replay contains a more complete version of an argument, the rewrite fires; otherwise the argument is left untouched, which avoids false positives on values the model already has right.

G.5 Interruption Lock

The interruption lock watches each finalized user transcript as it arrives, lowercases it, and tests it against a fixed list of about two dozen bilingual interruption keywords: "scratch that," "cancel that," "never mind," "instead of," "hold on," "wait," "actually," "i meant," and a few minor variants. On

the first match in a session, the harness sends a single `session.update` event to the Realtime API and overwrites the session-level instructions with a more cautious prompt: treat the most recent user statement as authoritative, drop assumptions from before the correction, and refuse to commit a booking, cancellation, or payment until every required argument has been explicitly re-stated. The lock is one-shot per session.

The lock acts at the policy layer: the `session.update` swaps the system prompt for a more conservative one and tool execution remains with the model under that prompt.

G.6 Intent-Layer Harnesses

Two further harnesses act on user-intent signals carried by transcripts. An action-bias gate watches user transcripts for committed-action phrases (e.g. "go ahead and," "lock that in," "make the reservation") and, on the first match, sends a `session.update` that biases the model toward acting immediately when an intent and the required parameters have both been stated. A response-delay gate complements it: once action bias is armed, whenever the assistant emits a transcript that is at least 120 characters long and ends in a question mark, the harness buffers the next 2.5 seconds of incoming user audio locally before forwarding it to the API. The goal is to let the user finish answering a long clarifying question before the model sees a fragmentary turn. Both gates show the same per-suite trade-off as the three protocol-layer harnesses, so we omit a separate row for them.