Towards Reliable Instruction-Following in LLM Multi-Turn Workflows: An Empirical Comparison of Prompting and Control Architectures

AAI Labs - Whitepaper

James Wanjiku · Andrii Zhurba · Yeabtsega Yifat

March 2026

Abstract

Large Language Models exhibit systematic instruction-following degradation in multi-turn conversational workflows (skipping prescribed stages, failing objection-handling protocols, and re-asking previously answered questions as context depth increases). This paper empirically evaluates four LLM architectures across three models, comprising 216 live multi-turn test calls (using a structured sales call as an evaluation proxy): a static prompting Baseline, a Reactive Validator (RV), a Continuous Prompt Updates (CPU) parser, and an External State Machine (ESM). We find that dynamic control architectures achieve superior Workflow Adherence Rates (WAR up to 0.88) at smaller prompt sizes (5K characters), while the static Baseline requires 20K characters to reach comparable performance on cooperative call paths. The CPU architecture delivers the best adherence-to-latency trade-off (2.9 s mean turn latency at 5K). However, all advanced architectures introduce qualitatively different failure modes (observer timing lag, probabilistic extraction failures, and parser misclassification) that produce robotic conversational breakdowns absent from the Baseline. We conclude with architecture selection guidance and a roadmap for future work in fine-tuning and multi-agent control.

---

1 Summary

Large Language Models (LLMs) are increasingly used to power conversational agents. However, when these agents are required to follow strict multi-turn workflows (like a sales call or medical intake), they struggle to reliably adhere to their instructions. This instruction-following degradation is not solely a function of conversation length; models can fail at any point, whether forgetting early instructions, skipping mandatory steps right from the introduction, or getting stuck in repetitive loops as the dialogue progresses.

This paper investigates whether we can mitigate this "instruction-following degradation" by utilizing external architectural controls rather than relying solely on prompting. We compare a standard "all-in-one-prompt" approach against three alternative architectures that use secondary

LLMs to control the conversation flow. Our primary evaluation uses GPT-5.1, Llama-3.1-70B and Qwen-3.5-27B models, yielding 216 total live calls under identical scenarios and prompt sizes.

Our main takeaways:

- **Additional architectures improve tracking but are slower.** Approaches that dynamically update the prompt or use a state machine can drastically reduce errors, but they introduce delays that might be unacceptable for real-time applications such as live voice calls.
- **Smaller prompts are often better.** When using multi-agent architectures, smaller, focused instructions (5K characters) outperformed massive, detailed rulebooks (20K characters) across GPT-5.1, Llama-3.1-70B, and Qwen-3.5-27B.
- **Architecture matters more than model choice.** Once architectures are held constant, open-weight models (Llama-3.1-70B, Qwen-3.5-27B) achieve instruction-following performance comparable to GPT-5.1; the largest gains come from moving from Baseline → CPU/ESM rather than from swapping models.
- **The right choice depends on your needs.** Baseline/RV architectures suit real-time voice systems when using direct provider APIs; RV performance degrades significantly on slower routing layers (e.g., OpenRouter) due to its dependency on near-real-time observer synchronization. For text-based chat where slight delays are acceptable, CPU offers the highest accuracy and generalizes robustly across all three model families.

2 Introduction

Large Language Models (LLMs) have emerged as a compelling foundation for building conversational agents capable of executing structured, multi-turn workflows. Unlike document retrieval or question-answering tasks, multi-turn automation imposes a strict sequential dependency between conversation stages (e.g., introduction, gatekeeping, discovery, pitch, and close) while simultaneously requiring the agent to adapt to unpredictable prospect behaviour, handle objections gracefully, and maintain a consistent persona across an entire call.

The prevailing deployment strategy for such agents is *pure system prompting*: encode all procedural rules, persona constraints, objection-handling protocols, and company knowledge into a single static system prompt. While operationally simple, this approach conflates memory, role, and workflow instruction into one monolithic input, creating conditions that are known to produce instruction-following degradation as early as the first conversational turn and compounding as prompt complexity grows [1], [2], [3].

This paper investigates whether architectural interventions (introducing secondary control agents to manage state, route stage transitions, or inject corrections mid-call) can meaningfully reduce this degradation in a real-world multi-turn conversational setting. We evaluate four architectures spanning a spectrum from static prompting to deterministic state machines, conducting 216 live conversational test calls with GPT-5.1, Llama-3.1-70B and Qwen-3.5-27B, across three system prompt sizes to produce a directly comparable cross-model dataset. This cross-model replication allows us to separate architectural effects from provider-specific or training-specific idiosyncrasies, and to quantify how reliably architecture-level interventions transfer across model families.

Our findings reveal that while advanced architectures demonstrably improve structural adherence under cooperative conditions, they introduce latency penalties, observer reliability constraints, and qualitatively different failure modes that must be weighed against raw performance gains.

We characterise these trade-offs and show that the relative architectural ordering (CPU > ESM > RV > Baseline) remains stable across all three model families, offering concrete guidance on architecture selection for production multi-turn agent deployments.

3 Related Work

The phenomenon of instruction-following degradation in LLMs is well-documented. As established by [2], LLMs exhibit a "U-shaped" performance curve in long context windows, where they "struggle to extract relevant information if it resides in the middle of their input context." This is further complicated by premise positioning, with [3] demonstrating that "Large Language Models are sensitive to the order of premises."

This degradation becomes critically pronounced in extended, multi-turn dialogues. Recent benchmarks like MMMT-IF [1] demonstrate a severe drop in the Programmatic Instruction Following (PIF) metric, observing that adherence falls rapidly from an average of 0.81 at turn one down to 0.64 by turn twenty. As [4] highlight in their work on the Parrot framework, enhancing multi-turn instruction following remains a primary challenge as models frequently struggle to maintain coherence when instructions are introduced incrementally.

Furthermore, this failure to adhere to multi-turn workflows is increasingly recognized as a systemic issue. A 2024 study on "Instruction Position Matters in Sequence Generation with Large Language Models" [5] categorizes this as "guidance drift," where sequential interactions lead to a decoupling of instruction adherence over time, despite the retention of underlying knowledge. Recent studies have attempted to mitigate this through prompt repetition, with [6] concluding that "Prompt Repetition Improves Non-Reasoning LLMs" by using repetition to anchor the model's attention. While repetition can be effective, we argue that relying on external architectural controls (like state machines or dynamic prompt injection) provides a more deterministic and robust solution for complex multi-turn workflows. The ongoing challenge of multi-turn instruction following is also reflected in emerging benchmarks like EvolIF [7], which highlight the vast gap between single-turn capabilities and sustained conversational adherence. Whereas most existing studies evaluate degradation within a single-model or single-provider setting, our cross-model results reinforce this view: instruction-following degradation appears as a model-agnostic property of current LLMs, manifesting similarly in GPT-5.1, Llama-3.1-70B, and Qwen-3.5-27B despite substantial differences in training regimes and deployment infrastructure.

On the architectural side, prior work has explored multi-agent and state-driven control as a means of improving LLM task reliability. AutoGen [8] introduces a framework of conversable agents that coordinate via structured message passing to execute complex workflows, demonstrating that decomposing tasks across cooperating agents improves robustness over single-agent prompting. StateFlow [9] formalizes this further by modeling LLM task execution as finite state machines, showing that explicit state transitions with deterministic routing significantly improve success rates on multi-step tasks. Our work builds on this direction but shifts the focus from task completion benchmarks to live conversational instruction-following across multiple model families, and documents the failure modes that emerge specifically when these architectures are applied under real-time latency constraints.

4 Problem Definition

4.1 The Instruction-Following Degradation Hypothesis

An LLM agent in a multi-turn workflow is most accurately modelled as attempting to maintain two simultaneous objectives: (1) conversational coherence (responding contextually and naturally to the immediate preceding utterance) and (2) procedural adherence (executing a predefined stage sequence regardless of conversational drift. These objectives are in tension.

As conversation depth increases, the proportion of the active context window occupied by the conversation history grows relative to the system prompt. Prior work demonstrates that LLMs exhibit recency bias, over-weighting recent tokens when the context window is saturated, which degrades compliance with instructions anchored earlier in the prompt [2]. In a multi-stage workflow, this manifests as:

- **Skipped stages:** The agent bypasses mandatory steps (e.g., in our specific evaluation proxy, skipping verifying the Decision Maker) to pursue conversational goals that are implicitly rewarded by RLHF training (rapport, fluency, goal completion).
- **Premature transitions:** The agent moves to a concluding stage (e.g., Pitch or Close) before completing prerequisite analysis (e.g., Discovery), optimising for perceived conversational progress.
- **Incomplete objection handling:** Under resistance, the agent fails to execute prescribed protocols (e.g., the `acknowledge → clarify → qualifying question` protocol), instead pivoting directly to re-attempting a value proposition.
- **Redundant questioning:** The agent re-asks questions already answered earlier in the call, indicating failure to track conversational state across the full context.

4.2 Research Questions

This study addresses the following empirical questions:

1. **RQ1: Prompt Size Effect:** Does increasing system prompt verbosity (5K, 10K, 20K characters) improve instruction-following reliability in a static prompting baseline?
2. **RQ2: Architecture Effect:** Do secondary control mechanisms (reactive validation, continuous prompt updates, deterministic state machines) produce measurable improvement over the baseline ceiling?
3. **RQ3: Production Trade-offs:** What are the latency, reliability, and implementation complexity costs of adopting advanced architectures, and do the gains justify them?
4. **RQ4: Cross-Model Generalization:** Do the relative benefits and failure modes of each architecture remain stable across different model families (GPT-5.1, Llama-3.1-70B, Qwen-3.5-27B)?

4.3 Study Context and Agent Configuration

All experiments were conducted using a structured B2B sales scenario where an agent attempts to book a meeting on behalf of **Aegis Marine Systems (AMS)**. This scenario serves as a proxy for multi-turn workflow adherence. The task was selected to provide a controlled, stage-based interaction, but the findings generalize to other domains such as healthcare intake or customer support triage.

To formalize the evaluation environment for the LLM architectures, we defined a strict 7-stage schema (`SALES_STAGE_DESCRIPTIONS`):

1. **Introduction:** Identify agent and company; confirm prospect identity; state call reason.

2. **Gatekeeping:** If non-decision maker, route and request transfer/callback.
3. **Discovery:** Confirm relevance (priorities/timing) via 1-2 focused questions.
4. **Pitch:** Deliver a concise value statement with a credibility hook.
5. **Objection Handling:** Acknowledge constraint, clarify, and ask one qualifying question.
6. **Close:** Ask for meeting, propose times, and secure calendar slot.
7. **Incompatible:** Acknowledge poor domain fit and end call politely.

The prescribed chronological sequence evaluated is:

$$\text{Introduction} \rightarrow \text{Gatekeeping} \rightarrow \text{Discovery} \rightarrow \text{Pitch} \rightarrow \text{Close}$$

with an optional `Objection Handling` loop and an `Incompatible Domain` exit path.

Evaluation was conducted under two modalities: Baseline and Reactive Validator (RV) were tested via live voice interactions (Deepgram Voice Agent API), while Continuous Prompt Updates (CPU) and External State Machine (ESM) architectures (as well as the Pub/Sub architecture spike) were evaluated in a CLI text environment.

All interactions were live, human-to-agent conversations (no synthetic or LLM-generated simulations). Researchers followed predefined scripts to control for variability while preserving realistic conversational timing.

5 Experimental Design

5.1 Architectures Under Evaluation

This section describes the architectures, models, and evaluation protocol used in this study. We investigated five different architectures during the initial technical scope, though only four were advanced to the formal evaluation phase:

- **Baseline (Pure Prompting):** The current industry standard: a single system prompt encodes persona, workflow, and behavioral rules. The model processes full conversation history without external state tracking or intervention.

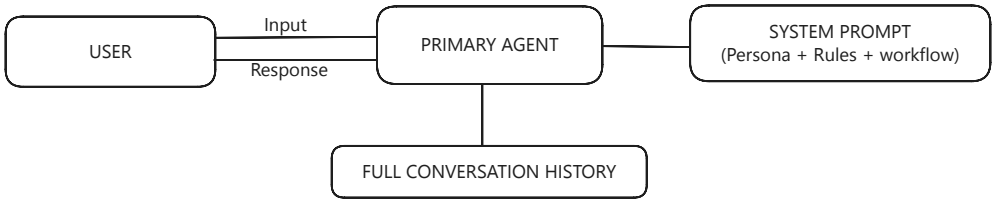

Figure 1: Baseline Architecture overview

- **Continuous Prompt Updates (CPU):** Utilizes a secondary *Parser LLM* to manage stage transitions. The workflow is discretized into stages (*Introduction*, *Gatekeeping*, *Discovery*, *Pitch*, *Close*, *Objection Handling*, *Incompatible Domain*). After each agent-user turn, the Parser analyzes the exchange and outputs a structured JSON object containing the `next_stage` and an `explanation`. Transition-specific instructions are then injected into the primary agent's context.

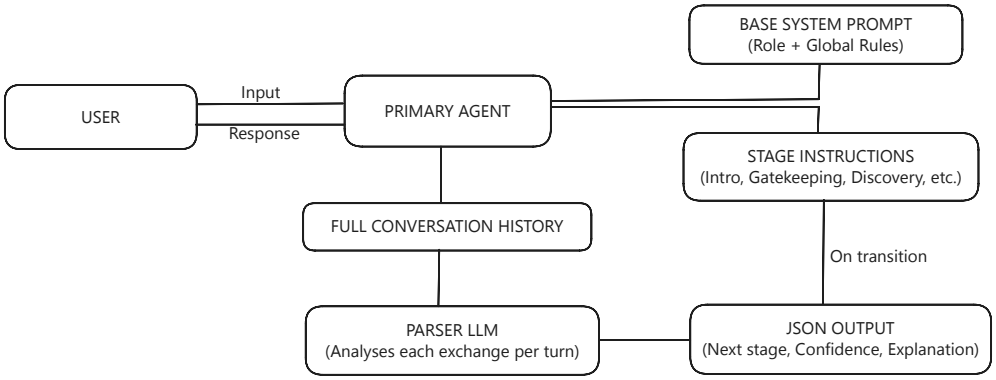

Figure 2: CPU Architecture overview

- **External State Machine (ESM):** While similar to the Continuous Prompt Updates architecture, this architecture decouples state transition logic from the LLM. The Parser LLM serves exclusively as an information extractor to update a structured memory schema. A deterministic rules engine (e.g., LangGraph [10]) then evaluates this schema to trigger stage transitions. Additionally, this architecture employs full system prompt replacement rather than incremental injection to minimize context-window noise.

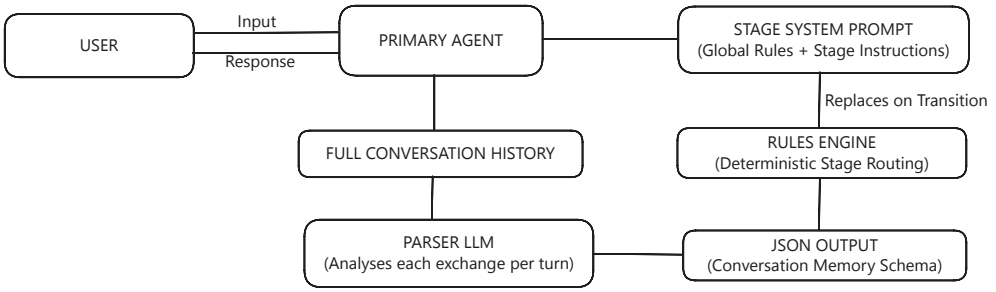

Figure 3: ESM Architecture overview

- **Reactive Validator (RV):** Features a secondary observer agent running in parallel with the primary actor. This agent monitors the conversation for immediate deviations and intervenes via `[SYSTEM MESSAGE]` injections or dynamic prompt messages to resolve inconsistencies in real-time.

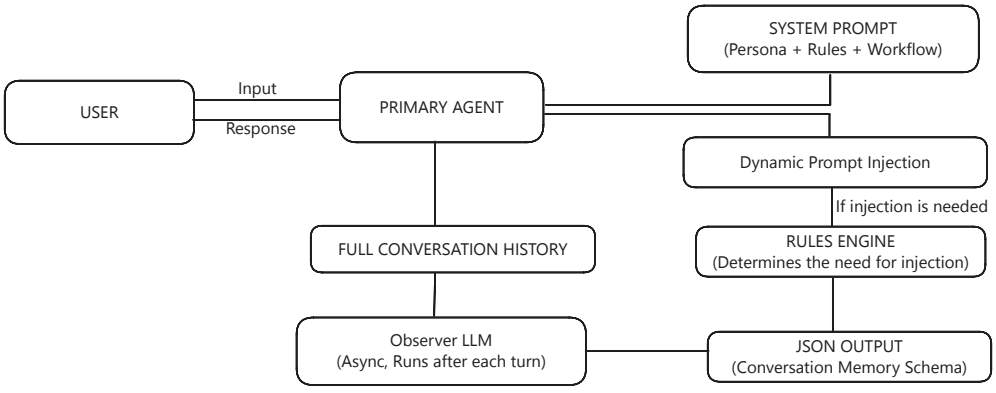

Figure 4: RV Architecture overview

- **Publisher-Subscriber (Pub-Sub):** Modeled using a Pipecat parallel pipeline, this architecture utilizes a true parallel validator that evaluates the agent's response and can inject a correction frame (`LLMMessagesAppendFrame`) to force output regeneration. **This architecture was intentionally excluded from the formal 216-call evaluation.** While technically powerful for background JSON processing or API tasks, applying regeneration retries directly to streaming paths produced severe conversational stuttering and robotic repetitions that fundamentally broke real-time voice and unscripted chat requirements.

5.2 Model Configuration

The study is structured in two phases. The primary evaluation phase used **GPT-5.1** for all conversational agents across 72 live multi-turn calls. This established the architectural baseline against which dynamic and static control strategies were compared.

In GPT-5.1 experiments, auxiliary control functions (parsing, extraction, and validation) in the Reactive Validator (RV), Continuous Prompt Updates (CPU), and External State Machine (ESM) architectures were performed using GPT-4o-mini. This separation reflects a production-oriented design in which a smaller, lower-latency model handles structured control tasks, while the primary conversational agent (GPT-5.1) focuses on dialogue generation.

A subsequent cross-model extension replicated the same 72-call protocol using two open-weight models accessed via OpenRouter: **Llama-3.1-70B** and **Qwen-3.5-27B**. Both models were tested under identical scenario conditions and prompt sizes (5K, 10K, 20K characters), yielding 216 total evaluation calls. Architectures employing auxiliary control mechanisms used the same open-weight model for both primary agent and Parser/Observer roles.

All three models operated at default temperature with tool/function calling disabled. Evaluation and metric annotation were conducted using **GPT-5.2** as the LLM-as-a-Judge across all model families.

5.3 Test Suite and Evaluation Protocol

The protocol evaluates architectures across varying prompt complexities and conversational paths. We employ a controlled human-in-the-loop simulation to generate the evaluation dataset.

5.3.1 Input Variables

To test the degradation as a function of context density, each architecture is tested against three system prompt sizes: **5K, 10K, and 20K characters**. While the core instructions remain constant, the verbosity and granularity of the constraints are scaled to simulate high-complexity production environments.

5.3.2 Conversation Scenarios

For each configuration, we conduct six standardized conversational trials ($N = 6$):

- **Happy Path (n=3):** The agent interacts with a cooperative prospect without deliberate objections. The call progresses smoothly to a booked meeting.
- **Objection Path (n=3):** The prospect raises one or more resistance signals (timing, disinterest, channel preference, "not interested").

Conversations are driven by researchers following predefined scripts to ensure consistency across different model iterations.

5.4 Evaluation Metrics

We utilize an LLM-as-a-Judge framework to provide quantitative labels for qualitative conversational failures. The judge model analyzes each conversation transcript to calculate the following metrics:

5.4.1 Workflow Adherence Rate (WAR)

This measures the agent's success in navigating the linear sequence (Introduction $\rightarrow$ Decision Maker Verification $\rightarrow$ Discovery $\rightarrow$ Pitch $\rightarrow$ Close) without unauthorized skipping or reordering. The judge assigns a score $s \in [0, 1]$ per conversation.

$$\text{WAR} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

5.4.2 Step Violation Frequency (SVF)

This metric is used to quantify the severity of structural breakdowns. We measure the average number of skipped or out-of-order steps per interaction:

$$\text{SVF} = \frac{\sum_{i=1}^{N} \text{Number of Skipped Steps}}{N}$$

5.4.3 Identity Drift Rate (IDR)

This measures the frequency with which the agent deviates from its assigned persona or system-defined constraints:

$$\text{IDR} = \frac{\sum_{i=1}^{N} \text{Identity Drift Instances}}{N}$$

5.4.4 Redundant Question Rate (RQR)

This quantifies the failure to utilize conversational memory by counting how often an agent re-asks questions previously answered by the user:

$$\text{RQR} = \frac{\sum_{i=1}^{N} \text{Redundant Questions}}{N}$$

5.4.5 Global State Precision (GSP)

This evaluates the agent's accuracy in identifying and responding to non-linear triggers. The judge calculates a score $s \in [0, 1]$ per conversation:

$$\text{GSP} = \frac{1}{N} \sum_{i=1}^{N} s_i$$

6 Results

This section outlines the quantitative findings from 216 live multi-turn call tests across four architectural paradigms (Baseline, Reactive Validator (RV), Continuous Prompt Updates (CPU), External State Machine (ESM)) and three model families (GPT-5.1, Llama-3.1-70B, Qwen-3.5-27B). The primary GPT-5.1 results (72 calls, 18 per architecture) are presented first to establish architectural baselines. A cross-model extension presents the replicated Llama-3.1-70B and Qwen-3.5-27B results (72 calls each) to evaluate cost, latency, and adherence trade-offs across providers.

6.1 Prompt Size vs. Adherence

Our testing evaluated three system prompt sizes (5K, 10K, and 20K characters) to determine if prompt verbosity correlates with improved Workflow Adherence Rate (WAR). The data reveals a clear pattern of diminishing returns.

While the Baseline architecture saw a mechanical improvement in WAR as prompt size increased (scaling from 0.71 at 5K to 0.85 at 20K), the advanced architectures (RV, CPU, ESM) plateaued or even degraded with larger prompts.

Prompt Size	Baseline	RV	CPU	ESM
5K	0.71	0.79	0.84	0.77
10K	0.81	0.76	0.87	0.86
20K	0.85	0.76	0.88	0.78

Table 1: Comparison of Workflow Adherence Rates (WAR) across paradigms (GPT-5.1).

Key Finding: CPU shows marginal WAR gains from 5K to 20K (0.84→0.88), but the difference is small relative to the latency cost. 5K is the safest default for dynamic architectures; larger prompts are only warranted for static Baseline deployments.

6.2 Reliability Under Conversational Load

To isolate how multi-turn workflows impact system reliability, we tracked four key metrics across all architectures and prompt sizes: Identity Drift Rate (IDR), Redundant Question Rate (RQR), Step Violation Frequency (SVF), and Global State Precision (GSP). Each trial consisted of an average of 8 conversational turns, with complex paths reaching up to 13 turns. The data illustrates that while persona maintenance (IDR) is effectively solved in modern models, structural degradation (SVF, RQR and GSP) remains highly sensitive to both the prompt size and the underlying control architecture.

6.2.1 1. Identity Drift Rate (IDR)

Identity drift is functionally non-existent in GPT-5.1 (one incident across 72 calls), but surfaced as a significant failure mode in open-weight models. Qwen and Llama exhibited drift in up to 33% of trials, particularly in ESM/RV configurations where state extraction errors forced unnatural context regressions.

6.2.2 2. Redundant Question Rate (RQR)

RQR measures conversational loops (asking already-answered questions). Baseline and Reactive Validator Architectures showed an inverse relationship with prompt size: larger prompts helped the agent track previously provided information more reliably. However, dynamic routing architectures (CPU, ESM) drastically reduced redundancy at the 5K level, though they showed some regression at higher prompt sizes due to Parser LLM extraction failures.

Prompt Size	Baseline	RV	CPU	ESM
5K	1.17	2.17	0.00	0.00
10K	1.00	0.83	0.00	0.83
20K	0.83	0.33	0.50	0.50

Table 2: Mean Redundant Questions per Session (GPT-5.1).

6.2.3 3. Average Step Violation Frequency (SVF)

SVF isolates the number of skipped or out-of-order steps, and premature transitions. While larger prompts slightly reduced violations in the static Baseline and Reactive Validator, the Continuous Prompt Updates (CPU) and External State Machine (ESM) architectures proved vastly superior across all sizes, stabilizing at roughly 1 violation per call (mostly driven by evaluator artifacts rather than actual behavioral failures).

Prompt Size	Baseline	RV	CPU	ESM
5K	3.17	3.50	1.00	1.00
10K	2.17	3.50	1.00	1.17
20K	2.33	2.83	1.17	1.50

Table 3: Average Number of Workflow violations (GPT-5.1).

6.2.4 4. Global State Precision (GSP)

GSP measures how accurately the agent tracks non-linear conversational states, particularly during adversarial objection handling. The **Baseline architecture** showed strong performance at the 20K prompt size (0.94), suggesting that given enough context and examples, the model can navigate complex states. However, the **Continuous Prompt Updates (CPU) architecture** demonstrated the most stability across all sizes, notably achieving the highest precision (0.89) at the 5K level. This indicates that while a massive prompt can eventually stabilize a Baseline model, the CPU architecture achieves superior state tracking with significantly less prompt overhead, maintaining a more consistent conversational flow regardless of context density.

Prompt Size	Baseline	RV	CPU	ESM
5K	0.86	0.81	0.89	0.79
10K	0.80	0.79	0.83	0.83
20K	0.94	0.81	0.88	0.88

Table 4: Average Global State Precision (GSP) (GPT-5.1).

6.3 Memory Strategy Impact

Performance differences are primarily driven by memory strategy:

- Baseline: No structured memory; minimal latency but weak state tracking.
- RV: Structured validation improves tracking but introduces synchronization overhead.
- ESM: Deterministic routing via full schema extraction; high latency and failure sensitivity.

- CPU: Minimal state representation; optimal balance of latency and adherence.

Architecture	Memory Strategy	Primary Impact
Baseline	None	High speed, low context
Reactive Validator (RV)	Comprehensive JSON	Improved tracking, moderate overhead
ESM	Comprehensive JSON	High latency, deterministic routing
CPU	JSON but with only a few fields	Optimized latency, probabilistic routing

Table 5: Summary of Memory Influence.

6.4 Failure Analysis

Despite sophisticated prompt engineering and routing layers, all tested architectures exhibited specific degradation patterns. Below are the root causes of instruction-following collapse in multi-turn workflows.

6.4.1 Instruction Priority Collapse

LLMs inherently prioritize conversational fluidity and rapport-building over procedural checklists. This results in a persistent "priority collapse", where mandatory foundational steps are bypassed regardless of system prompt size.

- **The Introduction Gap:** The `skipped_stage: introduction` violation was the most persistent error, occurring in 14 of 18 Baseline calls and 15 of 18 RV calls. Agents routinely jumped straight to discovery without fully identifying their company or the reason for the call.
- **Missed instructions in closing:** In ESM architecture, agents occasionally missed mandatory closing instructions (e.g., confirming an email or proposing a specific time) immediately after securing a meeting agreement, which resulted in structurally inconsistent outcomes where the agent successfully booked a meeting but failed to confirm required logistical details, producing interactions that were technically complete but operationally invalid.
- **Premature Transition**: In Baseline trials, agents frequently skipped the discovery phase entirely. By delivering a value proposition and requesting a meeting before gathering minimal prospect data, the model prioritized the "Close" goal over the "Discovery" prerequisite.

6.4.2 Recency Bias Dominance

Under conversational load, models over-weight the immediate preceding user utterance, allowing local context to override global structural directives.

- **Objection Misclassification:** In the CPU architecture, Global State Precision (GSP) collapsed during objection handling because the Parser LLM interpreted minor resistance as terminal rejections. Instead of executing the `objection_handling` protocol, this forced a premature jump to the `Close` or termination stage without achieving any meaningful objection resolution.
- **Excessive Discovery Loops:** Conversely, when users provided answers to pain-point questions, the agent could go into clarification cycles asking discovery questions even after sufficient information had been gathered to do pitching. The agent failed to transition to the Pitch stage because it became hyper-fixated on the immediate conversational thread.

6.4.3 Role Drift Under Load

While explicit Identity Drift Rate (IDR) remained low, the architectures exhibited significant "Role Drift", a failure to maintain behavioral constraints and persona consistency when the system's state-tracking mechanisms lagged or errored.

- **Observer-Induced Drift:** In Reactive Validator (RV) tests, incomplete state extractions (e.g., failing to identify the Decision Maker despite clear confirmation) forced the system to inject state regressions. The agent, following the injected state, would regress in conversational state, entering repetitive verification loops that disrupted forward progression.
- **Latency-Induced Guardrail Failure**: Asynchronous behaviour of the Observer and the Primary Agent can create a "race condition" between the model's response generation and the safety injector. If a terminal violation, such as a domain incompatibility, is detected, the guardrail signal may arrive after the agent has already committed to its next turn. This results in scenarios, where the agent continues a prohibited interaction because the architectural intervention came late.

6.4.4 Working Memory Saturation

The External State Machine (ESM) architecture exposed the limits of utilizing LLMs for synchronous, high-schema state extraction as conversation length increases.

- **Schema Extraction Failure:** As the token count grew, Parser LLM accuracy decayed. This led to "State Stalling," where the user explicitly agreed to a meeting, but the Parser failed to update the `meeting_confirmed` field. Consequently, the agent continued pitching instead of transitioning to the Close stage.
- **Termination Blindness:** A related failure occurred during call exits. When users issued clear termination signals (e.g., "Goodbye"), the Parser frequently failed to update the `meeting_ended_by_user` field, leading to the agent continuing the interaction beyond an appropriate termination point.
- **Identity Detection Lag:** the start of the call was also impacted. The Parser often failed to recognize that a Decision Maker was already on the line, preventing the router from entering the Discovery phase and leaving the agent stuck in Gatekeeping logic.

6.4.5 Cross-Model Pattern Degradation

Primary open-weight evaluation revealed failure modes not present in the GPT baseline, but also cases where open-weight models resolved GPT failure patterns. For example, Llama reduced excessive discovery loops, though it sometimes overshot directly to a premature pitch. Model substitution involves trade-offs, not a uniform regression.

- **Technical Leakage:** In Llama Baseline trials, the agent occasionally exposed internal state representations to the user (e.g., explicitly stating "no function call was required" or "staying in discovery stage").
- **Role Inversion:** In Llama RV scenarios, the agent reversed roles under conversational pressure, asking the user "How can I help you today?" as if it were the customer service recipient.
- **Abrupt Termination:** Llama-RV frequently truncated calls immediately after Decision Maker confirmation, hanging up without executing the pitch or close protocols.
- **Premature Pitch:** In CPU configurations, Llama and Qwen models occasionally bypassed Discovery questions to deliver the Pitch prematurely, prioritizing close-goals over information-gathering prerequisites.

- **Validator Amplification:** For Llama, the Reactive Validator (RV) performed significantly worse than the Baseline (WAR 0.51 vs 0.60). The high latency of OpenRouter meant that validator corrections often arrived "out-of-sync" with the conversational turn, causing the model to become confused by its own corrective injections.

6.5 Cross-Model Comparison

The GPT-5.1 protocol was replicated with Llama-3.1-70B and Qwen-3.5-27B via OpenRouter. The table below shows WAR averaged across all prompt sizes.

Architecture	GPT-5.1	Llama-3.1-70B	Qwen-3.5-27B
Baseline	0.79	0.60	0.80
RV	0.77	0.51	0.76
CPU	**0.86**	**0.86**	**0.82**
ESM	0.80	0.82	0.81

Table 6: Mean Workflow Adherence Rate (WAR) by architecture and model (averaged across 5K / 10K / 20K prompts).

Key Finding: CPU and ESM architectures are model-robust, sustaining 0.80–0.86 WAR across all families. Notably, Qwen-3.5-27B matches GPT-5.1 performance across most configurations, confirming that local models are viable for this use case when the right architecture is selected. RV is highly model-sensitive, collapsing to 0.51 with Llama-3.1-70B and 0.76 with Qwen-3.5-27B. The RV drop in open-weight configurations is likely infrastructure-driven: RV depends on near-real-time synchronization between the observer and the primary agent, and OpenRouter latency pushes corrections out of turn. This result should not be interpreted as a ceiling on RV capability; direct provider APIs or self-hosted inference are expected to recover significant performance. GPT-5.1 improves with prompt size; Llama-3.1-70B exhibits **inverse scaling** (0.68→0.51).

6.5.1 Latency and Cost

Open-weight models carry a significant latency surcharge from OpenRouter routing overhead. Mean turn latency at the recommended 5K prompt size and model cost comparisons are shown below. This gap is an infrastructure artifact; switching to direct provider APIs or self-hosted inference (e.g., vLLM) is the primary lever for closing it.

Model	Input $/M	Output $/M	Avg WAR	Best Fit
GPT-5.1	$0.83	$10.00	0.81	Primary agent; Baseline at 20K
Llama-3.1-70B	$0.40	$0.39	0.70	Parser/Observer in CPU or ESM; avoid as RV agent
Qwen-3.5-27B	$0.195	$1.56	0.80	Cost-sensitive Baseline or RV; large-scale sweeps

Table 7: Model cost and architectural best-fit (Avg WAR across all architectures and prompt sizes).

Comparative Finding: Llama-3.1-70B is 25x cheaper per output token than GPT-5.1 and excels at schema extraction. However, OpenRouter latency (up to 19s) makes it unsuitable for real-time synchronization, particularly in RV configurations.

6.6 Architecture Comparison: The Latency and Complexity Bottleneck

Theoretical adherence gains must be weighed against computational latency. The table below shows mean turn latency for CPU and ESM (GPT-5.1) across prompt sizes; cross-model latency figures are in the Latency and Cost table above.

Architecture	5K (ms)	10K (ms)	20K (ms)
Continuous Prompt Updates	2883.7	3057.6	3208.4
External State Machine	4547.8	4991.2	5108.6

Table 8: Mean Turn Latency across dynamic architectures (GPT-5.1).

- **ESM:** Mean latency of 4.5–5.1s makes it unsuitable for real-time applications. When probabilistic extraction fails, deterministic routing breaks entirely.
- **CPU:** Averaging 2.9s at 5K by avoiding full schema generation, CPU is the closest dynamic architecture to a real-time threshold.

6.6.1 Implementation ROI: Accuracy vs. Failure Realism

A critical qualitative distinction exists between static and dynamic architectures: the **nature** of their failures. CPU and ESM produce structurally incoherent conversational breakdowns when the Parser misclassifies intent, e.g., locking into Excessive Discovery Loops or missing a clear meeting confirmation. Baseline violations (`skipped_stage: introduction`, `premature_move: pitch`) mimic an overly eager human agent and are far less disruptive to the conversational experience.

Summary: Dynamic architectures shift reliability onto the Parser/Observer LLM. For real-time streaming, Baseline or RV are most appropriate. When adherence is primary, CPU is the optimal choice, as incremental injection keeps latency manageable, though it requires Parser tuning. The cross-model extension confirms the architectural hierarchy (CPU > ESM > RV > Baseline) holds across all three model families. The dominant open-weight trade-off is OpenRouter latency, not adherence quality; infrastructure selection matters more than model choice for production viability.

7 Discussion

7.1 Cross-Model Behavior: Architecture Matters More Than Model Choice

Across all three evaluated models (GPT-5.1, Llama-3.1-70B, Qwen-3.5-27B), the most consistent result is that **architecture, not model selection, is the primary determinant of workflow adherence**.

In both Continuous Prompt Updates (CPU) and External State Machine (ESM) architectures, all models converged to similar Workflow Adherence Rates (WAR), typically in the 0.80–0.86 range depending on prompt size and scenario. While qualitative differences in behavior were observed (e.g., Llama exhibiting premature pitch transitions where GPT-5.1 exhibited excessive discovery loops), these represent opposite manifestations of the same underlying failure: **routing errors introduced by the Parser or control mechanism**. The behavioral contrast reflects model-

specific tendencies in how the Parser overshoots or undershoots stage transitions, not fundamentally different failure classes.

This reveals a critical structural insight: in multi-agent architectures, reliability is no longer bounded by the capabilities of the primary conversational model, but by the accuracy of the **control layer**. In practice, this transforms instruction-following from a single-model limitation into a **multi-agent coordination problem**, where failures arise from misalignment between actor and controller rather than deficiencies in language understanding.

In contrast, model differences become more pronounced in architectures with minimal external control. In Baseline and Reactive Validator (RV) configurations, Llama-3.1-70B exhibited significantly lower adherence, while Qwen-3.5-27B tracked closer to GPT-5.1. This suggests that model selection matters most in **unstructured or weakly controlled systems**, but becomes secondary once strong architectural constraints are introduced.

Taken together, these results directly answer RQ4: **the relative performance ordering of architectures remains stable across model families**, indicating that instruction-following degradation is a model-agnostic phenomenon and that architectural interventions generalize reliably across providers.

7.2 Prompt Scaling and Context Competition

Increasing system prompt size from 5K to 20K characters produced limited and inconsistent improvements in adherence. While the Baseline architecture showed gradual gains with larger prompts, advanced architectures (CPU, ESM) exhibited flat or degraded performance at higher prompt sizes.

This behavior is best explained by **context competition**. In static prompting, larger prompts increase the likelihood that relevant instructions remain salient, improving adherence. However, in architectures that dynamically inject instructions (CPU, ESM), static prompt content competes with runtime routing signals for attention within the model's context window.

As prompt size increases, the relative influence of injected control signals diminishes, reducing the effectiveness of the architecture. This explains why smaller prompts (5K) consistently achieved comparable or superior performance in CPU and ESM configurations while also reducing latency.

These findings suggest that prompt scaling is not a universal solution: it is beneficial primarily in static systems, but can actively interfere with dynamically controlled architectures. The effect is most consistent for open-weight models (Llama, Qwen), where ESM and CPU performance was flat or declining from 5K to 20K. For GPT-5.1, CPU showed marginal gains at larger sizes, indicating the interference effect is partially model-dependent.

7.3 Model-Specific Behavioral Differences

Although architecture dominates overall performance, several consistent behavioral differences emerged.

Qwen-3.5-27B demonstrated the most stable WAR cross-architecture performance among open-weight models, closely matching GPT-5.1 in most configurations. However, this parity is limited to adherence scores: Qwen ESM exhibited judge-reported identity drift in 17–33% of runs per prompt size, compared to 0% in GPT-5.1 ESM, indicating that qualitative stability under strong architectural control is not fully equivalent. Llama-3.1-70B performed well in parser-driven roles

but showed weaker adherence in Baseline and RV setups, including premature termination and occasional exposure of internal reasoning.

GPT-5.1 remained the strongest overall performer in static prompting scenarios, particularly at higher prompt sizes, but its advantages diminish under strong architectural control where all models converge to similar performance levels.

Importantly, no model fully resolves multi-turn instruction-following degradation on its own. Even the strongest models exhibited stage violations and routing errors, reinforcing the necessity of architectural intervention.

7.4 Latency and Infrastructure Constraints

Latency emerged as a critical constraint in production viability. Architectures such as CPU and ESM require additional model calls per conversational turn, introducing measurable delays relative to Baseline systems.

This effect is amplified when using proxy infrastructure such as OpenRouter, where routing overhead can significantly increase response times. In contrast, direct provider APIs or self-hosted inference (e.g., vLLM) can substantially reduce latency and make advanced architectures more viable in real-time settings.

These findings emphasize that **infrastructure choice is as important as model or architecture selection**. In many cases, the primary bottleneck is not model capability but system-level latency introduced by orchestration and routing layers. Production trade-offs across architectures are detailed in Section 4 (Architecture Comparison); the key conclusion is that CPU at 2.9s and ESM at 4.5–5.1s represent the GPT-5.1 ceiling, with open-weight models adding further overhead via OpenRouter.

8 Conclusion

This study explored different ways to make conversational AI agents follow instructions more reliably during multi-step conversations. We tested several system designs using three language models: GPT-5.1, Llama-3.1-70B, and Qwen-3.5-27B. From the experiments, four main lessons emerged.

The way the system is built matters more than the model itself. LLM models on their own often lose track of structured workflows during longer conversations. But when we add a “manager” layer that guides the conversation step by step, performance improves significantly. In other words, giving the LLM structure is more effective than simply upgrading the model.

Bigger prompts are not always better. Adding more instructions to the prompt can help in simple setups, but in more advanced systems it often has little benefit - and can even make things worse. Shorter, more focused prompts tend to work just as well while being faster and easier to manage.

Open-source models can perform competitively. Models like Llama and Qwen were able to match the performance of more advanced models in structured systems. This means companies don’t always need expensive proprietary models to build effective conversational agents - as long as the system design is solid.

There is a trade-off between accuracy and speed. More reliable systems usually require extra steps behind the scenes, which makes them slower. Simpler systems respond faster but are more likely to make mistakes. Choosing the right approach depends on the use case: real-time voice applications may prioritize speed, while chat-based systems can afford slightly slower but more accurate responses.

In summary, the key to building reliable conversational AI is not just better models, but better system design. The most effective solutions combine clear structure, lightweight prompts, and carefully chosen infrastructure to balance accuracy, cost, and responsiveness.

9 Limitations and Trade-offs

9.1 Model Coverage and Generalization

While this study extends beyond a single-model setup by incorporating GPT-5.1, Llama-3.1-70B, and Qwen-3.5-27B, the evaluation still represents a limited slice of the rapidly evolving LLM landscape. Other model families (e.g., Claude, Gemini, smaller distilled models, or fine-tuned variants) may exhibit different degradation profiles, particularly in how they balance conversational fluency against procedural adherence. However, a key finding of this work is that relative architectural performance remains stable across model families. This suggests that while absolute adherence scores may shift with newer models, the underlying trade-offs between architectures are likely to generalize, even as base model capabilities improve.

9.2 Infrastructure-Induced Variance

The cross-model evaluation introduced an important confounding variable: inference infrastructure. Open-weight models (Llama, Qwen) were accessed via OpenRouter, introducing significant latency overhead.

This creates critical limitations:

- Observed failures in Reactive Validator (RV) are partially attributable to asynchronous timing misalignment, not purely model capability.
- Architectural conclusions regarding latency and synchronization are therefore tightly coupled to deployment infrastructure, not just model behavior.

As a result, the performance of open-weight models in production may differ substantially when deployed via direct provider APIs or self-hosted inference (e.g., vLLM).

9.3 Scripted vs. Real Users

All prospect interactions were conducted by researchers following predefined conversation scripts. Real prospects exhibit considerably higher variability in language, pacing, interruptions, and off-topic statements. Domain incompatibility scenarios, in particular, were tested in a controlled format that may underrepresent the ambiguity present in actual simulations. Thus, real-world degradation may be more severe and less predictable than observed in this study.

9.4 Domain Specificity

The experimental proxy (B2B technology sales with a six-stage linear workflow) represents one specific task structure. Industries with non-linear, highly context-sensitive workflows (e.g., healthcare intake, financial advisory, customer support triage) may exhibit different degradation profiles. The introduction-skipping bias observed consistently across all architectures may reflect training data characteristics specific to the chosen evaluation domain.

9.5 Function Calling Constraints

Function calling (or tool calling) was successfully implemented as a utility layer across all architectures (e.g., executing the `end_call` or `ask_further_assistance` functions). However, it was not investigated as a primary mitigator for instruction-following degradation. Function calling fundamentally relies on the same internal model memory as strict prompting. Attempting to use a model's intrinsic generation capabilities to solve its own context saturation and recency bias introduces a circular dependency constraint. Furthermore, extensive tool libraries actively consume context window capacity, which can aggravate the very cognitive load and recency bias problems we aim to solve. This limitation is corroborated by both industry documentation concerning API tool limits [11] and prior research, justifying our exclusive focus on externalized architectural controls for adherence enforcement.

9.6 Alternative Architectures Investigated

During the initial scoping of mitigating strategies, a publisher-subscriber model implemented via a frame-based Pipecat parallel pipeline [12] was technically spiked and verified. This architecture utilizes a background validator that evaluates the agent's response in real time, injecting corrections and regenerating the output before streaming. While highly effective at correcting errors natively, regeneration is strictly unsuitable for real-time applications because the initial erroneous message may have already been partially synthesized or spoken, causing confusing conversational repetitions or stuttering. However, this pattern is highly recommended for scenarios interacting with internal APIs or utilizing complex function calling. In such non-streaming contexts, if an initial output generation fails, the pub/sub pipeline can intercept and regenerate the proper output safely before any result is surfaced to the user.

9.7 Text vs. Voice Differences

While the CPU and ESM architectures were evaluated via a text-based CLI, the Baseline and Reactive Validator utilized the Deepgram Voice Agent [13]. Voice integration adds constraints not present in text-based LLM evaluations: speech recognition errors, latency from transcription, interruption handling, and the impossibility of visual formatting cues. Results from text-based instruction-following benchmarks should not be assumed to transfer to streaming, unscripted deployments without empirical validation. The Observer timing vulnerability documented in the Reactive Validator, where injections arrive one full agent turn after the triggering event, is an asynchronous constraint that does not exist in non-streaming text architectures.

10 Recommendations

Based on the empirical findings, we offer the following actionable guidance for teams deploying LLM agents on structured multi-turn workflows.

Prioritize architecture over model selection.

Across GPT-5.1, Llama-3.1-70B, and Qwen-3.5-27B, architectural choice consistently dominated performance outcomes. CPU and ESM architectures converged to similar adherence levels across all models, while Baseline and RV showed higher model sensitivity. Teams should prioritize selecting the correct control architecture before optimizing model choice.

Align architecture choice with latency constraints.

For real-time voice or latency-sensitive systems, Baseline or Reactive Validator (RV) are the most viable options for latency. However, both carry important caveats. Baseline performance is model-dependent: GPT-5.1 Baseline at 20K achieves the highest WAR, but Qwen Baseline at 20K underperforms its own 10K configuration. RV is effective primarily on happy-path calls; on objection-heavy workflows, GPT-5.1 RV objection WAR declined vs Baseline (0.81 $\rightarrow$ 0.68 at 20K), and Llama RV collapsed to 0.51 avg WAR. RV should therefore not be selected solely on latency grounds without validating against the expected call distribution. CPU becomes viable for near-real-time systems when latency is kept below 3 seconds per turn via optimized infrastructure.

Consider the CPU architecture for applications prioritizing adherence.

The Continuous Prompt Updates (CPU) architecture consistently achieves the highest overall Workflow Adherence Rate (WAR). Because it incrementally injects stage instructions rather than replacing the entire system prompt each turn, it avoids massive token overhead. While the secondary Parser LLM introduces some latency (2.9 seconds), this injection method makes it a viable option for real-time agent architectures where a slight conversational pause is acceptable, as well as the premier choice for asynchronous chat deployments.

Reserve Pub-Sub regeneration for background functions.

As investigated, while the parallel publish-subscribe architecture provides powerful error trapping, applying generation retries directly to streaming paths produces conversational stuttering. This architecture should be strictly recommended only for internal tool calling or background API orchestration where the evaluator can safely retry structural constraints out of sight from the end user.

Prefer 5K system prompts with dynamic control.

For open-weight models (Llama, Qwen) in CPU and ESM setups, 5K prompts match or outperform larger sizes while reducing latency. For GPT-5.1, the difference across prompt sizes in CPU is small (WAR 0.84–0.88) and within noise; 5K remains a safe default. Larger prompts should be reserved primarily for Baseline systems where no dynamic control layer is present, particularly when objection handling is critical.

Treat the ESM as a latency-prohibitive reference architecture.

The External State Machine demonstrates the ceiling of deterministic routing, but its 4.5-5.1 second mean latency makes it unsuitable for latency-sensitive applications. Its failure on probabilistic state extraction (schema stalling, termination blindness) also indicates that LLMs are ill-suited for synchronous, high-schema state extraction across long calls. ESM-style determinism is better implemented as a rule layer on top of a lighter Parser, not as the Parser itself.

Implement a semantic fast-path incompatibility detector.

Both the RV and ESM architectures failed to intercept domain incompatibility signals before the agent had already responded. To eliminate this failure mode at negligible latency cost, we recommend deploying a fast, lightweight semantic classifier (such as an optimized Sentence-Transformer or a local BERT model) ahead of the LLM observer cycle. Unlike brittle regex or keyword filters, a semantic classifier understands context, correctly distinguishing between a negative trait and a negation of that trait (e.g., identifying "not a small business" as a positive signal) while maintaining near-zero latency overhead.

Address the Introduction Gap directly.

The single most persistent violation across all 216 calls and all four architectures was `skipped_stage: introduction`. A session-start injection (delivered before the first agent response, via a static prepend to the initial context window) can enforce Introduction compliance without requiring an LLM observer cycle.

11 Future Work

The findings from this study open several directions for continued investigation.

Infrastructure-aware benchmarking. Future work should isolate model capability from infrastructure effects by benchmarking:

- Self-hosted inference (e.g. vLLM)
- Direct provider APIs

This will clarify whether observed latency-driven failures are architectural or infrastructural.

Scaling across model sizes and tiers. The current study evaluates large models (70B+, GPT-class). Future experiments should explore:

- Smaller distilled models (7B–13B)
- Fine-tuned domain-specific models

This would allow us to map the cost vs. adherence vs. latency frontier more precisely.

Fine-tuning for adherence. Supervised fine-tuning on trajectories that correctly execute the full stage sequence (with negative examples penalising stage skipping) may produce models that internalise procedural constraints more robustly than in-context instruction.

Reinforcement Learning for adherence. Using WAR and SVF as reward signals in an RL training loop would allow the base model to be shaped specifically toward stage compliance, potentially eliminating the recency bias degradation observed in this study.

Multi-agent controller architectures. The CPU and ESM architectures each use a single secondary LLM for all routing decisions. Decomposing this into specialised micro-agents (a stage router, a memory extractor, an incompatibility detector) may improve both latency and reliability by reducing the cognitive load on any single LLM call.

Longer memory horizons. The current observer architectures extract and store only current-turn state. Maintaining a rolling structured summary of the full call history (updated incrementally) may improve extraction accuracy on long calls and reduce the schema stalling and identity detection lag observed in the ESM architecture.

12 References

Bibliography

[1] Z. Epstein and others, "MMMT-IF: A Challenging Multimodal Multi-Turn Instruction Following Benchmark," *arXiv preprint*, 2024.

[2] N. F. Liu *et al.*, "Lost in the Middle: How Language Models Use Long Contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[3] X. Chen, R. A. Adams, K. Schuster, and E. H. Chi, "Premise Order Matters in Reasoning with Large Language Models," *arXiv preprint arXiv:2402.08939*, 2024.

[4] Y. Sun and others, "Parrot: Enhancing Multi-Turn Instruction Following for Large Language Models," *arXiv preprint*, 2024.

[5] Y. Liu, X. Zeng, C. Shao, F. Meng, and J. Zhou, "Instruction Position Matters in Sequence Generation with Large Language Models," in *Findings of the Association for Computational Linguistics: ACL 2024*, Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 11652–11663. doi: 10.18653/v1/2024.findings-acl.693.

[6] Y. Leviathan, M. Kalman, and Y. Matias, "Prompt Repetition Improves Non-Reasoning LLMs," *arXiv preprint arXiv:2512.14982*, 2025.

[7] Q. Jia *et al.*, "One Battle After Another: Probing LLMs' Limits on Multi-Turn Instruction Following with a Benchmark Evolving Framework," *arXiv preprint arXiv:2511.03508*, 2025.

[8] Q. Wu *et al.*, "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation," *arXiv preprint arXiv:2308.08155*, 2023, doi: 10.48550/arXiv.2308.08155.

[9] Y. Wu, T. Yue, S. Zhang, C. Wang, and Q. Wu, "StateFlow: Enhancing LLM Task-Solving through State-Driven Workflows," *arXiv preprint arXiv:2403.11322*, 2024, doi: 10.48550/arXiv.2403.11322.

[10] LangChain, "LangGraph: Building Stateful, Multi-Actor Applications with LLMs." [Online]. Available: https://github.com/langchain-ai/langgraph

[11] OpenAI, "Function calling - OpenAI API Documentation." [Online]. Available: https://platform.openai.com/docs/guides/function-calling

[12] P. AI, "Pipecat: Open Source Framework for Voice and Multimodal Agents." [Online]. Available: https://github.com/pipecat-ai/pipecat

[13] Deepgram, "Deepgram Voice Agent API Documentation." [Online]. Available: https://developers.deepgram.com/docs/voice-agent

[14] Confident-AI, "DeepEval: The Open-Source LLM Evaluation Framework." [Online]. Available: https://docs.confident-ai.com/

13 Appendix

13.1 A. Repository

The full source code, system prompts, conversation scripts, raw transcripts, and evaluation outputs are available in the internal research repository: **AAI Labs - Instruction Following Research** (available in the internal AAI Labs research repository; access on request).

13.2 B. System Prompts

Three system prompt size variants were used across all four architectures. The prompts share identical core stage rules and behavioural constraints; larger variants extend them with example dialogues, objection libraries, and company knowledge.

Label	Path	Characters
5K	`data/test_scenarios/prompts/5k.txt`	4,331
10K	`data/test_scenarios/prompts/10k.txt`	9,649
20K	`data/test_scenarios/prompts/20k.txt`	23,239

13.3 C. Test Scripts and Evaluation Tools

All test execution and evaluation tooling referenced in this paper is available in the repository under the `tools/` directory:

- `tools/run_test_scenario.py` : CLI runner supporting all evaluated architectures and models via the `--architecture` and `--llm-model` flags; drives the LLM Agent with a specified scenario JSON.
- `tools/run_eval_sweep.py` : Evaluation sweep runner; applies the Judge LLM and DeepEval [14] metrics to all transcripts under a given architecture's output directory.
- `src/evaluators/judge_evaluator.py` : LLM-as-a-Judge evaluator (GPT-5.2).
- `src/evaluators/deepeval_metrics.py` : DeepEval [14] metric wrapper.