Attentive Reasoning Queries: A Systematic Method for Optimizing Instruction-Following in Large Language Models

Bar Karov, Dor Zohar and Yam Marcovitz

NLP Research, Emcie Co Ltd.

*Corresponding author(s). E-mail(s): bar@emcie.co;
Contributing authors: dor@emcie.co; yam@emcie.co;

Abstract

We present Attentive Reasoning Queries (ARQs), a novel structured reasoning approach that significantly improves instruction-following in Large Language Models through domain-specialized reasoning blueprints. While LLMs demonstrate remarkable capabilities across diverse tasks, they often fail to maintain adherence to complex, use-case-specific instructions during multi-turn conversations, presenting challenges for business-critical applications. ARQs address this limitation by guiding LLMs through systematic reasoning steps with targeted queries that reinstate critical instructions and facilitate intermediate reasoning throughout the completion process. In extensive testing within Parlant, our framework for reliable customer-facing agents in which ARQs were born out of necessity, they achieved a 90.2% success rate across 87 test scenarios, outperforming both Chain-of-Thought reasoning (86.1%) and direct response generation (81.5%). ARQs showed particular strength in addressing persistent failure modes like guideline re-application and hallucination prevention. Our analysis also revealed that ARQs can potentially be more computationally efficient than free-form reasoning when carefully designed. These findings demonstrate that structured reasoning approaches provide effective mechanisms for controlling how LLMs process information and make decisions in complex scenarios.

Keywords: Large Language Models, Attentive Reasoning Queries, Structured Reasoning, Conversational AI, Chain-of-Thought, Reasoning Framework, Reasoning Optimization, Hallucination Prevention, Customer-facing AI

Supplementary Materials: Source code, prompt examples and other supplementary materials are available on our GitHub.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse tasks, from knowledge retrieval to creative content generation [1, 2]. However, ensuring that these models perform systematic, reliable reasoningparticularly in multi-turn conversational settingsremains challenging [3]. LLMs often struggle with hallucinations, remembering instructions, and maintaining consistent reasoning patterns across complex tasks. These challenges are especially pronounced in high-stakes customer-facing applications, such as a bank's customer service where a dynamic and temporal understanding of the context, and adherence to specific behavioral guidelines in relation to it, are critical.

Traditional approaches to enhancing LLM reasoning, such as free-form chain-of-thought prompting [4] or step-by-step instruction, have shown promise but offer limited control over how models process information. While these methods encourage models to "think aloud," they provide minimal structure to guide the reasoning process through domain-specific considerations or known failure modes. Furthermore, as conversation context grows, LLMs often fail to maintain focus on critical instructions and constraints that should govern their behavior [5, 6].

In this paper, we introduce Attentive Reasoning Queries (ARQs), a structured approach to guide LLMs through systematic reasoning steps using targeted, task-specific queries. ARQs leverage domain knowledge to redirect the model's attention to critical instructions, decisions, and potential pitfalls at the points where such attention is most crucial. This approach serves two key functions: (1) Reinstating important instructions, (2) Facilitating intermediate reasoning steps. These functions are particularly instrumental in complex and nuanced conversational contexts in which adherence to specific instructions is essential.

We implement and evaluate ARQs within Parlant [7], a framework for developing reliable conversational AI agents suitable for business-specific customer-facing applications. This framework requires agents to maintain strict adherence to behavioral guidelines, appropriately utilize available tools, and avoid hallucinations. By structuring the reasoning process through predefined JSON schemas with targeted queries, we test how ARQs can enhance performance across key processing components.

2 Related Work

2.1 Prompting Techniques for Reasoning

Many recent advances in LLM reasoning capabilities have been driven by specialized prompting techniques. Chain-of-Thought (CoT) prompting [4] demonstrated that eliciting intermediate reasoning steps before producing answers significantly improves performance on complex tasks. Chain-of-Verification (CoVe) [8] extends this approach

by having models explicitly verify their outputs against potential errors. Other variations include zero-shot CoT [9] using simple prompts like "Let's think step by step" and Tree-of-Thought (ToT) approaches [10] that explore multiple reasoning pathways.

While effective, these general prompting strategies provide limited guidance on task-specific reasoning steps. They depend largely on the model's internal capabilities to determine appropriate reasoning patterns and typically lack domain-specific guidance that could prevent common failure modes or highlight critical considerations.

2.2 Conversational Agents

Conversational agents built on LLMs often incorporate additional structures to maintain coherence, adhere to guidelines, and extend functionality. ReAct [11] pioneered the integration of reasoning and action, while frameworks like LangChain [12] provide infrastructure for tool use and workflow management.

These frameworks often treat the core reasoning process as a black box, with limited mechanisms to guide how the LLM processes information or makes decisions. This creates challenges in ensuring consistent adherence to complex behavioral guidelines, particularly in business-critical customer-facing applications. Recent work highlights both the importance of explicit planning mechanisms and the challenges of tool integration [13]. Studies show LLMs often struggle to select appropriate tools or provide correct parameters [14].

To address these challenges, recent advancements have focused on enhancing the interpretability and control of LLMs within conversational agents. For instance, the development of LangGraph [15] extends LangChain's capabilities by introducing an orchestration framework for building stateful agents. Such approaches seek to address the difficulty of maintaining attention to numerous complex instructions by allowing more granular control over agent processing stages, restricting the set of provided instructions to the conversational contexts in which they are most relevant.

Parlant [7] is a new open-source framework which addresses these challenges through managed, supervised guidelines. It implements a dynamic control system that evaluates conversational context against a structured repository of behavioral guidelines. The framework follows a modular architecture with specialized processing components for guideline filtering and matching, tool calling, and message generation, each operating with explicit reasoning protocols using ARQs. In this work, we use Parlant's test suite as a controlled environment to evaluate whether ARQs can improve reasoning performance over CoT in complex and nuanced conversational contexts.

2.3 Common Pitfalls in LLM-based Systems

LLMs exhibit several persistent failure modes in conversational applications that limit their reliability. Some notable pitfalls are:

- **Alignment drift** occurs when models gradually deviate from specified guidelines over extended conversations [16].
- **Hallucination** the generation of factually incorrect information, or the offering of unwarranted services, represents a related challenge to alignment drift [17].

As models drift from alignment constraints, they tend to generate content lacking proper grounding in the provided context. This is particularly problematic for domain-specific agents that must maintain factual accuracy within their expertise areas.

- **Context forgetfulness** manifests as pronounced recency bias [18], where models preferentially attend to information near the end of their context window while neglecting earlier content. This creates significant challenges in extended conversations where important instructions or context may be buried among various exchanges and constraints.

3 Our Contributions

This paper makes the following contributions:

1. **ARQ Methodology:** We introduce Attentive Reasoning Queries (ARQs) as a structured approach to guide LLM reasoning, which demonstrate how domain knowledge can be incorporated into reasoning blueprints to address task-specific challenges and known failure modes.
2. **ARQ Implementation:** We implement ARQs within Parlant, a framework for developing conversational agents supporting strict quality requirements with respect to agent behavioral patterns and tool-use.
3. **Empirical Evaluation:** We conduct an evaluation comparing ARQ performance against both Chain-of-Thought and no-reasoning (control) approaches, demonstrating that ARQs achieve superior performance, particularly in addressing challenging failure modes such as guideline re-application and hallucination prevention.

The dataset and code used to produce our experiment is available on our GitHub (see title page), along with the full code for the Parlant framework.

4 Attentive Reasoning Queries (ARQ)

As a motivating example, consider how we might choose a restaurant for a group dinner. Our decision process follows a structure. First, we review the available restaurants in terms of dietary preferences, budget, and location. Next, we evaluate each individual's preferences with respect to these options.

Importantly, this pattern of systematic review followed by structured evaluation can be abstracted and applied across many similar tasks.

This reasoning process inspires our development of Attentive Reasoning Queries (ARQs). ARQs guide Large Language Models (LLMs) through systematic reasoning blueprints. This is achieved by requiring responses to follow a predefined JSON schema in which keys are pre-defined, pinpointed queries designed to direct the model's attention to relevant information. The values are then filled by an LLM during its response completion process.

These queries can either be general-purpose or domain-specific, depending on the task. For example, for the described task, a reasonable reasoning chain would be answering the following questions before coming to a decision:

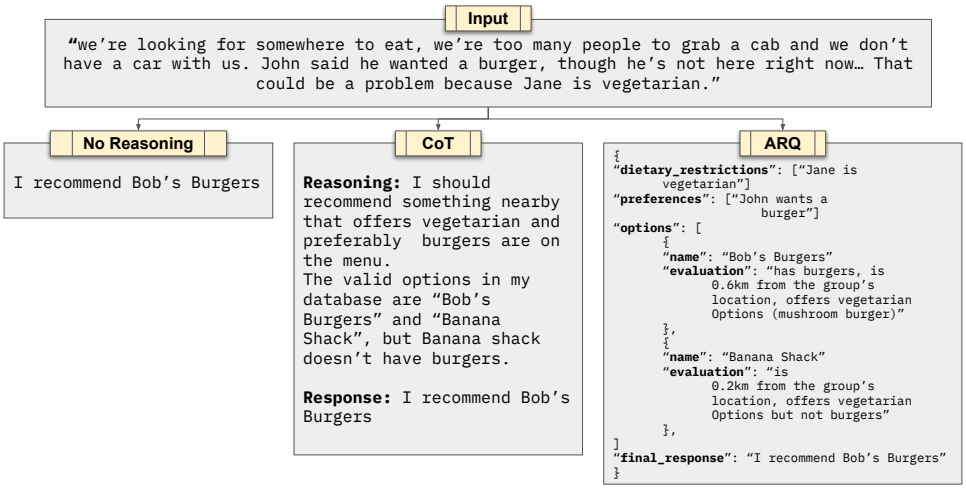

Fig. 1 Output examples: (1) using ARQs, (2) using CoT, and (3) performed without reasoning, all leading to the same final output. We assume the agent is equipped with information about nearby restaurants and their menus.

- What are the constraints of the group (dietary, financial, or otherwise)?
- What restaurants are open and within range?
- For each of these restaurants, how suitable is their menu to our constraints?

Given this desired reasoning blueprint, we can prompt an LLM to respond to these queries before arriving at its final recommendation. This is accomplished by instructing the model to return a JSON object (or another structured format compatible with the chosen LLM) containing evaluations for each query in the sequence. See Figure 1 for a simple implementation example.

This structured approach also makes extracting the model's final answer easier, compared to other reasoning methods, as conclusions appear in specific query responses rather than within lengthy reasoning text, simplifying both human review and automated processing.

The ARQ-guided process consists of the following steps:

1. **Leading ARQ Phase:** The LLM processes a sequence of pre-determined leading questions that serve three key functions:
   - Reinstating critical instructions
   - Reinstating important contextual information from the prompt
   - Facilitating step-by-step reasoning and intermediate computations
2. **Response Generation:** Based on the reasoning from the leading query phase, the LLM produces a response.

3. **(Optional) Response Verification:** The LLM evaluates if its suggested response satisfies all requirements, by answering pre-defined verification questions (E.g, "Is the response consistent with the constraints listed in the leading ARQ stage?"). If not, a new response candidate is generated and verified, until a satisfactory one is found.

4.1 Query Design

By leveraging domain knowledge about specific tasks, ARQs can be designed to target known failure modes and critical reasoning steps. Unlike free-form CoT or CoVe approaches that rely on generic prompting strategies for nonspecific use cases, ARQs can be crafted to address task-specific challenges and guide the LLM through sensitive decision points.

For example, in information retrieval tasks, ARQs can guide the LLM to systematically identify relevant sources, assess their reliability, cross-reference claims across multiple documents, and check for temporal consistency and contextual relevance before synthesizing a response.

Critical ARQs can be identified either through analytic decomposition of a complex reasoning process, or, better yet, through experimentation and feedback. An additional important benefit of ARQs emerges in practice, as their corresponding completions often help identify potential contradictions, misinterpretations, or lack of contextual grounding that might be overlooked in open-ended reasoning methods.

ARQs leverage a key characteristic of LLM behaviorthe enhanced recall of information that appears near the end of the input context. ARQs are designed to make the LLM reiterate critical instructions using the leading-query completions just-in-time, right before attending to the completion of the required output. Leveraging these properties of autoregressive models, this recency effect, wherein the LLM reiterates critical information at the end of its context window, empirically helps maintain important constraints and requirements in the LLM's active context during response generation.

This approach may also provide additional benefits through the LLM's attention mechanism. Specifically, we hypothesize that responding to leading queries which ask the LLM to repeat critical instructions allows the LLM to highlight and thus establish stronger attention patterns between task-specific input (such as different instances of user queries) and general instructions (such as how to handle user queries). See Figure 2 for an example of such an ARQ. However, a detailed investigation of this attention-based hypothesis falls outside the scope of this paper.

5 Setting

To evaluate the efficacy of Attentive Reasoning Queries (ARQs) in real-world applications, we test them in the scope of a Conversational AI engine for customer-service use casesa domain where systematic reasoning and adherence to specific guidelines can be particularly challenging and consequential.

The reference engine design described here forms part of Parlant, a framework for developing reliable conversational AI agents suitable for customer-facing applications.

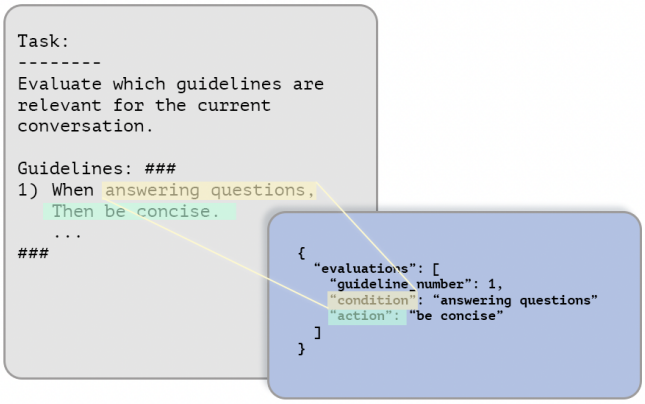

Fig. 2 An illustration of query-attention hypothesis: an ARQ asking the LLM to parrot the guideline before evaluating its relevance

In Parlant, each agent is initialized with four key components provided by its designer:

1. **Agent Profile:** A concise, natural-language description defining the agent's purpose and operational scope.
2. **Behavioral Guidelines:** A collection of conditional instructions in the form of "When ⟨X⟩ Then ⟨Y⟩" statements (e.g., "When discussing weather Then use metric units").
3. **Tool Suite:** A set of external functions accessible via structured API methods, enabling the agent to retrieve information or execute actions within its environment.
4. **Domain Lexicon:** A comprehensive glossary of domain-specific terms, essential to the agent's operational context.

The agent engages in multi-turn conversations while maintaining four critical constraints: (1) Strict adherence to its prescribed guidelines, (2) Appropriate utilization of available tools, (3) Accurate application of information provided in its profile and domain lexicon, and (4) Elimination of hallucinated information or services not explicitly authorized by its designer.

5.1 Engine Architecture

To fulfill these requirements, agent responses undergo a modular processing pipeline with specialized LLM calls using predefined prompt templates. When processing a user message, the agent executes the following sequence, as shown in Figure 3:

- **Guideline Proposition:** Identifies which guidelines are applicable to the current state of the conversation.
- **Tool Calling:** Determines optimal tool selection and parameter configuration based on user intent and conversation state.
- **Message Generation:** Crafts a final response incorporating selected guidelines and tool outputs.

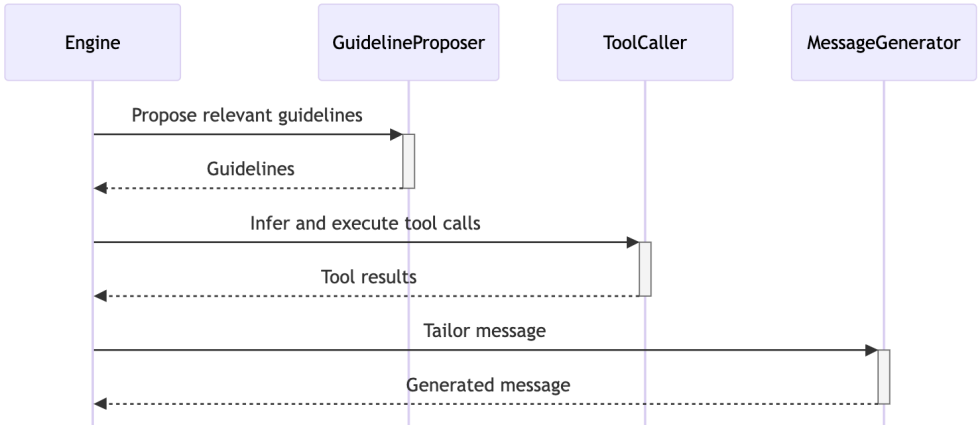

Fig. 3 A diagram of the Parlant Engine, including how its different modules interact. Modules execute from left to right, where each module feeds the next one with its outputs. The message generator receives comprehensive inputs from all other modules, before tailoring a final response.

Guideline proposition and tool calling operate in an iterative cycle, as tool call results may activate additional guidelines. For example, if a geolocation tool identifies that a user is located in Europe, this might trigger a guideline of the form "When the user is from Europe Then use metric units."

Additionally, each tool in the system is explicitly attached to at least one guideline, creating a controlled structure that governs tool access. A tool can only be called if its associated guideline has been determined to be active in the current context. This constraint ensures that tool usage aligns with the designer's intent and prevents inappropriate actions.

5.1.1 Guideline Proposition

A guideline in our framework consists of a condition (the "When" clause) and an action (the "Then" clause). The Guideline Proposer module determines which guidelines should be active given the current conversation state. This determination is more nuanced than simple condition matching and requires contextual reasoning.

The Guideline Proposer receives as input:

1. **Interaction History:** All past messages and tool call results.
2. **Agent Profile & Domain Lexicon:** The agent's role description and domain-specific terminology.
3. **Staged Tool Calls:** Results from recently executed tools.
4. **Guidelines to Filter:** The complete set of available guidelines.

It then outputs a score from 1 to 10, indicating how strongly each guideline currently applies. Guidelines with a score of 6 or more are activated. This score is also carried over to the Message Generator, using it to prioritize possibly conflicting guidelines.

Guideline activation follows a decision process that considers temporal context. To see why simple condition-matching is not sufficient, consider two guidelines with

identical conditions: "When the customer is ordering a pizza Then offer a 2-for-1 special" and "When the customer is ordering a pizza Then never recommend pineapple topping." The first should activate once and then become inactive after execution, while the second must remain active throughout the ordering process.

To handle these nuances, we implement the following activation protocol:

1. **Condition Evaluation:** Determine if the guideline's condition applies to the current context.
2. **Continuity Assessment:** Classify the action as either one-time or continuous. Continuous actions (like maintaining a specific tone or avoiding particular recommendations) remain active as long as their condition applies.
3. **Previous Application Check:** For non-continuous actions, verify if the action has already been performed.
4. **Reactivation Analysis:** For non-continuous actions previously performed, determine if the condition became false and then true again, warranting reactivation.

Our implementation includes a few additional nuanced rules for guideline reapplication that address edge cases such as partially fulfilled multi-part actions and assessing action parts as either cosmetic or functional in terms of their impact on the conversation. For brevity, these details are included in Appendix A, along with the guideline propoesr ARQ design used in Parlant.

5.1.2 Tool Calling

The Tool Caller module is responsible for determining which tools should be executed given the currently active guidelines.

The Tool Caller receives the following inputs:

1. **Interaction History:** All past messages and tool call results.
2. **Agent Profile & Domain Lexicon:** Contextual information about the agent's role and terminology.
3. **Active Guidelines:** The set of guidelines determined to be active by the Guideline Proposer.
4. **Available Tools:** Tools attached to active guidelines, each with its own parameter requirements and descriptions.
5. **Staged Tool Calls:** Results from tools already executed in the current processing cycle.

The Tool Caller follows specific instructions, provided to it in its prompt, to determine when and how tools should be activated, focusing on both contextual relevance and practical considerations:

1. **Proactive Tool Usage:** Tools may be suggested even when they don't directly address the user's latest message, if they could advance the conversation to a more productive state.
2. **Multiple Invocations:** Each tool may be called multiple times with different arguments within a single response cycle. For example, a product search tool might be called separately for each product category mentioned by the user.

3. **Call Duplication Prevention:** The system avoids calling a tool with identical arguments more than once, unless there is a clear justification, such as refreshing potentially outdated information.
4. **Dependency Management:** Each tool call is designed to be self-contained, relying only on information available in the immediate context or from already-executed tool calls. This prevents dependency chains that could fail when executed in parallel.

To further understand the tool caller's process, and to examine the ARQs it uses in Parlant, see Appendix B.

5.1.3 Message Generation

The Message Generator is the final module in the processing pipeline, responsible for synthesizing the outputs from previous stages into a coherent, contextually appropriate response to the user.

The Message Generator receives the following inputs:

1. **Interaction History**
2. **Agent Profile & Domain Lexicon**
3. **Active Guidelines:** Active guidelines, as determined by the guideline proposer module.
4. **Tool Call Results:** Data retrieved from any tools executed by the tool caller.

The Message Generator is provided with several preset instructions in its prompt that apply to all interactions, independent of the specific guidelines it receives. These instructions ensure consistent, high-quality responses across varied conversation contexts and cover aspects such as communication best-practices, information handling, and presentation format. For the complete set of instructions provided to the Message Generator and its ARQ implemenation, see either Appendix C, or the full Message Generator prompt in the supplementary materials on Github.

The Message Generator also handles guideline prioritization during response synthesis rather than in the earlier guideline proposition stage. This design choice avoids additional cross-batch LLM calls that would increase system latency. When resolving conflicts, the module considers both the applicability scores assigned during guideline proposition and specific conflict resolution instructions provided in its prompt.

To prevent hallucination, the Message Generator is explicitly instructed to offer only services and information explicitly provided in its context. This constraint ensures the agent doesn't suggest unauthorized optionsfor instance, a pizza delivery agent won't propose self-pickup unless this service was specifically included by the designer.

6 Experiment

6.1 Experimental Design

To empirically evaluate the effectiveness of ARQs, we implemented them within the Parlant framework described in Section 5. Our experiments were designed to compare ARQ performance against alternative reasoning approaches when deployed across

the three core modules of our system: Guideline Proposer, Tool Caller, and Message Generator.

For each module, we developed three methodologically distinct implementations:

1. **ARQ Implementation**: Employs the structured query-based reasoning approach described in Section 1, with module-specific queries designed to target known failure modes and critical decision points in each component. Care was taken to ensure that ARQs are general rather than overfitting the test cases.
2. **Chain-of-Thought (CoT) Implementation**: Incorporates free-form reasoning before generating the final output, allowing the LLM to develop its own reasoning pathway without the structured constraints of ARQs.
3. **Control Implementation**: Generates direct responses based on instructions, without any explicit reasoning process.

All implementations share identical base prompts, receiving the same instructions and functional requirements.

The evaluation framework is publicly available, including the full dataset of test scenarios, detailed evaluation criteria, and implementation code for all three reasoning methodologies. For further details see the attached Github repository.

6.1.1 In-Context Learning

To optimize performance across all implementations, we incorporated In-Context Learning (ICL) through carefully selected exemplars. Each module's prompt includes a set of few-shot examples that demonstrate successful execution patterns. These examples were iteratively refined based on observed failure modes in real customer interactions, and are identical across all 3 reasoning methods.

For the ARQ implementation, the examples include both the structured queries and their corresponding responses, modeling the expected reasoning pattern. For CoT and Control implementations, only the expected response is provided.

6.2 Evaluation Dataset

Our evaluation utilized a comprehensive dataset of 87 test cases, crafted to assess the system's adherence to framework requirements under diverse conversational scenarios. The dataset composition includes 22 scenarios focused exclusively on guideline proposition accuracy, and additional 65 comprehensive scenarios evaluating the full interaction pipeline (guideline proposition, tool calling, and response generation)

Each test case provides:

1. **Agent Configuration**: Profile description, behavioral guidelines, available tools, and domain lexicon
2. **Conversation History**: A sequence of user/agent interactions leading to the current state, ending with a user message to respond to
3. **Success Criteria**: Conditions that must be satisfied for the response to be considered correct and aligned

Response quality was assessed through multiple complementary approaches. Our primary evaluation used an LLM to judge whether responses met the predefined success criteria for each test case, examining both content accuracy (e.g., an LLM determines whether the agent's response satisfies the criteria "includes an offering of a 10% discount"). For scenarios requiring external tool usage, we evaluated whether the agent correctly identified when tools were needed, selected the appropriate tools from its available set, and provided the necessary parameters for successful execution.

Tests that apply exclusively to the guideline proposer were evaluated based on the set of guidelines that the guideline proposer suggested. A guideline proposer test is considered successful only if the correct set of guidelines is proposed.

6.3 Language Model

All experiments were performed using OpenAI's GPT-4o model family [19], with specific versions selected based on extensive testing in real-world Parlant applications. We used the gpt-4o-2024-11-20 version for the Tool Caller module and the gpt-4o-2024-08-06 version for both the Guideline Proposer and Message Generator modules. Temperature settings were configured as follows: 0.1 for the Message Generator, 0.15 for the Guideline Proposer, and 0.05 for the Tool Caller.

6.4 Results

We conducted experiments comparing the performance of each reasoning method (Control, Chain-of-Thought, and ARQ) across all three modules in our framework. Each test in the dataset was run 5 times to account for the stochastic nature of LLM outputs.

Reasoning Method	**Guideline Proposer Tests (%)**	**Comprehensive Tests (%)**	**Total (%)**
None	70.43	85.31	81.54
CoT	80.87	87.81	86.05
ARQ	**84.24**	**92.19**	**90.17**

Fig. 4 Performance comparison across reasoning methods. Comprehensive tests evaluate all three modules (message generator, tool caller, and guideline proposer) working in conjunction.

As shown in Figure 4, ARQs achieved the highest success rate on our dataset, outperforming both Chain-of-Thought and the Control setting where no reasoning was performed during the completion stage.

Our analysis revealed that tests passed exclusively by ARQ (failing under CoT) generally fall into two categories:

- **Guideline re-application:** Tests requiring nuanced decisions about the reactivation of guidelines that were previously followed in the agent's earlier responses.
- **Hallucination prevention:** Tests specifically designed to detect whether the agent offers hallucinated facts or services not supported by its available tools or context.

Based on our experience deploying conversational agents in production environments, these two failure cases represent some of the most challenging adherence issues for LLM-based systems. This fact highlights the ability of ARQs to target critical fail-points in the decision process, as structured reasoning queries can be strategically designed and added to address the most persistent weaknesses in the decision process.

6.4.1 Computational Efficiency

Module	Control	CoT	ARQ
Message Generator	54	330	596
Tool Caller	68	180	550
Guideline Proposer	48	405	289

Fig. 5 Average output token usage by module and reasoning method.

Our analysis reveals that the specific design of ARQs and the nature of the underlying task significantly impact their computational efficiency relative to other reasoning methods, as measured by the output tokens required. As shown in Figure 5, token usage patterns vary across different modules, demonstrating that ARQs can be either more or less efficient than Chain-of-Thought depending on implementation choices and task characteristics.

The Guideline Proposer module demonstrates lower token usage with ARQs than with CoT, requiring 29% fewer tokens while delivering superior performance. This efficiency stems primarily from the nature of the task- determining whether guidelines are active or inactive, which naturally lends itself to structured queries with concise responses. The task also has fewer edge cases compared to other modules, and does not require generating extensive natural language outputs.

In contrast, the ARQ implementations for the Message Generator and Tool Caller modules consumed substantially more tokens. These modules face more complex tasks requiring autoregressive natural language generation and handling numerous edge cases, which results in more extensive reasoning through ARQs.

This variation across modules underscores a critical finding: The efficiency of structured reasoning approaches depends on both how the queries are formulated and the inherent complexity of the underlying task. When queries direct the model to focus precisely on the most relevant aspects of a decision process within naturally bounded tasks like classification, they can reduce computational overhead. These findings suggest that ARQ design should be approached strategically based on the specific reasoning requirements and characteristics of each task.

7 Limitations and Future Research

While our experiments demonstrate the potential efficacy of ARQs in enhancing the reasoning capabilities of conversational AI agents, several limitations of the current study suggest important directions for future research.

Our evaluation focused specifically on conversational agents operating within the Parlant framework, leaving open questions about ARQ applicability in other contexts. Additionally, our current evaluation dataset, though carefully designed to test specific capabilities, remains modest in size and scope. Future work should validate ARQ performance on substantially larger and more diverse datasets of conversational scenarios that were not used during ARQ development.

The generalizability of our findings is constrained by our exclusive use of GPT-4o as the underlying language model. Preliminary research done in Parlant, which is not presented in this work, suggests that the results are reproducible across different models, though full empirical testing of this hypothesis remains a subject for future research.

Perhaps most significantly, our investigation primarily focused on establishing whether ARQs confer performance benefits in a conversational agent framework, without exploring the broader design space of ARQ construction or optimization strategies. We plan to conduct a systematic exploration of ARQ design principles in future work. This future research will establish formalized methodologies for constructing ARQs optimized for particular reasoning tasks or domains.

8 Conclusion

In this work, we introduced Attentive Reasoning Queries (ARQs), a structured approach to guide the reasoning processes of Large Language Models. ARQs utilize targeted, domain-specific questions organized within a predefined JSON schema to direct model attention to critical instructions and decision points. We implemented and evaluated ARQs within the Parlant framework, testing their effectiveness in conversational agent applications that require strict adherence to behavioral guidelines. Our evaluation compared ARQs against Chain-of-Thought (CoT) reasoning, demonstrating that ARQs improves performance across the system's core modules.

8.1 ARQs vs. Chain-of-Thought

While both Chain-of-Thought and ARQs aim to enhance LLM reasoning capabilities, they differ fundamentally in their structure and implementation. CoT prompting encourages models to generate intermediate reasoning steps in a free-form manner before producing a final answer. This approach relies on the model's inherent capabilities with minimal external guidance. In contrast, ARQs provide explicit structural scaffolding through predefined queries that guide the model's attention to specific objects during the reasoning process. This approach offers several advantages:

- **Domain-Specific Guidance:** Unlike the general-purpose nature of CoT, ARQs incorporate domain knowledge to address task-specific challenges and known failure modes.
- **Enhanced Debuggability:** The structured format of ARQs allows system designers to more easily inspect and debug reasoning processes. When errors occur, designers can identify exactly which query or reasoning step fell short of the goal.

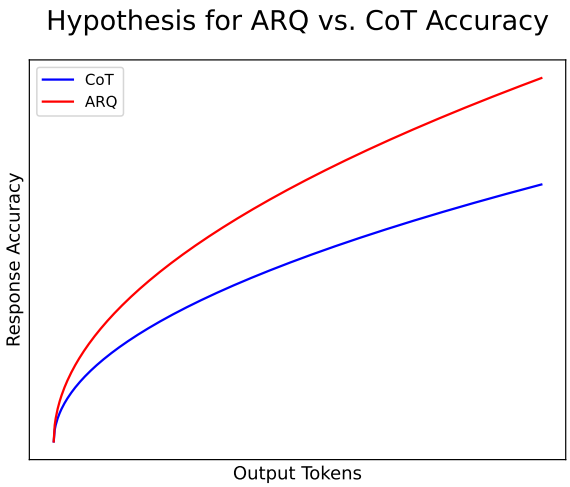

Fig. 6 Our hypothesis for how ARQ scales compared to CoT, as a factor of the reasoning length

- **Attention Preservation:** ARQs strategically reinstate critical instructions and constraints at key decision points, addressing the "lost in the middle" phenomenon where important information receives less attention from the model.

As shown in Figure 6, and in accordance with our practical experience, we hypothesizethough do not test within this paperthat as reasoning complexity increases, requiring more output tokens, both methods would show improved performance, but ARQs would likely scale more effectively. By explicitly directing the model's focus throughout extended reasoning chains, ARQs potentially avoid the degradation in reasoning quality that often occurs with longer free-form reasoning.

References

[1] Zhu, Y., Yuan, H., Wang, S., Liu, J., Liu, W., Deng, C., Chen, H., Liu, Z., Dou, Z., Wen, J.-R.: Large language models for information retrieval: A survey. arXiv preprint arXiv:2308.07107 (2023)

[2] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., *et al.*: A survey on evaluation of large language models. ACM transactions on intelligent systems and technology **15**(3), 1–45 (2024)

[3] Zhang, C., Dai, X., Wu, Y., Yang, Q., Wang, Y., Tang, R., Liu, Y.: A survey on multi-turn interaction capabilities of large language models. arXiv preprint arXiv:2501.09959 (2025)

[4] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., *et al.*: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems **35**, 24824–24837 (2022)

[5] Dong, Z., Tang, T., Li, J., Zhao, W.X., Wen, J.-R.: Bamboo: A comprehensive benchmark for evaluating long text modeling capacities of large language models. arXiv preprint arXiv:2309.13345 (2023)

[6] Li, T., Zhang, G., Do, Q.D., Yue, X., Chen, W.: Long-context llms struggle with long in-context learning. arXiv preprint arXiv:2404.02060 (2024)

[7] Emcie: Parlant: an open-source AI agent Framework. https://www.parlant.io (2025)

[8] Dhuliawala, S., Komeili, M., Xu, J., Raileanu, R., Li, X., Celikyilmaz, A., Weston, J.: Chain-of-verification reduces hallucination in large language models. arXiv preprint arXiv:2309.11495 (2023)

[9] Kojima, T., Gu, S.S., Reid, M., Matsuo, Y., Iwasawa, Y.: Large language models are zero-shot reasoners. Advances in neural information processing systems **35**, 22199–22213 (2022)

[10] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., Narasimhan, K.: Tree of thoughts: Deliberate problem solving with large language models. Advances in neural information processing systems **36**, 11809–11822 (2023)

[11] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., Cao, Y.: React: Synergizing reasoning and acting in language models. In: International Conference on Learning Representations (ICLR) (2023)

[12] Chase, H.: LangChain (2022). https://github.com/langchain-ai/langchain

[13] Sypherd, C., Belle, V.: Practical considerations for agentic llm systems. arXiv preprint arXiv:2412.04093 (2024)

[14] Shen, Z.: Llm with tools: A survey. arXiv preprint arXiv:2409.18807 (2024)

[15] Inc., L.: LangGraph: Build resilient language agents as graphs (2025). https://github.com/langchain-ai/langgraph

[16] Shen, T., Jin, R., Huang, Y., Liu, C., Dong, W., Guo, Z., Wu, X., Liu, Y., Xiong, D.: Large language model alignment: A survey. arXiv preprint arXiv:2309.15025 (2023)

[17] Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., *et al.*: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. ACM Transactions on Information Systems **43**(2), 1–55 (2025)

[18] Liu, N.F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., Liang, P.: Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics **12**, 157–173 (2024)

[19] OpenAI: GPT-4o-2024-08-06: Version of the GPT-4o multimodal model. https://platform.openai.com/docs/models/gpt-4o. Accessed: 2025-02-27 (2024)

A Guideline Proposer Prompt and ARQs

Examples for the full guideline proposer prompt, using all 3 reasoning modes, are available on our GitHub.

The instructions regarding guideline application and re-application, as specified in the guideline proposer's prompt, are:

```
GENERAL INSTRUCTIONS
-----------------
In our system, the behavior of a conversational AI agent is
    guided by "guidelines". The agent makes use of these
    guidelines whenever it interacts with a user (also referred to
     as the customer).
Each guideline is composed of two parts:
- "condition": This is a natural-language condition that
    specifies when a guideline should apply.
           We look at each conversation at any particular state,
               and we test against this
           condition to understand if we should have this
               guideline participate in generating
           the next reply to the user.
- "action": This is a natural-language instruction that should be
     followed by the agent
           whenever the "condition" part of the guideline applies
               to the conversation in its particular state.
           Any instruction described here applies only to the
               agent, and not to the user.


Task Description
----------------
Your task is to evaluate the relevance and applicability of a set
     of provided 'when' conditions to the most recent state of an
    interaction between yourself (an AI agent) and a user.
These conditions, along with the interaction details, will be
    provided later in this message.
For each condition that is met, determine whether its
    corresponding action should be taken by the agent or if it has
     already been addressed previously.


Process Description
--------------------
```

```
a. Examine Interaction Events: Review the provided interaction
    events to discern the most recent state of the interaction
    between the user and the agent.
b. Evaluate Condition/s: Assess the entire interaction to
    determine whether each condition is still relevant and
    directly fulfilled based on the most recent interaction state.
c. Check for Prior Action: Determine whether the condition has
    already been addressed, i.e., whether it applied in an earlier
     state and its corresponding action has already been performed
    .
d. Guideline Application: A guideline should be applied only if:
     (1) Its condition is currently met and its action has not
         been performed yet, or
     (2) The interaction warrants re-application of its action (e.
         g., when a recurring condition becomes true again after
         previously being fulfilled).

For each provided guideline, return:
     (1) Whether its condition is fulfilled.
     (2) Whether its action needs to be applied at this time. See
         the following section for more details.

Insights Regarding Guideline re-activation
-------------------
A condition typically no longer applies if its corresponding
    action has already been executed.
However, there are exceptions where re-application is warranted,
    such as when the condition is re-applied again. For example, a
     guideline with the condition "the customer is asking a
    question" should be applied again whenever the customer asks a
     question.
Additionally, actions that involve continuous behavior (e.g., "do
     not ask the user for their age", or guidelines involving the
    language the agent should use) should be re-applied whenever
    their condition is met, even if their action was already taken
    .
If a guideline's condition has multiple requirements, consider it
     continuous if at least one of them is continuous. Actions
    like "tell the customer they are pretty and help them with
    their order" should be considered continuous, since 'helping
    them with their order' is continuous.
Actions that forbid certain behaviors are generally considered
    continuous, as they must be upheld across multiple messages to
     ensure consistent adherence.

IMPORTANT: guidelines that only require you to say a specific
    thing are generally not continuous. Once you said the required
     thing - the guideline is fulfilled.
```

```
Conversely, actions dictating one-time behavior (e.g., "send the
    user our address") should be re-applied more conservatively.
Only re-apply these if the condition ceased to be true earlier in
     the conversation before being fulfilled again in the current
    context.

IMPORTANT: Some guidelines include multiple actions. If only a
    portion of those actions were fulfilled earlier in the
    conversation, treat the guideline as though it has been fully
    executed.
In such cases, re-apply the guideline only if its condition
    becomes true again later in the conversation, unless it is
    continuous.
```

When using ARQs, the guideline proposer is instructed to return a dictionary whose keys are pre-defined questions, and values are the LLM's responses to these questions.

As an example, when evaluating the guideline:

- **Condition:** a client asks for a drink
- **Action:** check if the drink is available in stock

We end the guideline proposer prompt by telling the LLM to return the following[1]:

```
{
    "guideline_id":"...",
    "condition":"a client asks for a drink",
    "condition_application_rationale":"<Explanation for why the
       condition is or isn't met>",
    "condition_applies":"<BOOL>",
    "action":"check if the drink is available in stock",
    "guideline_is_continuous":"<BOOL: Optional, only necessary if
        guideline_previously_applied is true. Specifies whether
       the action is taken one-time, or is continuous>",    "
       capitalize_exact_words_from_action
    _in_the_explanations_to_avoid_semantic_pitfalls":true,
    "guideline_previously_applied_rationale":{
        "<action_segment_1>":"<explanation of whether this action
            segment was already applied; to avoid pitfalls, try
           to use the exact same words here as the action segment
            to determine this. use CAPITALS to highlight the same
            words in the segment as in your explanation>",
        "<action_segment_N>":"<explanation...>"
    },
    "guideline_current_application_refers_to_a_new_or_subtly
```

[1] See ending of 'ARQ Guideline Proposer Example Prompt.txt' in the supplementary materials to view in .txt format, for better readability

```
    _different_context_or_information":"<if the guideline DID
        previously apply, explain here whether or not it needs to
        re-apply due to it being applicable to new context or
        information>",
    "guideline_previously_applied":"<str: either 'no', 'partially
        ' or 'fully' depending on whether and to what degree the
        action was previously preformed>",
    "is_missing_part_cosmetic_or_functional":"<str: only included
         if guideline_previously_applied is 'partially'. Value is
        either 'cosmetic' or 'functional' depending on the nature
        of the missing segment.",
    "guideline_should_reapply":"<BOOL: Optional, only necessary
        if guideline_previously_applied is not 'no'>",
    "applies_score":"<Relevance score of the guideline between 1
        and 10. A higher score indicates that the guideline should
         be active>"

}
```

Where text in angled brackets represents our instruction to the LLM, rather than actual text it has to output.

These ARQs guide the LLM through an assessment of whether:

1. The condition currently applies to the conversation state
2. The action has been previously performed (fully, partially, or not at all)
3. The guideline represents continuous or one-time behavior
4. The current context warrants re-application of previously fulfilled guidelines

When receiving a response from the LLM, the guideline in question can be identified by its ID, and the guideline becomes active or inactive depending on its `applies_score`. When receiving a response from the LLM, the guideline at question can be identified by its ID, and the guideline becomes active or inactive depending on its `applies_score`. For guidelines that have been previously applied, we also require the returned value for the key `guideline_should_reapply'` to be `true`.

B Tool Caller Instructions and ARQs

The tool caller is responsible for determining which tools should be executed based on the current conversation state and active guidelines. Below are the instructions provided to the tool caller in its prompt, along with its expected output format while in ARQ mode:

```
TASK DESCRIPTION
-----------------
Your task is to review the provided tool and, based on your most
    recent interaction with the customer, decide whether to use it
    .
For the provided tool, assign a score from 1 to 10 to indicate
    its usefulness at this time, where a higher score indicates
    that the tool call should execute.
```

```
For any tool with a score of 5 or higher, provide the arguments
    for activation, following the format in its description.

While doing so, take the following instructions into account:
1. You may suggest tools that d o n  t directly address the
    c u s t o m e r  s latest interaction but can advance the
    conversation to a more useful state based on function
    definitions.
2. Each tool may be called multiple times with different
    arguments.
3. Avoid calling a tool with the same arguments more than once,
    unless clearly justified by the interaction.
4. Ensure each tool call relies only on the immediate context and
     staged calls, without requiring other tools not yet invoked,
    to avoid dependencies.
5. Use the "should_run" argument to indicate whether a tool
    should be executed, meaning it has a high applicability score
    and either (a) has not been staged with the same arguments, or
     (b) was staged but needs to be re-executed.
6. If a tool needs to be applied multiple times (each with
    different arguments), you may include it in the output
    multiple times.


Produce a valid JSON object according to the following format:
‘‘‘json
{
    "last_customer_message": "<REPEAT THE LAST USER MESSAGE IN
        THE INTERACTION>",
    "most_recent_customer_inquiry_or_need": "<customer's inquiry
        or need>",
    "most_recent_customer_inquiry_or_need_was_already_resolved":
        <BOOL>,
    "name": "<TOOL NAME>",
    "subtleties_to_be_aware_of": "<NOTE ANY SIGNIFICANT
        SUBTLETIES TO BE AWARE OF WHEN RUNNING THIS TOOL IN OUR
        AGENT'S CONTEXT>",
    "tool_calls_for_candidate_tool": [
        {
            "applicability_rationale": "<A FEW WORDS THAT EXPLAIN
                 WHETHER AND HOW THE TOOL NEEDS TO BE CALLED>",
            "applicability_score": <INTEGER FROM 1 TO 10>,
            "argument_evaluations": <EVALUATIONS FOR THE
                ARGUMENTS. CAN BE DROPPED IF THE TOOL SHOULD NOT
                EXECUTE>,
            "same_call_is_already_staged": <BOOL>,
            "comparison_with_rejected_tools_including_
            references_to_subtleties": "<A VERY BRIEF OVERVIEW OF
                 HOW THIS CALL FARES AGAINST OTHER TOOLS IN
                APPLICABILITY>",
```

```
            "relevant_subtleties": "<IF SUBTLETIES FOUND, REFER
                TO THE RELEVANT ONES HERE>",
            "a_rejected_tool_would_have_been_a_better_fit_if_it
            _werent_already_rejected": <BOOL>,
            "potentially_better_rejected_tool_name": "<IF
                CANDIDATE TOOL IS A WORSE FIT THAN A REJECTED TOOL
                , THIS IS THE NAME OF THAT REJECTED TOOL>",
            "potentially_better_rejected_tool_rationale": "<IF
                CANDIDATE TOOL IS A WORSE FIT THAN A REJECTED TOOL
                , THIS EXPLAINS WHY>",
            "the_better_rejected_tool_should_clearly_be_run_in_
            tandem_with_the_candidate_tool": <BOOL>,
            "should_run": <BOOL>
        }
        ...
    ]
}
```

These ARQs guide the LLM through an evaluation process that includes:

1. Identifying the customer's most recent inquiry or need
2. Determining the applicability score of the tool (1-10)
3. Evaluating each required parameter to determine if it is available in context
4. Assessing if the same tool call is already staged
5. Comparing the current tool with other potential tools

When determining parameter values, the tool caller analyzes each parameter's availability and appropriateness using ARQs, checking if:

- The parameter is provided in the current context
- The parameter should principally be provided by the customer
- The parameter was already provided and needs to be provided again
- It would be problematic to guess the parameter value if not provided

A tool is executed if the the LLM's response for it has `should_run` set to `true` in the ARQ response.

An example of a full tool caller prompt, which includes these instructions in txt format, is available in the supplementary materials.

C Message Generator Prompt and ARQs

The Message Generator module is the final component in the processing pipeline, responsible for synthesizing previous module outputs into a coherent response. It follows these instructions, which are provided to it in its prompt:

```
TASK DESCRIPTION:
-----------------
Continue the provided interaction in a natural and human-like
    manner.
```

```
Your task is to produce a response to the latest state of the
    interaction.
Always abide by the following general principles (note these are
    not the "guidelines". The guidelines will be provided later):
1. GENERAL BEHAVIOR: Craft responses that feel natural and human-
    like. Keep them concise and polite, striking a balance between
     warmth and brevity without becoming overly verbose.
2. AVOID REPEATING YOURSELF: When replying  avoid repeating
    yourself. Instead, refer the customer to your previous answer,
     or choose a new approach altogether. If a conversation is
    looping, point that out to the customer instead of maintaining
     the loop.
3. DO NOT HALLUCINATE: Do not state factual information that you
    do not know or are not sure about. If the customer requests
    information you're unsure about, state that this information
    is not available to you.
4. ONLY OFFER SERVICES AND INFORMATION PROVIDED IN THIS PROMPT:
    Do not output information or offer services based on your
    intrinsic knowledge - you must only represent the business
    according to the information provided in this prompt.
5. REITERATE INFORMATION FROM PREVIOUS MESSAGES IF NECESSARY: If
    you previously suggested a solution, a recommendation, or any
    other information, you may repeat it when relevant. Your
    earlier response may have been based on information that is no
     longer available to you, so it  s important to trust that it
     was informed by the context at the time.
6. MAINTAIN GENERATION SECRECY: Never reveal details about the
    process you followed to produce your response. Do not
    explicitly mention the tools, context variables, guidelines,
    glossary, or any other internal information. Present your
    replies as though all relevant knowledge is inherent to you,
    not derived from external instructions.
7. OUTPUT FORMAT: In your generated reply to the customer, use
    markdown format when applicable.

MESSAGE GENERATION MECHANISM
-----------------
To generate an optimal response that aligns with all guidelines
    and the current interaction state, follow this structured
    revision process:

1. INSIGHT GATHERING (Pre-Revision)
   - Before starting revisions, identify up to three key insights
       from:
     * Explicit or implicit customer requests
     * Relevant principles from this prompt
     * Notable patterns or conclusions from the interaction
   - Each insight should be actionable and directly relevant to
      crafting the response
   - Only include absolutely necessary insights; fewer is better
```

```
   - Document insights' sources for traceability

2. INITIAL RESPONSE
   - Draft an initial response based on:
     * Primary customer needs
     * Applicable guidelines
     * Gathered insights
   - Focus on addressing the core request first

3. REVISION CRITERIA
   The response requires further revision if any of these
       conditions are met:
   - Facts or services are offered without clear sourcing from
       this prompt - deonted by
       all_facts_and_services_sourced_from_prompt being false
   - Guidelines or insights are broken (except when properly
       prioritized, or when broken due to insufficient data) -
       denoted by either `instructions_broken_due_to_missing_data`
        or `instructions_broken_only_due_to_prioritization`
   - The response repeats previous messages - denoted by `
       is_repeat_message` being true.

4. REVISION DOCUMENTATION
   Document each revision in JSON format including:
   - Complete revised message
   - Facts and sources used
   - Services offered and their sources
   - Guidelines/insights followed and broken
   - Repetition assessment
   - Prioritization decisions and rationales
   - Missing data impacts

5. COMPLETION CRITERIA
   The revision process is complete when either:
   - All guidelines and insights are satisfied, or
   - 5 revisions have been attempted, or
   - Remaining issues are justified by:
     * Explicit prioritization decisions
     * Documented data limitations
     * Customer request conflicts


PRIORITIZING INSTRUCTIONS (GUIDELINES VS. INSIGHTS)
-----------------
Deviating from an instruction (either guideline or insight) is
    acceptable only when the deviation arises from a deliberate
    prioritization, based on:
    - Conflicts with a higher-priority guideline (according to
        their priority scores).
    - Contradictions with a customer request.
```

```
    - Lack of sufficient context or data.
    - Conflicts with an insight (see below).
In all other cases, even if you believe that a guideline's
    condition does not apply, you must follow it.

Guidelines vs. Insights:
Sometimes, a guideline may conflict with an insight you've
    derived.
For example, if your insight suggests "the customer is vegetarian
    " but a guideline instructs you to offer non-vegetarian dishes
    , prioritizing the insight would better align with the
    business's goals since offering vegetarian options would
    clearly benefit the customer.

However, remember that the guidelines reflect the explicit wishes
     of the business you represent. Deviating from them should
    only occur if doing so does not put the business at risk.
For instance, if a guideline explicitly prohibits a specific
    action (e.g., "never do X"), you must not perform that action,
     even if requested by the customer or supported by an insight.

In cases of conflict, prioritize the business's values and ensure
     your decisions align with their overarching goals.
```

The specific ARQs that the message generator responds to are:

```
'''json
{
    "last_message_of_customer": "Hey, can I order a large
        pepperoni pizza with Sprite?",
    "guidelines": [],
    "context_evaluation": {
        "most_recent_customer_inquiries_or_needs": <str, fill out
            accordingly>,
        "parts_of_the_context_i_have_here_if_any_with_specific_
        information_on_how_to_address_these_needs": "<fill out
            accordingly>",
        "topics_for_which_i_have_sufficient_information_and_can
        _therefore_help_with": "<fill out accordingly>",
        "what_i_do_not_have_enough_information_to_help_with
        _with_based_on_the_provided_information_that_i_have": "<
            fill out accordingly>",
        "was_i_given_specific_information_here_on_how_to
        _address_some_of_these_specific_needs": <BOOL>,
        "should_i_tell_the_customer_i_cannot_help_with_some_
        of_those_needs": <BOOL>
    },
    "insights": "[<Up to 3 original insights to adhere to>]",
    "evaluation_for_each_instruction": [
```

```
            {
                "number": 1,
                "instruction": "<Insight #1, if it exists>",
                "evaluation": "<your evaluation of how the
                    insight should be followed>",
                "data_available": "<explanation whether you are
                    provided with the required data to follow this
                     insight now>"
            },
            <Additional entries for all insights>

    ],
    "revisions": [
    {
        "revision_number": 1,
        "content": <response chosen after revision 1>,
        "factual_information_provided": [
            {
                "fact": <str, statement of a fact in the
                    suggested response>
                "source": <str, source of the fact - either a
                    specific part of this prompt or something else
                    >
                "is_source_based_in_this_prompt": <BOOL>
            },
            ...
        ],
        "offered_services": [
            {
                "service": <str, statement of a fact in the
                    suggested response>
                "source": <str, source of the fact - either a
                    specific part of this prompt or something else
                    >
                "is_source_based_in_this_prompt": <BOOL>
            },
            ...
        ],
        "instructions_followed": <list of guidelines and insights
             that were followed>,
        "instructions_broken": <list of guidelines and insights
            that were broken>,
        "is_repeat_message": <BOOL, indicating whether "content"
            is a repeat of a previous message by the agent>,
        "followed_all_instructions": <BOOL, whether all
            guidelines and insights followed>,
```

```
        "instructions_broken_due_to_missing_data": <BOOL,
            optional. Necessary only if
            instructions_broken_only_due_to_prioritization is true
            >,
        "missing_data_rationale": <STR, optional. Necessary only
            if instructions_broken_due_to_missing_data is true>,
        "instructions_broken_only_due_to_prioritization": <BOOL,
            optional. Necessary only if followed_all_instructions
            is true>,
        "prioritization_rationale": <STR, optional. Necessary
            only if instructions_broken_only_due_to_prioritization
             is true>
        "all_facts_and_services_sourced_from_prompt": <BOOL, if
            false, you must produce further revisions>,
        "further_revisions_required": <BOOL, true iff either
            instructions were broken due to invalid reasons, if
            is_repeat_message is true, or if
            all_facts_and_services_sourced_from_prompt is false>
    },
    ...
    ]
}
```

Where text in angled brackets represents our instruction to the LLM, rather than actual text it has to output.

These queries force explicit identification of:

1. Customer needs and available information
2. guideline applicability with reasoning
3. fact sourcing with guideline adherence tracking

The message generator includes verifying queries, meaning that its instructed to suggest responses and then evaluates them until a satisfactory response is generated. This revision process enables self-correction when guidelines are broken or hallucinations detected. The final response of the agent is taken from final revision in the message generator's output.