ARARA: A LLM-based Multi-Agent Development Framework for Conversational Recommender Systems

Fillipe Santos[1,2,3], Juliano Soares[1,2,3], Sildolfo Gomes[1,4], Julio Cesar dos Reis[1,3], and Marcelo S. Reis[1,2,3]

[1] Hub de Inteligência Artificial e Arquiteturas Cognitivas (H.IAAC), Campinas, Brazil
[2] Laboratório de Inteligência Artificial (Recod.ai), Campinas, Brazil
[3] Instituto de Computação, Universidade Estadual de Campinas (UNICAMP), Campinas, Brazil
[4] Instituto de Pesquisas Eldorado, Campinas, Brazil
{fillipesantos, juliano.soares, msreis, jreis}@ic.unicamp.br; sindolfogomes@gmail.com

Abstract. Large Language Models (LLMs) have accelerated the development of Conversational Recommender Systems (CRSs), enabling flexible language understanding and tool-augmented reasoning. However, most LLM-based CRS implementations embed architectural decisions—such as role specialization, memory access, and coordination logic—implicitly within prompts or tightly coupled pipelines. This conflation of reasoning structure with model capacity limits transparency, hinders systematic analysis, and makes architectural extensions ad hoc. We introduce ARARA, a modular, provider-agnostic framework that models CRS as a multi-agent orchestration problem. ARARA defines explicit abstractions for agents, users, modules, orchestration strategies, memory, and reusable skills, establishing governed execution semantics in which conversational behavior emerges from structured role specialization and controlled coordination. To evaluate structural effects, we instantiate ARARA on RecBench+ conversational recommendation benchmarks in the Movie and Book domains, comparing single-LLM baselines with matched multi-agent configurations under identical memory settings. Results show that architectural decomposition and governed routing systematically improve robustness, accuracy, and user satisfaction as reasoning complexity increases, particularly under implicit inference and deceptive inputs. Our findings demonstrate that CRS effectiveness is not solely a function of model capacity, but is critically shaped by architectural design. By elevating coordination, memory, and specialization to first-class architectural primitives, ARARA reframes CRS development as an inspectable and extensible engineering discipline grounded in structured multi-agent orchestration.

Keywords: Conversational Recommender Systems, Multi-Agent Frameworks, Large Language Models.

1 Introduction

Recommender Systems (RSs) play a central role in modern digital platforms, supporting decision-making across domains such as e-commerce and media [11,17]. As user needs become increasingly contextual and time-sensitive, recommendations can no longer be treated as a single prediction step, but as an interactive process shaped by evolving intents and constraints [14].

Conversational Recommender Systems (CRSs) respond to this shift by embedding recommendation within multi-turn dialogue [2,5]. Powered by Large Language Models (LLMs), these systems enable flexible language understanding and tool-augmented reasoning [11,13]. Yet, as conversational complexity increases, reasoning naturally decomposes into multiple interdependent sub-tasks rather than a single generation step.

A practical way to manage this complexity is to distribute responsibilities across specialized *actors*. Instead of relying on a monolithic prompt to implicitly perform all reasoning steps, conversational systems can model retrieval, normalization, verification, ranking, and memory conditioning as distinct components collaborating within a structured interaction [10,4]. Empirical studies indicate that such decomposition improves robustness, supports implicit queries, and mitigates misinformation [15]. These actors can be concretely implemented as *agents* with well-defined responsibilities and interaction boundaries.

Despite this promise, many multi-agent CRS implementations embed architectural decisions implicitly in prompts or tightly coupled pipelines. Questions regarding routing, memory access, tool invocation, and execution flow are often handled ad hoc. General-purpose LLM-agent frameworks provide coordination infrastructure, yet they are not tailored to the structural requirements of conversational recommendation [16,8]. As a result, the system architecture is frequently implicit rather than explicitly modeled and extensible.

This article introduces *ARARA*, a modular framework for developing multi-agent CRSs. We argue that advancing CRSs requires an explicit architectural foundation that models conversational reasoning as coordinated agent interaction rather than prompt engineering. The framework *ARARA* is composed of: **Agents**: components with clearly defined responsibilities (*e.g.*, retrieval, normalization, verification, ranking), operating as LLM-driven reasoning units; **User**: a specialized agent representing the human participant, interacting through the same message-based interface as other agents. Unlike autonomous agents, the User does not perform LLM-based generation; instead, messages are manually provided through an interaction interface, typically initiating the conversation with a user-issued query; **Modules**: reusable groupings of agents implementing higher-level reasoning patterns; **Orchestrators**: agents responsible for selecting the next component to act. Multiple orchestrators may coexist, and routing strategies may include LLM-based selection, random policies, round-robin strategies, user-driven selection, or custom algorithms; **Skills**: plug-and-play capabilities attachable to agents. Examples include *Vision* (image interpretation), *Web Search*, and controlled *Code Execution*.

Agents in ARARA are powered by LLMs. The framework adopts a provider-agnostic client interface, enabling transparent integration of different LLM backends. Current

integrations include widely used providers such as OpenAI[5], Anthropic[6], and OpenRouter[7], Groq[8], as well as locally hosted models. New providers can be incorporated by implementing additional client connectors, enabling the system to evolve alongside the rapidly changing LLM ecosystem. ARARA also supports an optional human-in-the-loop mode, distinct from the end user, in which a developer or supervisor may inject input during execution for experimentation, supervision, or debugging.

Memory in ARARA is accessed through skills and currently supports two primary forms: *episodic memory*, which records interaction histories and conversational events, and *shared memory*, which provides a common knowledge space accessible across agents. Both memory structures and their retrieval mechanisms can be customized according to application requirements, allowing developers to define how information is stored, accessed, and updated during execution.

By explicitly modeling users, agents, orchestration, memory, and capabilities, *ARARA* enables conversational behavior to emerge from coordinated interactions among agents. Extension points are exposed at multiple layers—including orchestration strategies, memory mechanisms, skills, and LLM backends—allowing developers to tailor the system to domain-specific CRS deployments.

To evaluate the architectural impact of ARARA, we instantiate it on structured conversational recommendation benchmarks from RecBench+ [7]. We compare single-LLM baselines with ARARA-based multi-agent configurations under matched memory settings, isolating the structural effects of specialization and orchestration across explicit, implicit, and misinformed queries. Rather than claiming state-of-the-art performance, our objective is to quantify the extent to which explicit architectural decomposition and orchestration shape system behavior under varying reasoning conditions.

This study provides the following original contributions: **(i)** present ARARA as a composable architectural framework for developing multi-agent CRSs, integrating users, agents, orchestrators, memory, skills, and provider-agnostic LLM backends within a unified execution model; **(ii)** provide a controlled empirical evaluation demonstrating how architectural decomposition and orchestration influence CRS effectiveness under varying reasoning conditions.

The remainder of this article is organized as follows. Section 2 reviews related work; Section 3 details the ARARA framework; Section 4 presents the experimental methodology; Section 5 reports our obtained experimental results; Section 6 discusses our findings and their implications; and Section 7 presents the concluding remarks.

2 Related Work

We examine existing LLM-based systems for conversational and agent-oriented applications. Prior work is categorized into single-agent and multi-agent approaches, with an emphasis on architectural properties directly relevant to ARARA's design, including

[5] https://openai.com/
[6] https://www.anthropic.com/
[7] https://openrouter.ai/
[8] https://groq.com/

Table 1: Structural comparison of ARARA with representative multi-agent LLM frameworks discussed in this section across five architectural dimensions relevant to conversational recommendation design.

Dimension	ARARA	BabyAGI	CAMEL	Multi-Agent Debate	AutoGen	LangGraph	MetaGPT
Modular Composition	✓	✗	✗	✗	✗	✗	partial
Orchestration Strategy	dynamic & customizable	static	static	static	flexible	graph-based	static
User Explicitly Modeled	✓	✗	✗	✗	✗	✗	✗
Memory as Architectural Primitive	✓	partial	✗	✗	partial	✗	✗
Designed for CRS	✓	✗	✗	✗	✗	✗	✗

modular composition, orchestration strategy, explicit user modeling, memory handling, and domain specificity.

Single-agent systems rely on a single LLM instance augmented with tools, memory, or prompting strategies, without explicit multi-agent coordination. Early CRSs adopt monolithic pipeline architectures combining dialogue management and recommendation backends [5]. Recent LLM-based recommenders integrate language models as rankers or conversational layers over retrieval components [14], improving linguistic flexibility while keeping reasoning centralized in a single instance. Tool-augmented frameworks such as Haystack[9] and DSPy[10] support dynamic workflows and external tools, but do not define modular abstractions for coordinated multi-agent orchestration.

Multi-agent systems distribute reasoning across multiple LLM instances or role-based agents. To enable a structural comparison aligned with ARARA's architectural perspective, we analyze representative frameworks across five dimensions: **(i) Modular Composition**: whether agents are organized into reusable higher-level modules rather than interacting as isolated roles; **(ii) Orchestration Strategy**: how the next acting component is selected, including static interaction patterns, graph-based execution, or dynamic runtime routing; **(iii) User Explicitly Modeled**: whether the user is represented as an architectural participant within the execution loop; **(iv) Memory as Architectural Primitive**: whether memory is treated as an explicit architectural component rather than an auxiliary prompt mechanism; **(v) Designed for CRS**: whether the framework is explicitly tailored to conversational recommender systems rather than serving as a general-purpose multi-agent infrastructure.

Representative frameworks are summarized in Table 1. **BabyAGI** [12], **CAMEL** [9], and **Multi-Agent Debate** [3] implement role-based or deliberative interactions under largely static coordination schemes, without modular composition tailored to CRSs. **AutoGen** [16] and **LangGraph** [8] provide flexible communication and routing infrastructures, yet remain general-purpose frameworks without CRS-specific architectural abstractions. **MetaGPT** [6] organizes role-based agents into structured workflows oriented toward software generation rather than CRSs.

[9] https://haystack.deepset.ai/
[10] https://dspy.ai/

While existing multi-agent approaches demonstrate the benefits of distributed reasoning, they typically prioritize generic coordination infrastructures or task-specific interaction patterns. In contrast, ARARA positions conversational recommendation architecture itself as the primary design object, integrating modular composition, explicit orchestration, reusable memory and skill abstractions, and domain-specific structure within a unified framework for CRSs. ARARA builds upon AutoGen [16], inheriting its multi-agent communication and orchestration abstractions while extending them to address the structural demands of conversational recommender systems, and further incorporates architectural principles from LangGraph and CrewAI [8,1], particularly graph-based coordination, modular composition, and role-oriented agent design.

3 The ARARA Framework

ARARA is a modular engineering framework for developing multi-agent CRSs powered by LLMs. Rather than prescribing a fixed conversational pipeline, ARARA models conversational recommendation as the coordinated interaction of explicitly defined architectural components. Conversational behavior emerges from structured role specialization, explicit orchestration, modular decomposition, and extensible capabilities operating under a unified execution model. Although ARARA exposes reusable multi-agent abstractions, it is designed to make the architectural structure of conversational recommender systems explicit, *e.g.*, modular decomposition, user modeling within the interaction loop, and controlled memory usage—whereas general-purpose agent frameworks provide coordination mechanisms but leave such CRS-specific design decisions implicit, often embedded in prompts or loosely defined pipelines.

Figure 1 presents the architectural flow. Execution begins when a user submits a query. The orchestrator selects the next executable component—either an agent or a module. The selected component processes the message, may invoke external tools or execute code, may access memory through skills, and produces structured output. The output updates the shared state and is routed back to the orchestrator. This message-driven loop continues until a termination condition is reached.

Agents. Agents are the fundamental reasoning units in ARARA. Each agent embodies a well-defined role—such as retrieval, normalization, verification, ranking, explanation, or evaluation—and transforms structured inputs into structured outputs consumed by other agents or modules. Agents operate under explicit orchestration control and serve as the primary units of reasoning within the system.

An agent's behavior is defined through two complementary mechanisms. First, its role is specified via a system-level instruction that constrains its reasoning scope and behavioral objectives. Second, the agent may invoke external functions or computational routines during execution. For instance, an agent may invoke a classical machine learning model (e.g., a random forest or regression model), execute a deep learning inference routine, or invoke a domain-specific function with structured parameters. The result of such computation can then be reformulated or contextualized by the agent through its LLM-based reasoning process. This design ensures that agents may combine prompt-based reasoning with deterministic or statistical computation within a unified abstraction.

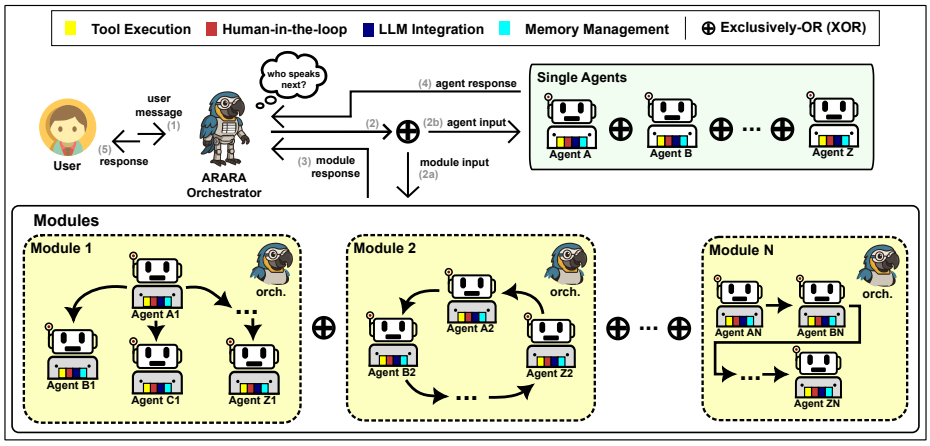

Fig. 1: Overview of the ARARA architecture. A user submits a query, which the orchestrator routes to agents or modules. Agents operate as LLM-driven reasoning units and may access memory, invoke tools, or execute code through skills. The message-driven loop continues under explicit orchestration control until termination.

ARARA adopts a provider-agnostic LLM client interface. Agents can be configured with different backend providers—including OpenAI, Anthropic, OpenRouter, Groq, or even locally hosted models—allowing heterogeneous model configurations within the same system. The client layer supports response caching and cost tracking, enabling reproducible experimentation and operational monitoring. New providers can be integrated by implementing additional connectors without modifying the architectural core.

Agents communicate exclusively through message passing. Each message preserves the sender and recipient identities, role, and content, ensuring traceability while maintaining compatibility with chat-based LLM interfaces.

User. Within ARARA, the `User` is modeled as a first-class agent that interacts through the same message-based interface as other components. Execution begins when the user formulates a query. This message is routed through the orchestrator and integrated into the same conversation state as agent-generated responses. Unlike autonomous agents, the `User` does not perform LLM-based generation. Modeling the user architecturally ensures uniform execution semantics and avoids special-case routing or memory handling.

Orchestrator. The orchestrator is modeled as a specialized agent that governs the execution flow. It selects which component—an individual agent or a module—acts next, regulates information routing, and determines termination conditions. Routing policies may rely on deterministic rules, stochastic strategies, LLM-guided decisions, or external decision models. Because the orchestrator follows the same abstraction as other agents, alternative coordination mechanisms can be implemented without altering other architectural layers. This enables experimentation with different dialogue planning strategies while preserving structural consistency.

Modules. Modules provide structural organization by grouping agents into problem-specific units. Each module encapsulates one or more agents designed to address a specific subtask of the conversational recommendation workflow. For example, a *Recommendation Module* may contain agents responsible for candidate retrieval, contextual filtering, and ranking, while a separate *Explanation Module* may include agents dedicated to justification generation, critique, or fidelity analysis. When a module is activated, only the agents within that container participate in the reasoning sequence. This decomposition allows developers to divide complex CRS workflows into well-defined structural units aligned with domain reasoning stages. Modules are flexible design constructs: agents may be reused across modules, and a single query may involve one or multiple modules, depending on the desired decomposition and coordination strategy. ARARA does not impose a fixed execution structure, leaving these design decisions to the developer. Modules are activated by the system-level orchestrator, which treats each module as a single executable unit. Once activated, control is delegated to the module-level orchestrator, which coordinates the agents contained within that module.

Skills. Skills encapsulate reusable capabilities that can be attached to agents without altering their reasoning roles. This abstraction separates infrastructural functionality from decision logic. Examples include **(i)** Vision, enabling multimodal image interpretation; **(ii)** Web Search, enabling external information retrieval; and **(iii)** Code Execution, enabling controlled execution of generated programs. Skills are plug-and-play and extensible, allowing new capabilities to be implemented and attached to one or multiple agents without structural refactoring.

Memory. Memory is implemented as a specialized skill. The framework supports **(i)** episodic memory, which records chronological interaction histories associated with individual agents; and **(ii)** shared memory, a global knowledge space accessible across agents with ownership-aware write semantics. Both memory types share a common interface that defines storage, retrieval, and context injection. Retrieval strategies and update mechanisms are customizable. Additional memory structures may be introduced without modifying agent logic or orchestration policies.

Runtime Execution and Extensibility. Execution begins when a user submits a query. The orchestrator selects the next agent or module to act. The selected component processes the message, optionally invoking functions, executing code, or accessing memory through skills, and produces structured output. This output updates the shared state and is returned to the orchestrator. The conversational loop terminates under two conditions: **(i)** when an agent signals completion of its reasoning through a configurable termination marker (e.g., "finish"), which must then be confirmed by the user; or **(ii)** when the user explicitly ends the session by issuing the command "exit".

ARARA exposes extensibility at multiple architectural layers. Developers may **(i)** define new agent roles or integrate additional LLM providers; **(ii)** implement alternative orchestration strategies; **(iii)** introduce custom memory structures and retrieval mechanisms; and **(iv)** design new skills encapsulating domain-specific capabilities. Because these extension points are isolated, systems can evolve without structural redesign. ARARA is implemented in Python and released as open source under the MIT License[11].

[11] https://github.com/fsant0s/arara/

4 Experimental Methodology

This section presents the procedures of an evaluation to assess how modeling conversational recommendation as a *multi-agent orchestration problem* affects behavior, robustness, and reasoning outcomes. Our goal is not to propose a new recommendation algorithm but to quantify the structural effects of agent decomposition, coordination policies, and controlled memory usage. We instantiate the ARARA architecture on a challenging CRS benchmark and compare it to single-LLM baselines under matched conditions, focusing on queries that require multi-hop inference and correction of incorrect premises.

Subsection 4.1 describes the evaluation datasets and query types. Subsection 4.2 defines the adopted metrics. Subsection 4.3 introduces the single-agent baselines and memory configurations. Subsection 4.4 details the ARARA agents, modules, and orchestration setup used in the experiments.

4.1 Datasets

We adopt the *Movie* and *Book* subsets of RecBench+ [7] as structured and challenging use cases for conversational recommendation. These datasets are designed to stress reasoning, factual consistency, and contextual understanding—properties that are difficult to address with single-agent or purely prompt-based approaches. Each subset extends a traditional collaborative dataset—MovieLens-1M for movies and Amazon-Book for books—by integrating structured knowledge from Wikipedia and Amazon metadata. The resulting Item Knowledge Graph (KG) links items to multiple attributes such as directors, actors, genres, and authors, enabling multi-relational and multi-hop reasoning. RecBench+ defines three condition-based query types that capture distinct reasoning challenges. *Explicit* queries directly specify entities or attributes (e.g., "movies directed by Christopher Nolan"), *Implicit* queries refer to examples that require relational inference (e.g., "movies directed by the director of *The Abyss*"), and *Misinformed* queries intentionally contain factual inaccuracies to simulate user misconceptions (e.g., "books by George Orwell who wrote *Brave New World*"). Each query type is associated with a ground-truth item set extracted from the Item KG, enabling reproducible and semantically grounded evaluation. Following [7], we focus exclusively on these three query types and exclude profile-based queries to maintain controlled reasoning conditions. To balance computational feasibility and representativeness, we sampled 200 queries per reasoning condition across domains. This results in 600 queries per domain (Explicit, Implicit, and Misinformed) and 1,200 queries in total across both domains. These queries serve as a testbed to analyze how different reasoning structures—single-agent versus multi-agent—behave as reasoning complexity increases.

4.2 Evaluation Metrics

Five complementary metrics are adopted. Let R_{rec} denote the set of items recommended within the top-k positions, and R_{true} the ground-truth relevant item set extracted from the

Item KG. Recall (R) measures the proportion of relevant items successfully retrieved, capturing how well the system covers the ground-truth set. It is defined as

$$R@k = \frac{|R_{\text{rec}} \cap R_{\text{true}}|}{|R_{\text{true}}|}. \tag{1}$$

Precision (P) measures the concentration of relevance within the recommended list, reflecting how much of what is returned is actually correct. It is defined as

$$P@k = \frac{|R_{\text{rec}} \cap R_{\text{true}}|}{|R_{\text{rec}}|}. \tag{2}$$

Normalized Discounted Cumulative Gain (NDCG) evaluates ranking quality by assigning higher importance to relevant items appearing at top positions, thus capturing ordering sensitivity. It is computed as

$$NDCG@k = \frac{DCG@k}{IDCG@k}, \tag{3}$$

where $DCG@k$ denotes the discounted cumulative gain up to position k, and $IDCG@k$ corresponds to the ideal (maximum) DCG obtained when all relevant items are perfectly ranked at the top positions. Satisfied Ratio (SR) reflects user-level success by verifying whether at least one relevant item appears within the top-k results for a given query. It is defined as

$$SR = \frac{1}{N} \sum_{u=1}^{N} \mathbb{I}(|R^u_{\text{rec}} \cap R^u_{\text{true}}| > 0), \tag{4}$$

where N denotes the total number of evaluated queries. Finally, Fail-To-Recommend (FTR) measures robustness by quantifying the proportion of queries for which the system fails to return any valid recommendation. It is defined as

$$FTR = \frac{N_{\text{fail}}}{N}. \tag{5}$$

A failure is defined as a query for which the system returns no valid item in the top-k results. All ranking-based metrics are computed at a query-dependent cutoff k, where k corresponds to the number of ground-truth relevant items associated with the input query (bookCount/movieCount). In our benchmark, k varies between 2 and 5 across queries. Higher values of R, P, NDCG, and SR indicate stronger recommendation effectiveness and user-facing success, whereas lower FTR indicates greater robustness under increasing reasoning difficulty. All reported metric values correspond to macro-averages computed across the evaluated queries. All experiments were executed once per query without stochastic repetition.

4.3 Baselines

We adopt three representative LLMs from RecBench+ [7] as structural baselines: **GPT-4o**, **DeepSeek-V3**, and **Claude 3.5 Sonnet**. These models are not treated as competing algorithms, but as reference single-agent configurations against which ARARA-based

multi-agent instantiations are compared. Each model is evaluated under two settings: *with memory* (*WM*) and *without memory* (*W/O M*). In the memory-enabled setting, the model receives the user's ten most recent interactions (`history_line`) as context. In the memory-free setting, reasoning is based solely on the current query. This design allows us to isolate the effect of *agent orchestration and task decomposition* from the underlying language model capabilities.

4.4 Agents, Modules and Orchestration Setup

We use the *ARARA* architecture to evaluate how modeling conversational recommendation as a *multi-agent orchestration problem* affects behavior under challenging reasoning scenarios. For each underlying LLM, we compare: (i) a **single-agent baseline** (LLM + RecBench+ prompt) and (ii) a **multi-agent ARARA instantiation** embedding the same LLM within ARARA's agent abstraction, where role-specific system configurations define bounded reasoning responsibilities under orchestration control. Both systems are evaluated under matched memory settings (*WM* / *W/O M*), so differences isolate reasoning structure and coordination rather than model capacity or additional supervision.

Agents and Memory. All ARARA agents implement the framework's unified agent abstraction, are instantiated through role-specific system configurations that define their reasoning behavior, and share the same LLM client within each run. For each model (**GPT-4o**, **DeepSeek-V3**, **Claude 3.5 Sonnet**), we instantiate two ARARA variants: *ARARA-*$_{WM}$ and *ARARA-*$_{W/O\,M}$. To isolate memory effects at the decision stage, only *Recommender* agents access the user's ten most recent interactions (`history_line`) through the episodic memory skill; all other agents operate statelessly. Agents are explicitly specialized into bounded reasoning functions (e.g., retrieval, normalization, validation, correction, ranking), enabling complex queries to be decomposed into interpretable steps.

Modules. Agents are grouped into three modules aligned with RecBench+ query types: *Explicit*, *Implicit*, and *Misinformed*. Exactly one module is activated per query under a mutually exclusive (XOR) selection policy. **Explicit module** consists of a `RetrieverAgent` for deterministic candidate retrieval and a `RecommenderAgent` for ranking (optionally conditioned on `history_line`). **Implicit module** decomposes multi-hop inference into `TitleNormalizer`, `CommonAttributeExtractor`, `RelationTypeDetector`, `MultiHopRetriever`, and `RecommenderAgent`. **Misinformed module** introduces explicit validation and correction via `EntityResolverAgent`, `FactCheckerAgent`, `FactCorrectionAgent`, `RetrieverAgent`, and `RecommenderAgent`, preventing incorrect premises from propagating into retrieval and ranking.

Orchestration. We use an **LLM-guided routing** strategy in which a prompt-based decision step selects one module (Explicit/Implicit/Misinformed) per query based on the module descriptions. Upon completion of the selected module, the orchestrator returns the final ranked recommendations to the user (Figure 2).

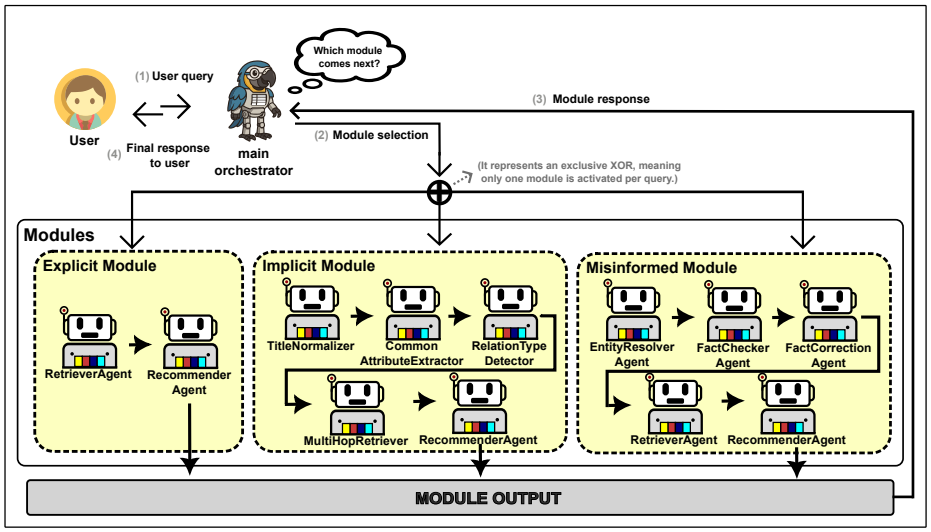

Fig. 2: Overview of the instantiated ARARA-based architecture. A prompt-based, LLM-guided orchestrator routes each query to exactly one module (Explicit, Implicit, or Misinformed), which coordinates specialized agents for structured reasoning.

This setup is designed to attribute observed gains to *structural properties*—role specialization, governed coordination, and controlled memory—rather than algorithmic novelty. In the present experimental setup, agent-level termination signals were not enabled. Execution proceeds until the user explicitly issues the command `exit`, which terminates the conversational loop and concludes the experimental run. All LLM APIs were executed with a temperature of 0.0 to eliminate stochastic variability, and identical decoding configurations were used across all experimental conditions. All agent definitions, system prompts, orchestration configurations, and experimental settings are publicly available on GitHub for full reproducibility[12].

5 Experimental Results

This section evaluated ARARA-based multi-agent instantiations from a structural perspective, focusing on how reasoning decomposition and orchestration affected conversational recommendation behavior. Rather than targeting absolute performance gains, we examine how multi-agent coordination affects retrieval quality, ranking quality, robustness to misinformation, and user-facing satisfaction as reasoning complexity increases.

The evaluation was organized into three analyses. First, Subsection 5.1 reports results for the *Movie* domain. Second, Subsection 5.2 presents results for the *Book* domain using the same protocol. Together, these two subsections compared single-agent

[12] `https://github.com/fsant0s/arara_experiments/tree/exp/pilot/pilot`

LLM baselines against their ARARA-based multi-agent counterparts across three reasoning conditions (*Explicit*, *Implicit*, and *Misinformed*). Third, Subsection 5.3 analyzed orchestration behavior itself, examining routing accuracy and module selection patterns across both domains. Recall that, for each underlying LLM baseline, two configurations were evaluated: *with memory* (*WM*) and *without memory* (*W/O M*). ARARA instantiations embedded the same LLM backends within the previously described modular architecture (*Explicit*, *Implicit*, and *Misinformed* modules coordinated by an orchestrator), preserving identical memory settings. Consequently, observed differences were interpreted as structural effects induced by agent specialization, governed coordination, and controlled memory usage, rather than by changes in model capacity.

5.1 Movie Domain Results

Table 2 reports the results for the *Movie* domain across three reasoning conditions: *Explicit* (Easy), *Implicit* (Medium), and *Misinformed* (Hard). Results were shown for base LLM configurations and their ARARA-based counterparts, under both memory settings.

In the *Explicit* condition, base models and ARARA configurations exhibited comparable behavior, with modest gains in recall and user satisfaction. For example, GPT-4o$_{(WM)}$ obtained R = 0.36 and SR = 0.72, while ARARA GPT-4o$_{(WM)}$ reached R = 0.40 and SR = 0.82. Comparable trends were observed across DeepSeek-V3 and Claude-3.5-Sonnet families, indicating limited separation between single-agent and multi-agent settings for well-specified queries. In the *Implicit* condition, ARARA configurations consistently yielded higher user satisfaction and precision. Under GPT-4o$_{(WM)}$, SR increased from 0.33 to 0.72, while precision increased from 0.16 to 0.28. For DeepSeek-V3$_{(WM)}$, ARARA maintained comparable recall while improving SR from 0.65 to 0.69. Similar patterns were observed across models and memory settings. The *Misinformed* condition exhibited the largest differences. Single-agent baselines showed higher failure rates, whereas ARARA configurations reduced FTR and increased SR. For instance, GPT-4o$_{(W/O\ M)}$ recorded FTR = 0.27, compared to 0.16 for ARARA GPT-4o$_{(W/O\ M)}$, while SR increased from 0.10 to 0.35. For Claude-3.5-Sonnet$_{(W/O\ M)}$, ARARA increased SR from 0.12 to 0.28 while maintaining comparable robustness levels.

Across several configurations, ARARA exhibited trade-offs between recall and satisfaction. For example, ARARA DeepSeek-V3$_{(W/O\ M)}$ achieved lower recall (R = 0.34 vs. 0.40) but higher satisfaction (SR = 0.87 vs. 0.62).

5.2 Book Domain Results

Table 3 presents results for the *Book* domain using the same evaluation protocol and reasoning conditions. In the *Explicit* condition, ARARA and base models exhibited different recall profiles. While base models achieved lower recall, ARARA configurations substantially increased recall, often at the cost of higher failure rates. For example, GPT-4o$_{(WM)}$ achieved R = 0.14, while ARARA GPT-4o$_{(WM)}$ reached R = 0.62, with a higher FTR. Similar recall–failure trade-offs were observed across other models. In the *Implicit* condition, ARARA configurations consistently improved ranking and satisfaction metrics. ARARA DeepSeek-V3$_{(WM)}$ increased NDCG from 0.20 to 0.50 and

Table 2: Quantitative results for the *Movie* domain across the three reasoning conditions — *Explicit* (Easy), *Implicit* (Medium), and *Misinformed* (Hard). Each block reports five metrics: FTR ↓, *R* ↑, *P* ↑, NDCG ↑, and SR ↑. Results compare base LLMs and their ARARA-enhanced counterparts, under both memory configurations: $_{(\text{WM})}$ for *with memory* and $_{(\text{W/O M})}$ for *without memory*. Bold values indicate the best result per condition.

	Explicit (Easy)					**Implicit (Medium)**					**Misinformed (Hard)**				
Model	FTR↓	R↑	P↑	NDCG↑	SR↑	FTR↓	R↑	P↑	NDCG↑	SR↑	FTR↓	R↑	P↑	NDCG↑	SR↑
GPT-4o$_{WM}$	**.03**	.36	.33	.36	.72	**.01**	.23	.16	.21	.33	.26	**.05**	**.03**	**.03**	.22
+ ARARA	.09	**.40**	**.43**	**.43**	**.82**	.15	**.29**	**.28**	**.28**	**.72**	**.18**	.04	**.03**	**.03**	**.24**
GPT-4o$_{W/O}$	**.01**	**.40**	**.30**	**.39**	.71	**.02**	.22	.14	.20	.30	.27	.03	.01	.02	.10
+ ARARA	.07	.26	.28	.27	**.85**	.16	**.28**	**.28**	**.28**	**.71**	**.16**	**.06**	**.06**	**.06**	**.35**
DeepSeek-V3$_{WM}$	**.00**	.44	.22	.39	.65	**.00**	**.26**	.09	.20	.65	**.00**	.06	.03	.05	.25
+ ARARA	.02	**.45**	**.46**	**.48**	**.86**	.12	**.26**	**.26**	**.28**	**.69**	.15	**.09**	**.08**	**.09**	**.26**
DeepSeek-V3$_{W/O}$	**.00**	**.40**	.18	**.35**	.62	**.00**	.26	.09	.20	.21	**.01**	**.07**	.02	.05	.10
+ ARARA	.02	.34	**.35**	**.35**	**.87**	.15	**.28**	**.28**	**.28**	**.69**	.07	**.07**	**.07**	**.07**	**.27**
Claude-3.5$_{WM}$	**.01**	.36	.20	.31	.65	**.01**	.16	.10	.12	.09	**.02**	.04	.06	.03	.21
+ ARARA	.06	**.42**	**.42**	**.45**	**.87**	.13	**.23**	**.23**	**.24**	**.59**	.05	**.08**	**.08**	**.09**	**.23**
Claude-3.5$_{W/O}$	**.00**	.31	.26	**.44**	.65	**.00**	**.21**	.12	**.23**	.24	.08	.07	.03	.05	.12
+ ARARA	.02	**.38**	**.38**	.39	**.79**	.09	.17	**.17**	.18	**.61**	**.04**	**.08**	**.08**	**.07**	**.28**

SR from 0.04 to 0.09. Comparable improvements were observed across models and memory settings. The *Misinformed* condition again showed larger separations. ARARA Claude-3.5-Sonnet$_{(\text{W/O M})}$ achieved R = 0.49 and SR = 0.15, compared to R = 0.05 and SR = 0.09 for the corresponding baseline. In several cases, ARARA configurations traded higher recall and user satisfaction for increased failure rates, particularly under misinformed queries.

5.3 Orchestration Analysis Results

To better interpret the differences observed in the results from the previous sections, we carried out an orchestration analysis to isolate the effect of the orchestrator's routing decisions. In ARARA, downstream module specialization could only improve performance if the query was first assigned to the appropriate reasoning module. Therefore, evaluating routing accuracy allowed us to disentangle whether observed gains or failures stem from module specialization itself or from errors in module selection. Table 4 reports routing accuracy for representative ARARA configurations across the *Movie* and *Book* domains. One representative per model family was analyzed, selected based on robustness under *Misinformed* queries, factual reliability (low FTR), and user satisfaction (high SR), to avoid redundant analysis across highly similar configurations. In the *Movie* domain (Table 4a), the orchestrator achieved consistently high accuracy for *Explicit* and *Implicit* queries across all representatives. Most errors were concentrated in the *Misinformed* condition, where deceptive inputs were occasionally misclassified as simpler reasoning types, which suggested increased ambiguity in decep-

Table 3: Quantitative results for the *Book* domain across the three reasoning conditions — *Explicit* (Easy), *Implicit* (Medium), and *Misinformed* (Hard). Each block reports five metrics: FTR ↓, R ↑, P ↑, NDCG ↑, and SR ↑. Results compare base LLMs and their ARARA-enhanced counterparts, under both memory configurations: $_{(WM)}$ for *with memory* and $_{(W/O\ M)}$ for *without memory*. Bold values indicate the best result per condition.

	Explicit (Easy)					**Implicit (Medium)**					**Misinformed (Hard)**				
Model	FTR↓	R↑	P↑	NDCG↑	SR↑	FTR↓	R↑	P↑	NDCG↑	SR↑	FTR↓	R↑	P↑	NDCG↑	SR↑
GPT-4o$_{WM}$	**.01**	.14	.07	.13	.06	**.01**	.12	.07	.11	.01	**.26**	.05	.03	.05	.00
+ ARARA	.15	**.62**	**.55**	**.62**	**.10**	.27	**.45**	**.42**	**.44**	**.06**	.28	**.42**	**.37**	**.41**	**.02**
GPT-4o$_{W/O}$	**.02**	.22	.09	.19	.05	**.01**	.14	.07	.13	.04	.39	.03	.02	.03	**.00**
+ ARARA	.12	**.56**	**.50**	**.54**	**.20**	.21	**.43**	**.40**	**.42**	**.15**	**.22**	**.34**	**.31**	**.32**	**.00**
DeepSeek-V3$_{WM}$	**.01**	.17	.06	.14	.05	**.02**	.18	.07	.20	.04	.10	.01	.01	.01	.00
+ ARARA	**.01**	**.57**	**.42**	**.63**	**.13**	.03	**.46**	**.32**	**.50**	**.09**	**.03**	**.28**	**.17**	**.30**	**.06**
DeepSeek-V3$_{W/O}$	**.00**	.26	.08	.21	.00	.04	.21	.07	.16	.00	**.08**	.00	.00	.00	.00
+ ARARA	.01	**.56**	**.43**	**.60**	**.27**	**.01**	**.37**	**.25**	**.39**	**.15**	.12	**.30**	**.21**	**.33**	**.07**
Claude-3.5$_{WM}$	**.00**	.16	.05	.15	.03	**.01**	.22	.05	.18	.02	**.04**	.09	.03	.06	.04
+ ARARA	.21	**.60**	**.52**	**.61**	**.16**	.03	**.41**	**.31**	**.43**	**.08**	.12	**.42**	**.33**	**.43**	**.12**
Claude-3.5$_{W/O}$	**.01**	.18	.06	.16	**.02**	**.01**	.18	.05	.16	.00	**.01**	.05	.02	.04	.09
+ ARARA	.07	**.34**	**.29**	**.33**	.00	.08	**.40**	**.33**	**.41**	**.08**	.22	**.49**	**.42**	**.48**	**.15**

tive queries. In the *Book* domain (Table 4b), routing remained comparatively reliable for *Explicit* queries but showed increased overlap between *Implicit* and *Misinformed* reasoning. This reflected higher linguistic ambiguity and weaker intent signals in the domain, which led to a tendency to overestimate reasoning complexity in challenging cases.

Table 4: Orchestration accuracy across Movie (a) and Book (b) domains, reporting inferred reasoning conditions (Explicit, Implicit, Misinformed), with 200 queries per condition and model.

(a) Movie domain

Model	**Condition**	**Exp.**	**Imp.**	**Mis.**
ARARA GPT-4o$_{(W/O\ M)}$	**Exp.**	197	0	4
	Imp.	2	200	15
	Mis.	1	0	181
ARARA DeepSeek-V3$_{(W/O\ M)}$	**Exp.**	200	0	126
	Imp.	0	200	21
	Mis.	0	0	53
ARARA Claude-3.5-Sonnet$_{(W/O\ M)}$	**Exp.**	199	0	29
	Imp.	0	200	1
	Mis.	1	0	170

(b) Book domain

Model	**Condition**	**Exp.**	**Imp.**	**Mis.**
ARARA GPT-4o$_{(W/O\ M)}$	**Exp.**	198	30	1
	Imp.	0	117	0
	Mis.	2	57	199
ARARA DeepSeek-V3$_{(WM)}$	**Exp.**	200	130	73
	Imp.	0	69	0
	Mis.	0	1	127
ARARA Claude-3.5-Sonnet$_{(WM)}$	**Exp.**	199	14	10
	Imp.	0	180	0
	Mis.	1	6	190

6 Discussion

Domain-level effectiveness. Across domains, ARARA reshapes CRS behavior by distributing reasoning across coordinated validation and refinement stages. In the *Movie* domain, it consistently improves recall, precision, and ranking quality through structured multi-agent collaboration, while in the *Book* domain, configurations like **ARARA Claude-3.5-Sonnet(W/O M)** maintain high recall and ranking quality, demonstrating that modular orchestration preserves semantic coherence even in denser relational contexts. These observations suggest that ARARA's effectiveness stems less from model capacity and more from its structured architectural design, which comprises traceable, interdependent stages that support transparency, reliability, and adaptive behavior across domains.

Reasoning across difficulty levels. For *Explicit* queries, baseline models and ARARA configurations behaved similarly due to the deterministic nature of retrieval. However, under *Implicit* and *Misinformed* conditions, where ambiguity, relational inference, and correction are required, single-agent reasoning exhibits increased error propagation under complex conditions. ARARA mitigates this effect by distributing reasoning across specialized modules that explicitly verify, correct, and recontextualize information. As a result, higher task difficulty is transformed into structured collaboration rather than compounded error, demonstrating that coordinated reasoning scales more reliably than isolated generation as reasoning complexity increases.

Memory analysis. Across both domains, memory-enhanced configurations improved contextual recall and user satisfaction under *Implicit* and *Misinformed* conditions by enabling the reuse of verified context during ranking. At the same time, memory-free configurations retained comparable stability, indicating that orchestration and modular validation alone already sustain reliable reasoning. These findings suggest that memory primarily supports continuity—reducing redundant reasoning and strengthening referential grounding—while its marginal gains diminish when coordination mechanisms are well calibrated. This reinforces ARARA's design goal of treating memory as an optional, controlled component rather than a mandatory source of performance gains.

Orchestrator effectiveness. In the *Movie* domain, routing accuracy was generally high, with minor confusion between *Misinformed* and *Explicit* queries, while in the *Book* domain, ambiguity was concentrated in *Implicit* cases due to dense semantic relations. Overall, ARARA effectively governs reasoning through explicit routing and coordination, though prompt-based routing remains sensitive to linguistic overlap. Rather than signaling structural flaws, these behaviors highlight natural extension points for the orchestration layer, such as adaptive routing thresholds, without compromising interpretability.

Overall analysis. Taken together, the results demonstrated that ARARA reshapes CRS by structuring how reasoning unfolds rather than modifying the underlying language models. While maintaining comparable behavior in simpler queries, it improves stability and robustness as reasoning becomes more implicit or misinformed. These effects highlight orchestration—through role specialization, explicit routing, and controlled memory—as the framework's core contribution.

Limitations and outlook. Routing remains sensitive to linguistic ambiguity, and memory usage is currently static. Moreover, ARARA does not aim to optimize absolute

performance, but to expose the structural effects of architectural design. As an extensible framework, ARARA provides a foundation for advancing multi-agent reasoning, adaptive orchestration, and memory strategies in CRS.

7 Conclusion

As conversational recommendation increasingly relies on LLMs, architectural decisions are often embedded implicitly within prompts or monolithic pipelines, limiting transparency and systematic analysis. This study introduced **ARARA**, a modular framework that models conversational recommendation as an explicit multi-agent orchestration problem, treating agents, modules, memory, and routing as composable architectural components. Rather than proposing a new recommendation algorithm, ARARA provides a structured foundation for decomposing reasoning into specialized roles coordinated under governed execution. Empirical results across two domains showed that such architectural decomposition improves stability and robustness as reasoning complexity increases, particularly under *Implicit* and *Misinformed* queries. While ARARA demonstrates clear benefits in structuring conversational reasoning, it also introduces additional architectural complexity and design overhead, highlighting the need for future research on adaptive orchestration mechanisms and more efficient memory management strategies to ensure scalability and broader applicability. In light of these trade-offs, architecture—rather than model capacity—emerges as a central design dimension in CRSs. Future work includes comparisons with alternative multi-agent frameworks to evaluate ARARA along additional dimensions such as adaptability and extensibility.

Acknowledgments. This project was supported by the Ministry of Science, Technology, and Innovation of Brazil, with resources granted by the Federal Law 8.248 of October 23, 1991, under the PPI-Softex. The project was coordinated by Softex and published as Intelligent agents for mobile platforms based on Cognitive Architecture technology [01245.003479/2024-10].

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Crewai: A framework for orchestrating role-playing autonomous ai agents. https://github.com/joaomdmoura/crewAI (2024), version X.X. Accessed: 2025-10-22
2. Deng, Y., Liao, L., Lei, W., Yang, G.H., Lam, W., Chua, T.S.: Proactive conversational ai: A comprehensive survey of advancements and opportunities. ACM Transactions on Information Systems **43**(3), 1–45 (2025)
3. Du, Y., Li, S., Torralba, A., Tenenbaum, J.B., Mordatch, I.: Improving factuality and reasoning in language models through multiagent debate. In: Forty-first international conference on machine learning (2024)
4. Fang, J., Gao, S., Ren, P., Chen, X., Verberne, S., Ren, Z.: A multi-agent conversational recommender system. arXiv preprint arXiv:2402.01135 (2024)
5. Gao, C., Lei, W., He, X., de Rijke, M., Chua, T.S.: Advances and challenges in conversational recommender systems. ACM Transactions on Information Systems **39**(6), 1–47 (2021). https://doi.org/10.1145/3472290, https://arxiv.org/abs/2101.09459

6. Hong, S., Zhuge, M., Chen, J., Zheng, X., Cheng, Y., Wang, J., Zhang, C., Wang, Z., Yau, S.K.S., Lin, Z., et al.: Metagpt: Meta programming for a multi-agent collaborative framework. In: The twelfth international conference on learning representations (2023)
7. Huang, J., Wang, S., Ning, L.b., Fan, W., Wang, S., Yin, D., Li, Q.: Towards next-generation recommender systems: A benchmark for personalized recommendation assistant with llms. arXiv preprint arXiv:2503.09382 (2025)
8. LangChainAI: Langgraph: Compositional graph framework for llm-orchestrated workflows. https://github.com/langchain-ai/langgraph (2024)
9. Li, G., Hammoud, H., Itani, H., Khizbullin, D., Ghanem, B.: Camel: Communicative agents for" mind" exploration of large language model society. Advances in neural information processing systems **36**, 51991–52008 (2023)
10. Li, X., Wang, S., Zeng, S., Wu, Y., Yang, Y.: A survey on llm-based multi-agent systems: workflow, infrastructure, and challenges. Vicinagearth **1**(1), 9 (2024)
11. Lin, J., Dai, X., Xi, Y., Liu, W., Chen, B., Zhang, H., Liu, Y., Wu, C., Li, X., Zhu, C., et al.: How can recommender systems benefit from large language models: A survey. ACM Transactions on Information Systems **43**(2), 1–47 (2025)
12. Nakajima, Y.: Babyagi. https://github.com/yoheinakajima/babyagi (2023), accessed: 2026-02-19
13. OpenAI: Gpt-4 technical report. arXiv preprint arXiv:2303.08774 (2023)
14. Peng, Q., Liu, H., Huang, H., Yang, Q., Shao, M.: A survey on llm-powered agents for recommender systems. arXiv preprint arXiv:2502.10050 (2025)
15. Wang, Z., Yu, Y., Zheng, W., Ma, W., Zhang, M.: Macrec: A multi-agent collaboration framework for recommendation. In: Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 2760–2764 (2024)
16. Wu, J., et al.: Autogen: Enabling next-gen llm applications via multi-agent conversation. arXiv preprint arXiv:2308.08155 (2023)
17. Zhang, Y., Chen, X.: Explainable recommendation: A survey and new perspectives. Foundations and Trends in Information Retrieval **14**(1), 1–101 (2020). https://doi.org/10.1561/1500000066