Multi-Agent Retrieval-Augmented Generation (MA-RAG): Methodology

Author Name

I. Methodology

The proposed Multi-Agent Retrieval-Augmented Generation (MA-RAG) system decomposes the query answering task into a sequence of specialized components, enabling modular and interpretable reasoning. Unlike conventional RAG systems that perform one-shot retrieval followed by generation, MA-RAG employs multiple agents (e.g., a query planner, several retrievers/extractors, a consensus module, a generator, and an optional verifier) that collaborate to refine the query, retrieve evidence, and synthesize the answer step-by-step. In this section, we describe the full MA-RAG pipeline, define mathematical notation for each component, interpret the key formulas, and illustrate the process with a concrete example. We also summarize agent roles in a table, present a system architecture diagram, outline the algorithm in pseudocode, and analyze computational complexity versus a single-agent RAG baseline.

A. System Overview and Pipeline

Figure 1 illustrates the MA-RAG architecture. The user issues a query q. A *Planner* or *Query Agent* may first rewrite or disambiguate q to a refined query q'. Multiple *Retrieval/Extraction Agents* then encode the query into dense vectors and search a large knowledge base $D = \{d_1, \ldots, d_N\}$ of documents or passages. Each retriever returns a set of top-k relevant documents; for example, Retriever i returns

$$R_i(q') = \{d_{i1}, d_{i2}, \ldots, d_{ik}\} \tag{1}$$

by selecting the documents whose embeddings are most similar to the query. The union of retrieved documents, or their interleaved refinement via a consensus mechanism, forms the evidence context. Finally, a *Generator* (LLM) produces an answer a conditioned on the query and the retrieved context. An optional *Verifier* agent can check or re-score the answer against the evidence. This multi-stage process allows MA-RAG to handle ambiguous or multi-hop queries by breaking them into sub-questions and iteratively gathering evidence.

B. Component Definitions and Notation

We now formalize the components with mathematical notation:

- **Query and Corpus.** Let q denote the input query (a sequence of tokens). Let $D = \{d_j\}$ be a large corpus or knowledge base of documents or passages. We assume a document d_j consists of text which can be embedded.
- **Embedding Function.** A neural encoder (e.g., a sentence-transformer or LLM encoder) maps any text x (query or document) to a dense vector $E(x) \in \mathbb{R}^d$. This embedding captures semantic meaning. In practice, $E(\cdot)$ may be a pre-trained encoder such as Sentence-BERT or CLIP for multi-modal data.
- **Similarity and Retrieval.** We define a similarity score between a query and a document via the dot product (or cosine similarity) of their embeddings:

  $$s(q, d_j) \;=\; E(q) \cdot E(d_j). \tag{2}$$

  A retriever agent returns the top-k documents by ranking $s(q, d_j)$. Formally,

  $$R(q) \;=\; \mathrm{TopK}\big(\{s(q, d_j) \mid d_j \in D\}\big). \tag{3}$$

  Typical systems use sparse (e.g., TF-IDF) or dense (neural) retrieval. Here we focus on dense embedding-based retrieval.
- **Multi-Agent Retrieval.** In MA-RAG, multiple retrieval agents $\{A_1, \ldots, A_m\}$ may run in parallel or sequence, possibly with different specializations (e.g. different indexes, query reformulations, or sub-queries). Each agent A_i yields a set of documents $R_i(q)$. The combined evidence set may be the union $R_\cup(q) = \bigcup_i R_i(q)$, or be filtered by a consensus mechanism as described below.
- **Consensus Mechanism.** After individual agents retrieve or propose intermediate answers, a consensus function $\mathcal{C}$ aggregates these outputs into a unified result. For example, if each agent A_i generates a candidate answer a_i, the consensus agent produces a final answer

  $$a^* \;=\; \mathcal{C}(a_1, a_2, \ldots, a_m). \tag{4}$$

  A simple consensus could be majority voting or a confidence-weighted average, but more sophisticated protocols (e.g. iterative voting) can be used. In our framework, consensus ensures that if one agent finds a correct piece of evidence, it is propagated to others, while conflicting or low-confidence content is down-weighted or discarded.
- **Generator.** A generative language model G (often a pre-trained seq2seq LLM) synthesizes the final answer. Given the query q and the retrieved evidence (concatenated passages) as context $C = \{d_{i1}, \ldots, d_{ik}\}$, the generator outputs an answer

  $$a \;=\; G(q, C). \tag{5}$$

  In practice, the query and evidence are formatted into a prompt, and the LLM produces a autoregressively. The

generator may also take as input intermediate chain-of-thought reasoning from earlier agents to improve coherence.

- **Verifier (Optional).** A verifier agent V can assess the correctness or faithfulness of the generated answer. This can be another model or a heuristic that outputs a score $V(a, q)$, or a binary label of valid/invalid. If V indicates low confidence, the pipeline may trigger additional retrieval or refinement steps. Verifiers are akin to fact-checkers that cross-verify the answer against the source evidence.

Each variable in these formulas has an intuitive meaning: $E(\cdot)$ produces semantic vectors, $R(\cdot)$ yields the most relevant documents, and $G(\cdot)$ constructs fluent answers. By defining these components explicitly, we achieve a system where each stage (retrieval, consensus, generation, verification) can be studied and optimized independently.

C. MA-RAG Pipeline Steps

The end-to-end pipeline from query to answer proceeds as follows:

1) **Query Refinement:** The raw query q is optionally processed by a *Query Agent* (e.g. a planner or rephraser) to produce a disambiguated or decomposed query q'. For instance, an ambiguous query may be split into sub-questions.
2) **Parallel Retrieval:** Multiple *Retrieval Agents* encode q' and fetch relevant texts. Each agent A_i computes embedding $E(q')$ and retrieves top-k documents $R_i(q')$ from D. Agents may use different vector stores or search strategies to increase coverage.
3) **Evidence Aggregation/Consensus:** The retrieved sets $\{R_i\}$ are merged. This could be a simple union $R_\cup$, or a consensus step where overlapping content is identified and disagreements (e.g. contradictory facts) are resolved. For example, if one agent retrieves document facts favoring answer a_1 and another favoring a_2, the consensus function selects the most credible answer via voting or re-ranking.
4) **Answer Generation:** The *Generator Agent* takes the query (or sub-queries) and the consolidated evidence context and generates an answer $a = G(q, C)$. Chain-of-thought prompts from previous steps can guide the generator to incorporate intermediate findings. The answer is usually a natural language response, possibly with citations or structured format.
5) **Verification:** Finally, an optional *Verifier Agent* examines the generated answer against the evidence. For instance, it may check consistency with retrieved facts or detect hallucinations. If the answer passes verification, the system returns it; otherwise, additional retrieval or agent iterations may be triggered.

This structured pipeline allows MA-RAG to handle complex, multi-hop queries more robustly than a single-pass RAG. For example, if a query requires multiple facts from different documents, one agent can retrieve each fact, and a consensus mechanism ensures they are correctly assembled before final answer synthesis.

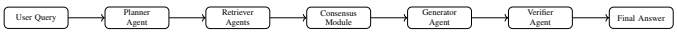

Fig. 1. Conceptual architecture of the MA-RAG system. A sequence of specialized agents processes the query to produce a final answer.

D. Example Scenario

Consider the query: *"Which country in Asia has the largest population, and what is its capital?"* A single LLM might guess or rely on outdated knowledge. In MA-RAG, the workflow could be:

- A **Planner Agent** confirms the query refers to Asian countries and decomposes it into sub-questions: (1) Identify the Asian country with the largest population; (2) Find the capital of that country.
- Two **Retriever Agents** operate: one looks up population statistics (retrieving e.g. a page or table), and another retrieves country profiles (e.g. an encyclopedia entry for "China", "India", etc.).
- The first agent returns candidate *China* (based on population data) and possibly *India*. The second confirms country capitals (e.g. Beijing for China, New Delhi for India).
- A **Consensus Agent** reconciles: China appears in both retrievals as the top candidate, while India appears only in population data. The consensus mechanism (e.g. voting or confidence scoring) selects China as the answer to (1).
- A **Generator Agent** then formulates the final answer using the evidence: "China has the largest population in Asia, and its capital is Beijing."
- A **Verifier Agent** checks that Beijing is indeed listed as the capital of China in the retrieved documents. Satisfied, it approves the answer.

This example shows how distinct agents contribute: one agent finds the populous country, another identifies capitals, and consensus/integration yields a correct, fact-backed answer. The intermediate chain-of-thought (e.g. "China: 1.4B; India: 1.3B; China capital - Beijing") is not shown to the user but is used internally for reasoning and can be logged for interpretability.

E. Agent Roles and Responsibilities

These roles are modular: depending on the query complexity, some may be bypassed or invoked multiple times. Modularity improves interpretability: one can inspect each agent's output (e.g. intermediate queries, retrieved texts, partial answers) to understand the reasoning path.

F. System Architecture (Conceptual Diagram)

The diagram captures the modular stages. In practice agents may operate in parallel or iteratively rather than strictly linearly, but the figure reflects the principal flow.

TABLE I
MA-RAG AGENTS AND THEIR ROLES.

Agent	Function
Query Planner / Refiner	Disambiguate or decompose the user query into subtasks, e.g. by rewriting or generating sub-questions.
Retriever Agent	Encode queries into embeddings and fetch relevant documents from the knowledge base.
Extractor Agent	(Optional) Extract specific facts or passages from retrieved documents (e.g. named entities, numbers).
Consensus Agent	Aggregate outputs of multiple agents (e.g. merge retrieved passages or answers) using voting or scoring to resolve conflicts.
Generator (QA) Agent	Produce the final answer text conditioned on the query and consolidated evidence.
Verifier Agent	Check the answer for factual consistency, detect hallucinations, or assign confidence scores.

Algorithm 1 Multi-Agent RAG Pipeline

Require: Query q, knowledge base D, agents $\mathcal{A}$

$q' \leftarrow$ Planner.process(q) {Refine or decompose the query}
for each Retriever A_i in Retrievers **do**
  $v_i \leftarrow E(q')$ {Compute embedding for the refined query}
  $R_i \leftarrow$ Retrieve(v_i, D) {Retrieve top-k docs}
end for
$C \leftarrow$ Consensus.aggregate$(\{R_i\})$ {Merge or re-rank retrieved evidence}
$a \leftarrow$ Generator.generate(q', C) {Generate answer conditioned on evidence}
if Verifier exists **then**
  if Verifier.score$(a, q', C) < \tau$ **then**
    {Low confidence: optionally refine and re-run}
    Go to step 1 or trigger additional retrieval
  end if
end if
return a

G. Pseudocode

Each procedure (Planner.process, Retrieve, Consensus.aggregate, Generator.generate, Verifier.score) may involve LLM prompts or learned models. For example, `Retrieve` could call a vector search index; `Consensus.aggregate` might perform voting; `Generator.generate` invokes an LLM with chain-of-thought prompts; and `Verifier.score` could be an LLM judging answer correctness. This pseudocode emphasizes interaction and order among agents.

H. Complexity Analysis and Comparison

The computational cost of MA-RAG can be analyzed in terms of retrieval and generation. Let N be the number of documents in the corpus and d the embedding dimension. A single dense retrieval (nearest-neighbor search) has worst-case cost $O(Nd)$, though indexing (e.g. FAISS) often gives sublinear performance in practice. In MA-RAG, if m retriever agents run independently, the cost is roughly m times that of a single retrieval (assuming disjoint searches), i.e. $O(mNd)$, plus any overhead for consensus. The generative step (LLM inference) is the same order as a standard RAG system. Verifier steps may add extra LLM calls or checks. In practice, many retrievals can be performed in parallel and some agents (like the planner) are relatively lightweight, so wall-clock latency may be only moderately higher than a single RAG pass.

Importantly, MA-RAG achieves significant gains in answer quality with this extra cost. Multi-agent RAG systems often outperform single-agent baselines on multi-hop or ambiguous queries: the added complexity is offset by modular parallelism and by avoiding expensive end-to-end fine-tuning. In summary, MA-RAG scales roughly linearly with number of agents in computation, but yields more robust, interpretable, and accurate answers than single-agent RAG by dividing labor and cross-verifying results.