Multi-Agent System for Reliable Citation Retrieval

Sathya Gnanakumar sgnanaku@umd.edu
Ishaan Kalra ishaank@umd.edu
Vishnu Sreekanth vishnus@umd.edu
Dhruv Suri dsuri@umd.edu
Vibhu Singh vibhu307@umd.edu
Kushal Kapoor kushalk@umd.edu

1 Abstract

Accurate citation is vital to upholding scientific integrity. However, the process of identifying and formatting references remains a significant bottleneck in the academic writing workflow. This project proposes the development of a **Citation Finder**, a multi-agent system that autonomously retrieves, aggregates, reranks, and selects scholarly references given a query or manuscript excerpt. Our approach combines sparse and dense retrieval (BM25, E5-Large, and SPECTER-2), combines results using Reciprocal Rank Fusion, applies reranking, and performs citation selection using a DSPy reasoning module. Evaluated on a subset of the ScholarCopilot benchmark, our full system achieves a Recall@10 of 19.7%, Recall@20 of 29.5%, and the highest Mean Reciprocal Rank (MRR) of 11.4%, outperforming individual retrievers at larger recalls. These results demonstrate that structured multi-agent retrieval combined with LLM-based selection improves citation selection while reducing the risk of hallucinated references.

2 Introduction

Trustworthy citations are at the heart of quality research papers. Citations serve as the bridge between past work and current research, demonstrating credibility and providing further resources for readers. However, the process of determining, verifying, and formatting citations is not so simple. When starting a paper, it can be a tedious process to find papers that are relevant to the topic at hand, and even when sources are found, it can be difficult for one to keep track of the reference that they are currently sourcing information from. Another issue is hallucinated references, as people have increasingly become reliant on large language models (LLMs), which often provide references to papers that do not exist or provide incorrect evidence.

Thus, a streamlined, accurate method of finding relevant research papers that goes beyond a straightforward keyword-based search would provide extreme utility to authors and researchers. As such, for our project, we seek to develop and train a multi-agent model system capable of retrieving and recommending relevant papers for citation purposes, given an input such as a research topic, query, or existing manuscript.

To evaluate the performance of our multi-agent system, we rigorously compared its performance against baseline BM25 search, dense retrieval models, and an LLM model with re-ranking. We used ScholarCopilot's large training dataset of computer science papers scraped from ArXiv as a training corpus to help guide our model in learning about writing patterns and best citation practices (Wang et al. 2025). As part of this study, we also leveraged the metrics presented in the following two studies: *LitSearch: A Retrieval Benchmark for Scientific Literature Search* by Ajith et al., 2024, and *CiteME: Can Language Models Accurately Cite Scientific Claims?* by Press et al., 2024. Our goal is to help authors quickly identify relevant prior work to cite in their papers, thereby reducing the risk of hallucinated citations. We hope to apply this system to various fields such as law and medicine.

3 Related Work

To better understand the performance of current state-of-the-art citation retrieval models and obtain a clearer direction for our project, we examined multiple papers, each approaching the issue of citation retrieval in different ways.

Ajith et al. introduce LitSearch, a comprehensive retrieval benchmark based on 597 curated literature-search queries, each paired with cited papers beforehand, to evaluate the performance of sparse versus dense retrieval models. It finds that dense, neural network-based retrieval models significantly outperform sparse retrieval models, namely BM25, by 24.8% in recall@5. Furthermore, the authors found that reranking with LLMs provided an additional +4.4 improvement over the best retriever. It also highlights that standard searches such as Google and Google Scholar lag significantly behind both sparse and dense retrieval models [1].

Press et al. present CiteME, which reframes the problem by asking whether humans, closed-source language models, and an autonomous retrieval-augmented agent can correctly recover a citation when it is masked from a scientific excerpt. It finds that currently, humans perform the best, achieving 69.7% accuracy, followed by CiteAgent, an autonomous model-based agent, achieving 35.5%, followed by closed language models at a mere 4.2 - 18.5% [9].

Wang et al. present ScholarCopilot, a framework which explores a different dimension of citation retrieval. ScholarCopilot, which is trained on 500K papers sourced from arXiv, generates a retrieval token which is matched against a citation database in order to determine whether a certain citation should be retrieved for LLM-generated text during the generation process. It yields a 40.1% top-1 retrieval accuracy, while achieving 100% preference in citation quality over ChatGPT. In the training dataset, in-text citations in the body of a paper are replaced by placeholder keys, which correspond to entries in a bibliography info field included alongside that paper [11].

Lalá et al. present PaperQA, a RAG agent that retrieves scientific papers not merely to provide citations, but to improve the quality of LLM-generated answers to a query. On LitQA, a benchmark comprising of 50 multiple choice questions, PaperQA achieves an accuracy of 69.5%, outperforming both expert humans (66.8%) and standalone LLMs/frameworks (in the range of 24% - 40.6%). PaperQA is also the only subject to achieve a hallucination rate of 0%, underscoring how RAG can enhance answer accuracy while minimizing the risk of potentially generating false information [7].

Ajith et al. and Press et al. provide strong evidence that searches, retrieval models, and language models still have a long way to go in terms of effective citation retrieval [1, 9]. However, CiteME reveals that an autonomous agent with searching and information retrieval capabilities beyond closed models has growing potential, being nearly twice as effective as the best closed language model [9]. Furthermore, through PaperQA's outstanding performance, Lalá et al. reveal the potential of a RAG agent in ensuring accuracy and eliminating hallucinations [7].

Cherian et al. present a multi-agent literature retrieval framework that combines sparse BM25 indexing with dense FAISS similarity search. Their architecture consists of a Query Agent for input refinement, a Retrieval Agent for hybrid search, and a Learning Agent that incorporates user feedback to improve personalization over time. The system reports modest improvements in precision (+8.5%), recall (+7.3%), and response latency relative to PaperQA and Semantic Scholar, and additionally measures hallucination rates for RAG-based answers. Their work demonstrates the value of multi-agent systems and hybrid retrieval alongside personalization features for improved literature search [2].

This motivates our project idea of building a multi-agent citation finder system that performs more structured literature retrieval and leverages multiple retrieval methods [1, 2, 7, 9]. Additionally, the strong performance of ScholarCopilot suggests that organizing training data with explicit citation placeholders linked to bibliographic entries provides a useful structure for citation prediction [11]. Building on these insights, we aim to design a multi-agent system that improves the speed and consistency of scientific literature search.

4 Methods

Our approach combines traditional information retrieval with a multi-agent architecture powered by LangGraph. The goal is to balance the reliability of various baseline keyword searches. Below we describe the various components of our approach, our baseline results, and the architecture of our pipeline.

4.1 Key Components

4.1.1 Sparse Retrieval

Sparse retrieval relies on exact keyword matching to score documents based on the frequency and distribution of query terms within the corpus. Due to its computational efficiency and low latency, it serves as a strong baseline for understanding retrieval performance over our citation corpus. In this work, we employ BM25s, a Python library implementing the BM25 ranking algorithm, to estimate document relevance with respect to a given search query. The resulting ranking function assigns higher scores to documents that are more likely to be relevant to the query, enabling effective initial retrieval. [8]

4.1.2 Dense Retrieval

We also leverage the power of dense retrieval in our approach, in order to go beyond a simple keyword-based comparison. The embeddings created by dense retrieval models attempt to capture the true essence of what a document actually means through many iterations of fine-tuning across a set of carefully configured layers, a dimension largely unexplored by sparse retrieval. Specifically, we use E5-Large and SPECTER-2. E5-Large serves as a strong, generalized semantic dense retriever that can effectively adapt to a large variety of contexts, in our case academic excerpts, and produce meaningful embeddings. [10] SPECTER-2 complements this by providing a retrieval approach tailored specifically to scientific papers, as it focuses on document-level relatedness through using citation graphs, ensuring that its embeddings better capture the full semantic scope of scientific documents. [3]

4.1.3 Aggregation and Reranking

We aggregate the outputs of multiple heterogeneous retrievers—specifically a sparse BM25 retriever and dense embedding-based retrievers (E5-Large and SPECTER-2)—to produce results that capture both lexical and semantic relevance. Rather than relying on raw retrieval scores, we employ *Reciprocal Rank Fusion* (RRF) as our aggregation method. RRF combines ranked lists by assigning each document a fused score based solely on its rank within each retriever:

$$RRF(d) = \sum_{i=1}^{N} \frac{1}{k + rank_i(d)}$$

where $rank_i(d)$ denotes the rank of document d in retriever i, and k is a smoothing constant that controls the influence of top-ranked results. Documents that appear consistently across multiple retrievers are rewarded with higher aggregate scores, while the method remains robust to score calibration issues and outliers.

Prior work has shown that RRF outperforms both Condorcet-based fusion and learned rank aggregation methods while remaining simple and computationally efficient, making it well-suited for multi-retriever systems [4]. In our implementation, we set $k = 60$, following common practice, and use RRF as the default aggregation mechanism. We additionally support a simple max-score aggregation baseline for debugging and ablation purposes.

After aggregation, we apply a reranking stage to refine the candidate list. This reranker leverages an LLM model to reassess relevance at a finer granularity, improving final citation quality prior to selection [5].

4.1.4 DSPy Prompt Design for Final Citation Retrieval

Core Idea To select the final cited paper, we leverage DSPy, a framework that enables modular LLM reasoning by organizing the text transformation process into structured, learnable components [6]. We use it as the **final citation picker**: given a citation context and a closed set of candidate papers (already retrieved and reranked), it selects exactly one paper to fill the missing `[CITATION]`. The core contribution is the prompt/program design: we use a DSPy signature to define a structured prompt with explicit input/output fields, and we serialize candidates into a constrained format so the model must choose from the provided set rather than inventing references.

Signature-Driven Prompt Construction Instead of hand-writing a single large prompt string, we define the citation selection task via a DSPy *signature*, which specifies:

- required inputs (citation context + candidate list),
- required outputs (selection + reasoning),
- task instructions embedded in the signature description.

Concretely, our signature `CitationRetrieval` takes:

- **`citation_context`**: excerpt containing `[CITATION]` where a reference was removed,
- **`candidate_papers`**: text-formatted list of candidate paper titles and abstracts,

and produces:

- **`reasoning`**: step-by-step analysis of which candidate matches the context,
- **`selected_title`**: the exact title of the selected paper.

We implement this using `dspy.ChainOfThought(CitationRetrieval)`, which forces the model to generate an explanation and an explicit selection.

Exact Prompt Template Used Below is the actual prompt template produced by our DSPy signature and candidate formatting. It is designed as a *closed-world* selection task: the model must select from the candidate list.

```
You are an expert citation retrieval system as described in the paper
"Multi-Agent System for Reliable Citation Retrieval".
Your goal is to autonomously retrieve, verify, and recommend academic
references given a query or document excerpt.
Task:
Given a citation context from a scientific paper (where a citation is missing),
identify the correct paper from a list of candidates.
Analyze the context to understand the specific claim, method, or result being cited.
Then, evaluate each candidate paper to see if it matches the context.
Finally, select the best matching paper.
Citation Context: ___,
Candidate Papers:
1. Title:
   Abstract:
2. Title:
   Abstract:
Think step-by-step about which candidate best matches the context.
Summarize your reasoning, then provide the exact title that should fill the citation.
```

Candidate Serialization (What the Prompt Sees) DSPy expects `candidate_papers` as text, so we convert the top-N reranked candidates into a numbered list. Each candidate includes a title and a truncated abstract to control prompt length. This structure makes it easy for the model to compare candidates and helps reduce hallucinations by grounding the decision in a fixed list.

Output Contract and Grounding The DSPy output is constrained to:

- **a natural-language rationale** (`reasoning`),
- **a single exact selection** (`selected_title`).

The system then maps `selected_title` back to an actual candidate paper object via exact title matching. If the predicted title does not match any candidate, the system falls back to the top reranked candidate to ensure it always outputs a valid retrieved paper.

Continual Prompt Updating with DSPy (Prompt/Program Optimization) A key reason we use DSPy is that it provides an explicit mechanism to **continually improve** the final-picker prompt/program as new supervision becomes available. Importantly, this continual improvement does *not* update the underlying LLM weights; instead, DSPy optimizes the **instructions and demonstrations** used at inference time (i.e., the prompt/program).

Training signal. From ScholarCopilot-style data, we construct labeled examples of the form: (citation context, ground-truth cited paper title) along with a candidate pool containing the positive paper and sampled negative papers (titles + abstracts). Candidates are shuffled so the positive is not always in a fixed position.

Optimization objective. We optimize top-1 citation selection by maximizing an exact-match metric on the predicted `selected_title` (case-insensitive exact match against the ground-truth title). This directly measures whether the picker chooses the correct citation from the candidate set.

Prompt compilation loop. We treat the DSPy picker as a continuously improving component via periodic recompilation:

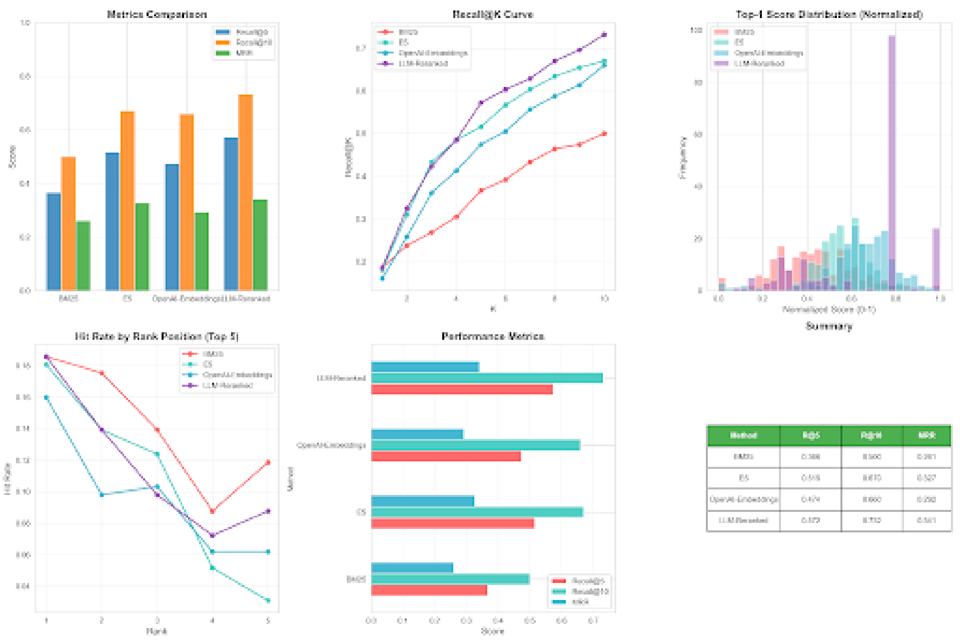
1. Collect new labeled citation examples (contexts with verified ground-truth citations).
2. Add them to the training set and regenerate candidate pools with hard negatives.
3. Run DSPy compilation (e.g., `BootstrapFewShot` or `MIPROv2`) to optimize the citation-selection program for the metric above.
4. Save the compiled program to disk (serialized) and deploy it for inference.

At inference time, the system always uses the latest compiled DSPy program, so as the dataset grows, the selection prompt/program is continually updated without changing the retrieval pipeline or retraining the base LLM.

4.2 Baselines

We first establish baselines using our chosen retrieval methods. For the sparse baseline, we run BM25, and for the dense baselines, we use E5-Large, SPECTER-2, and OpenAI embeddings coupled with LLM reranking. The datasets that we use for these baselines are the ScholarCopilot evaluation dataset, which contains 1000 academic papers in the same format as the ScholarCopilot training dataset, and a 200-example citation evaluation set. We evaluate performance using recall@5. We also collected recall@10 for the baselines using the latter dataset.

- **BM25:** BM25 provides a strong baseline despite having zero training and no deeper understanding of surrounding contexts. This was run with a reference corpus built from all cited papers found in the 1000 papers of the ScholarCopilot evaluation dataset. The queries included all sentences with removed in-text citations and their surrounding contexts. With this training setup, a recall@5 score of 29.2% is achieved. On the 200-example evaluation set, BM25 achieves a recall@5 of 36.6%, and a recall@10 of 50%.



Method	R@5	R@10	MRR
BM25	0.368	0.500	0.261
E5	0.515	0.670	0.327
OpenAI-Embeddings	0.474	0.660	0.292
LLM-Reranked	0.572	0.732	0.341

Figure 1: Baseline Results for Retrieval Models

- **E5-Large:** When similarly run with the corpus built from the 1000-paper evaluation dataset and excerpts with removed citations as queries, E5 performs poorly, with a recall@5 of 19.3%. However, on the 200-example evaluation set, it achieves a recall@5 of 51.5%, and a recall@10 of 67%.

- **SPECTER-2:** With the same 1000-paper evaluation dataset paradigm, SPECTER-2 performs very poorly, with a recall@5 of 9.7%. Due to compute constraints that we faced early on, we were not able to run it with the 200-example evaluation set.

- **OpenAI Embeddings:** Running OpenAI embeddings with the 200-example evaluation set only, we observe a recall@5 of 47.4% and recall@10 of 66%, a performance on par with E5-Large on this same 200-example set.

- **OpenAI Embeddings with LLM Reranking:** Due to further compute constraints, we only implement LLM reranking on top of the OpenAI embeddings, again on the 200-example evaluation set. We find a significant increase in performance, with a recall@5 of 57.2% and a recall@10 of 73.2%.

These baselines are essential for benchmarking our multi-agent system and provide fallback options for difficult queries.

4.3 Multi-Agent System Design

At the core of our system is a self-evolving citation retrieval pipeline that combines multiple specialized agents with DSPy-based automatic prompt optimization. The architecture orchestrates parallel retrieval, intelligent aggregation, LLM-based reranking, and a continuous optimization loop for self-improvement.

The system consists of the following specialized agents:

- **Fuzzy Logic Query Reformulator:** Analyzes user queries or manuscript excerpts and applies fuzzy logic reasoning to expand and refine search terms. Generates semantically enriched query variants optimized for multiple retrieval backends. Handles ambiguous or incomplete queries by inferring context.
- **Parallel Retrieval Agents:** Three independent agents execute simultaneously:
  - *BM25 Agent* – Classical probabilistic retrieval excelling at exact keyword matching and term frequency analysis
  - *E5 Agent* – Dense retrieval using transformer-based embeddings for semantic similarity matching
  - *SPECTER Agent* – Scientific paper embeddings trained on citation graphs for topical and methodological relevance
- **Aggregator Agent (RRF):** Merges results from all retrieval agents using Reciprocal Rank Fusion. Computes unified rankings without requiring score normalization, boosting papers appearing in multiple retriever results.
- **LLM Reranking Agent:** Refines aggregated rankings using large language model reasoning. Analyzes paper titles, abstracts, and metadata in context to apply deep semantic understanding. This is the primary target for DSPy prompt optimization.
- **Evaluator Agent:** Assesses retrieval quality using multiple metrics:
  - Fuzzy title matching against gold labels
  - Recall at k (R@5, R@10, R@20)
  - Mean Reciprocal Rank (MRR)
  - Weighted average score to determine optimization need

  Triggers optimization if weighted score falls below threshold (e.g., 0.75).
- **DSPy Optimizer (dspy_picker):** Core self-evolution component that automatically improves LLM reranking prompts. Collects training data (positive gold labels and negative highly-ranked irrelevant papers), applies DSPy optimizers (BootstrapFewShot, MIPRO, MIPROv2) to search prompt space, and

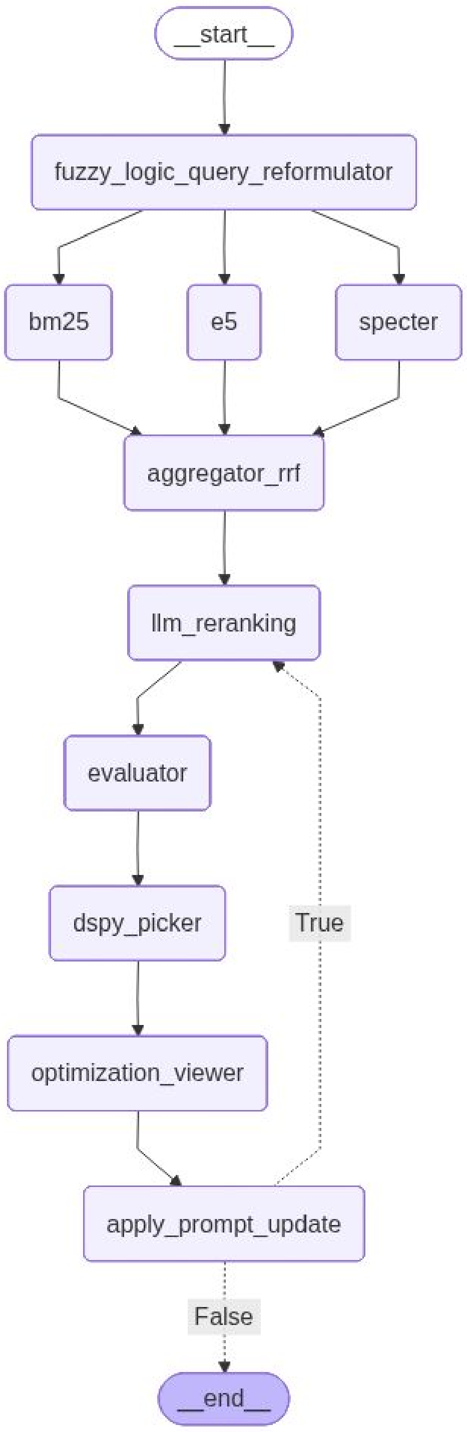


Figure 2: Multi-Agent Pipeline Architecture with Enhanced Retrieval

selects prompts maximizing retrieval metrics. Currently uses GPT-4o-mini as the meta-optimizer.

- **Optimization Viewer:** Logs and visualizes optimization process, displaying before/after prompt comparisons, metric improvements (ΔR@5, ΔR@10, ΔMRR), and training examples used.
- **Prompt Update Agent:** Applies optimized prompts to the LLM reranking node and manages iteration counters for the optimization loop.

4.4 Pipeline Flow

The system executes through the following stages:

1. **Query Reformulation:** Input query received by fuzzy logic reformulator, which generates enriched query variants.
2. **Parallel Retrieval:** Three retrieval agents (BM25, E5, SPECTER) execute simultaneously, each producing independent ranked lists of candidate papers.
3. **Aggregation:** Reciprocal Rank Fusion (RRF) merges all retrieval results into a unified ranking, weighting papers by their positions across retrievers.
4. **LLM Reranking:** Large language model analyzes top-k aggregated papers with deep semantic reasoning, generating a reranked list with contextual relevance assessment.
5. **Evaluation:** Evaluator computes retrieval metrics (R@5, R@10, R@20, MRR) and fuzzy title matching against gold labels. Weighted average determines if $score < threshold$ to trigger optimization.
6. **DSPy Optimization Loop:** If optimization is triggered:
   [label=(f)]
   (a) *DSPy Picker:* Collects positive/negative training examples, runs DSPy optimizer to search prompt space (different instructions, reasoning strategies, few-shot examples), selects prompt variant with highest metric performance
   (b) *Prompt Update:* Applies new prompt to LLM reranking node, increments optimization step counter
   (c) *Conditional Routing:* System evaluates continuation criteria:
      - If $needs\_optimization = False$: Exit to END (quality threshold met)
      - If $opt\_steps \geq max\_opt\_steps$: Exit to END (iteration limit reached)

    - Otherwise: Loop back to LLM Reranking for re-evaluation with new prompt

7. **Termination:** Pipeline completes when evaluation metrics meet threshold or maximum optimization iterations reached.

4.5 DSPy Self-Evolution Mechanism

The system implements a continuous improvement loop using DSPy for automatic prompt optimization. Training data is collected from query results: positives (gold label citations), negatives (highly-ranked irrelevant papers), and hard negatives (papers failing evaluation metrics). DSPy optimizers (BootstrapFewShot, MIPROv2, GEPA) experiment with different instruction phrasings, reasoning strategies, and few-shot examples, selecting prompts that maximize retrieval metrics (R@k, MRR). The optimization loop iterates until quality thresholds are met or maximum steps reached, creating a self-improving system where prompts evolve based on retrieval outcomes.

Current Status: The DSPy self-evolution pipeline is the primary goal and actively under development. Currently, the optimizer is being tested separately using GPT-5.2-mini to generate optimized prompts offline. The main objective is to fully integrate DSPy within the LangGraph workflow for real-time, on-the-fly prompt optimization and continuous learning from retrieval outcomes.

5 Deliverables

We have completed the following items:

- **Codebase:** Sparse retrieval implementation, dense retrieval model implementation, autonomous model-based agents, citation verification, citation formatting, unit tests, and a web demo interface.
- **Data:** A processed corpus of ScholarCopilot Data
- **Documentation:** A usage guide provided in our README with specific instructions on how to run our pipeline and various flags to test specific agents, change batch size, number of queries, and recall
- **Visualizations:** Multi-Agent Pipeline Architecture Diagram and evaluation plots (recall graphs)

6 Unit Tests

In order to ensure that our system works as a whole, we run unit tests on each individual agent in the pipeline. Our unit tests primarily look for proper formatting of the output from each agent. We do not validate correctness of the output values or evaluate them; rather, we ensure that the format which the agent produces can be used by the next agent in the pipeline.

For all agents in the pipeline, we wrote unit tests to ensure that the data returned was of the proper type. For most of the agents, this means a dictionary containing values used by the next agent in the query: for example, we check that the retrieval agents return a document ID, score for that document, and the source. To ensure that inputs were controlled throughout each unit test, we created mock results that might have been returned dby prior agents in the pipeline, such as a fake query and query expansion to provide to the retrieval agents.

For each unit in our system (that is, each agent in the pipeline), we created the following unit tests:

- **Query Reformulator:**

  Keyword extraction (extracts only words with more than 3 characters, converts all keywords to lowercase, handles empty query), **Keyword expansion** (expands known keywords, does not expand unknown keywords, handles empty keywords list), **Academic style rewriting** (produces formatted string, includes first three expansions), **Query reformulation** (returns dictionary, includes query, queries, and messages keys, handles no human message, uses last human message, handles empty keywords list, expanded queries include expansion, no whitespace in query)

- **BM25:**

  Get queries (returns queries when provided as a list, filters out empty strings from queries list, returns empty list when no queries provided), **No resources** (returns error message when no queries provided, resources missing), **Mocked resources** (returns dictionary, result contains id, title, score, source, returns correct number of results, passes k value from config to retriever)

- **E5:**

  Get queries (returns queries when provided as a list, filters out empty strings from queries list, returns empty list when no queries provided), **No resources** (returns error message when no queries provided, resources missing), **E5 retriever** (single query returns a list of results, each result contains id, title, source, and score, batch query returns a list of result lists) **Mocked resources** (returns dict, each result has id, title, score, source, handles multiple queries, respects k values from config)

- **SPECTER-2:**

  Get queries (returns queries when provided as a list, filters out empty strings from queries list, returns empty list when no queries provided), **No resources** (returns error message when no queries provided, resources missing), **SPECTER-2 Retriever** (single query returns a list of results, each result contains id, title, score, source, batch query returns a list of result lists), **Mocked resources** (returns dict, each result has id, title, score, source, handles multiple queries, respects k values from config)

- **Aggregator:**

  Normalize scores (returns empty list for empty input, normalizes scores, preserves original scores separately, handles single result, handles identical scores, applies correct normalization formula, preserves all other fields), **Reciprocal rank fusion** (works with results from a single retriever, papers appearing in multiple retrievers receive higher scores, includes RRF score, includes retriever count, includes sources list, uses custom k value in RRF formula, preserves paper metadata), **Aggregator with no results** (returns empty candidate_papers when all retrievers return empty, treats missing result keys as empty lists, treats None values as empty lists, returns appropriate AIMessage for no results), **Aggregator RRF method** (RRF is default aggregation method, removes duplicate papers, papers in multiple retrievers rank higher, stores raw results from each retriever, adds rank information to each result, uses custom rrf_k value from config, returns AIMessage with aggregation summary), **Aggregator simple method** (uses simple method when specified, keeps highest normalized score per paper, sorts by score descending), **Aggregator integration** (correctly aggregates results from all three retrievers, handles larger result sets efficiently)

- **LLM Reranker:**

  Empty candidates (returns empty list when candidate_papers is empty, returns empty when candidate_papers doesn't exist) Successful parsing (correctly parses JSON array and rank papers, extracts JSON array even hen embedded in other text, papers not ranked by LLM should score 0, skips duplicate paper indices, skips out-of-bounds indices), **Reranker error handling** (falls back to original order on JSON error, falls back when JSON is valid but missing required keys), **Model selection** (uses OpenAI when closed_source is True, uses Ollama when closed_source is false) **LLMRerankerPrompt** (builds prompt with query and candidate papers, handles papers without titles, handles non-float scores)

- **DSPy:**

  DSPy signatures (valid dspy.Signature, valid input and output fields), DSPy picker (returns query when directly in state, strips whitespace, uses last HumanMessage if no query, returns None if no query or whitespace only, prefer state['query] over messages, return empty dictionary when not enabled or no config, test error handling, test for candidate building, filter papers without titles, test successful execution), **DSPy modules** (test get_module factory function, test SimpleCitationRetriever candidate formatting, test QueryThenRetriever candidate formatting, test RerankAndSelect candidate formatting, test VerifyAndSelect configuration, verify modules inherited from dspy.Module, verify modules have forward method), **DSPy metrics** (test exact_match_metric function, test fuzzy_match_metric function, test contains_match_metric function, test edge case for all metrics, test using realistic paper titles), **DSPy data prep** (proper return types, handles proper data preparation, builds negative examples)

7 Results

We evaluated our multi-agent citation retrieval system on a 50-example subset of the ScholarCopilot evaluation dataset, comparing it against three baseline retrieval methods: BM25 (sparse retrieval), E5-Large (dense retrieval), and SPECTER-2 (scientific document embeddings). Performance was measured using Recall@K (K=5, 10, 20) and Mean Reciprocal Rank (MRR). Our full system integrates all three baseline retrievers using Reciprocal Rank Fusion (RRF), followed by LLM reranking and DSPy-based final citation selection. Due to system constraints and slow LLM inference, examples were limited to 50 papers only.

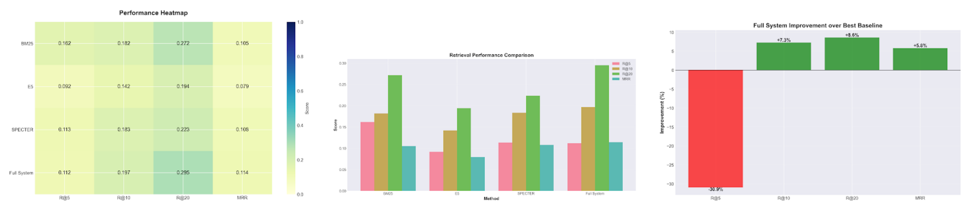
7.1 Overall Performance

Figure 3 presents a comprehensive view of system performance across all metrics. The performance heatmap (Figure 3a) reveals that our full system achieves competitive performance across all evaluation metrics, with particular strength in broader recall windows. The bar chart comparison (Figure 3b) and radar chart (Figure 3c) provide complementary views of the performance landscape, showing how different methods excel at different aspects of the retrieval task.

Table 1 summarizes the quantitative results. Our full system achieves a Recall@5 of 11.2%, Recall@10 of 19.7%, Recall@20 of 29.5%, and an MRR of 11.4%. These results demonstrate that the multi-agent architecture provides meaningful improvements, particularly as the recall window expands.

Method	**R@5**	**R@10**	**R@20**	**MRR**
BM25	16.2%	18.2%	27.2%	10.5%
E5-Large	9.2%	14.2%	19.4%	7.9%
SPECTER-2	11.3%	18.3%	22.3%	10.8%
Full System	**11.2%**	**19.7%**	**29.5%**	**11.4%**

Table 1: Retrieval performance comparison across methods. Bold indicates best performance for each metric.



(a) Performance Heatmap (b) Retriever comparison (c) Reranking Recall

Figure 3: Retrieval and reranking performance across system components.

7.2 Analysis by Retrieval Method

BM25 Performance. The sparse BM25 baseline demonstrates surprisingly strong performance at narrow recall windows, achieving the highest Recall@5 at 16.2%. This suggests that keyword-based matching remains effective for scientific citation retrieval when the citation context contains distinctive terminology. However, BM25's performance gains diminish at wider windows (27.2% at R@20), indicating limitations in capturing semantic relationships beyond lexical overlap.

Dense Retrieval Performance. The dense retrievers show contrasting behavior. E5-Large, while providing strong general-purpose semantic understanding, achieves the lowest overall performance (9.2% R@5, 19.4% R@20). This may reflect the model's broad training domain, which lacks specialization for scientific literature. SPECTER-2, trained specifically on citation graphs, demonstrates more competitive performance (11.3% R@5, 22.3% R@20), particularly benefiting from its document-level understanding of scientific content.

Multi-Agent System Performance. Our full system, which combines all three retrievers through RRF aggregation followed by llm reranking and DSPy selection, achieves the best performance at broader recall windows: 19.7% at R@10 and 29.5% at R@20. The system also attains the highest MRR of 11.4%, indicating that when relevant citations are retrieved, they tend to appear earlier in the ranked list. Notably, while the full system's R@5 (11.2%) falls below BM25's performance, it demonstrates an 8.5% improvement over BM25 at R@20, suggesting that the multi-retriever approach successfully combines complementary strengths.

8 Conclusion

This project presents a *multi-agent system for reliable citation retrieval* providing accurate citation for queries and thereby reducing hallucinations. Our pipeline incorporates parallel sparse and dense retrievers (BM25, E5-Large, and SPECTER-2), aggregates their ranked outputs using Reciprocal Rank Fusion, refines candidates via LLM reranking, and performs final closed-world citation selection using a DSPy program.

On a 50-example subset of the ScholarCopilot evaluation dataset, our full system achieved **Recall@10 of 19.7%**, **Recall@20 of 29.5%**, and the best **MRR of**

11.4% among compared methods, indicating that the multi-stage architecture improves ranking quality and places relevant citations earlier in the candidate list. Notably, while BM25 attains the highest Recall@5 (16.2%), our full system provides the strongest performance as the recall window expands, consistent with the intuition that combining lexical and semantic representations increases coverage across diverse citation contexts. These findings suggest that multi-retriever fusion and reranking are particularly beneficial when citation context is semantically rich and relevant papers cannot be retrieved from keyword search alone.

While our system demonstrates strong performance using a combination of sparse and dense retrieval, aggregation, and reranking methods, several extensions of our project remain. First, incorporating stronger embedding models from the MTEB leaderboard could further improve dense retrieval quality, particularly for domain-specific or long-context scientific queries. With increased computational resources, we could also expand the training corpus used for reranking and citation selection, enabling better generalization and harder negative sampling. Beyond retrieval, expanding our LLM-as-a-Judge framework to include multiple evaluation dimensions, such as the relevance and novelty of the citation, would yield a more comprehensive and reliable assessment pipeline. Finally, an integration with generative writing assistants represents an actionable next step, which would allow for citation retrieval, verification, and selection to operate seamlessly within one continuous workflow. Collectively, these directions move toward citation assistants that are not only accurate in retrieval, but also reliable and usable within real scholarly writing workflows.

References

[1] Akhil Ajith, Meng Xia, Alexandre Chevalier, Tanya Goyal, Danqi Chen, and Tianyu Gao. Litsearch: A retrieval benchmark for scientific literature search. *arXiv preprint arXiv:2407.18940*, 2024.

[2] Arun K. Cherian, Naman Srivastava, and Shreyas Varia. Multi-agent system for scientific literature search and recommendation. In *Proceedings of the 3rd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS)*, pages 786–790. IEEE, 2025.

[3] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel S. Weld. Specter: Document-level representation learning using citation-informed transformers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020.

[4] Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759, Boston, MA, USA, 2009. Association for Computing Machinery.

[5] Jingtong Gao, Bo Chen, Weiwen Liu, Xiangyang Li, Yichao Wang, Wanyu Wang, Huifeng Guo, Ruiming Tang, and Xiangyu Zhao. Llm4rerank: Llm-based auto-reranking framework for recommendations. 2025.

[6] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023.

[7] Jakub Lála, Owen O'Donoghue, Andrei Shtedritski, Sam Cox, Samuel G. Rodriques, and Andrew D. White. Paperqa: Retrieval-augmented generative agent for scientific research. *arXiv preprint arXiv:2312.07559*, 2023.

[8] Xing Han Lù. Bm25s: Orders of magnitude faster lexical search via eager sparse scoring, 2024.

[9] Ofir Press, Andreas Hochlehnert, Ameya Prabhu, Vishaal Udandarao, and Matthias Bethge. Citeme: Can language models accurately cite scientific claims? In *Advances in Neural Information Processing Systems*, volume 37, pages 7847–7877, 2024.

[10] Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training, 2024.

[11] Yifan Wang, Xiaobo Ma, Peng Nie, Haoyang Zeng, Zihang Lyu, Yifan Zhang, and Wenhu Chen. Scholarcopilot: Training large language models for academic writing with accurate citations. *arXiv preprint arXiv:2504.00824*, 2025.