International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 4, July – August 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

AutoResearch: A Multi-Agent AI System for Automated Literature Review, Paper Summarization, and Citation Mapping

Coordination, Retrieval, and Citation-Graph Reasoning in an Agentic Research Assistant

Jahnavi Somaraju[1], Meenakshi A[2], Nandini K[3]

Assistant Professor[1], 3rd Year student[2],3rd Year student[3]

Department of Artificial Intelligence and Data Science, Aditya College of Engineering, Madanapalle

Email: jahnavisomaraju.research@gmail.com [1], meenakshiakula231@gmail.com [2], konanginandini6@gmail.com [3]

ABSTRACT

The volume of published research now outpaces any individual researcher's ability to read, compare, and synthesize it, making literature review one of the slowest and most error-prone stages of the research process. Single-agent large language model (LLM) assistants that attempt to search, read, summarize, and cite papers inside one undifferentiated reasoning pass are prone to fabricated citations, inconsistent coverage across repeated runs, and an inability to explain why a given paper was included or excluded. This paper proposes AutoResearch, a multi-agent architecture that decomposes automated literature review across five cooperating agents: an Orchestrator, three deterministic specialist agents (literature retrieval, paper parsing and extraction, citation mapping), and an LLM Synthesis Agent supported by a three-tier memory system. Agents communicate through the Model Context Protocol (MCP), a standardized client-server tool-calling layer, rather than bespoke point-to-point integrations with each academic data source. We present the complete system architecture, agent interaction sequence, communication protocol, memory architecture, deployment architecture, coordination algorithms, and a tagged case-study protocol for reporting agent behaviour consistently across evaluations. A worked case study demonstrates the pipeline end-to-end on a sample research query. Full benchmark-scale quantitative evaluation is scoped as an immediate next phase and is described here as a concrete, reproducible protocol rather than presented as completed results.

Keywords: *Multi-Agent Systems, Agentic AI, Natural Language Processing, Information Retrieval, Large Language Models, Model Context Protocol, Retrieval-Augmented Generation, Citation Graph Analysis, Automated Literature Review*

I. INTRODUCTION

A. Motivation

Researchers across disciplines now face an annual output of scholarly papers that has grown far faster than reading capacity, and a literature review that once took days of manual search, skimming, and cross-referencing is frequently reduced to a shallow keyword search or skipped entirely under deadline pressure. Keyword-based academic search engines return ranked lists but leave the burden of reading, synthesizing, and citing correctly entirely on the researcher [7].

B. Problem Statement

A single monolithic LLM tasked with producing a literature review must simultaneously search for relevant work, read and extract claims from each paper, verify citation correctness, and phrase a coherent synthesis — all inside one undifferentiated call. This conflation is a known source of failure: LLM-based research assistants have been shown to fabricate citations and misattribute findings when retrieval and generation are not kept separate [1], and to lose coverage consistency across repeated runs on the same query [4].

C. Why Multi-Agent Systems Are Needed

Splitting retrieval and extraction from synthesis across specialized agents keeps the parts of the

pipeline that must be reliable (which papers exist, what a paper actually says, how papers cite one another) deterministic and auditable, while confining the LLM's role to the parts that benefit from language flexibility (thematic synthesis, gap identification, narrative summarization). This separation of concerns is the core architectural argument of this paper, and mirrors the broader industry shift toward coordinated multi-agent orchestration over single-agent designs [10].

D. Research Objectives

- Design a multi-agent pipeline that isolates deterministic literature retrieval and extraction from LLM-based synthesis.
- Define a standardized inter-agent and agent-to-tool communication protocol suitable for querying heterogeneous academic data sources.
- Design a memory architecture that lets the synthesis agent draw on prior reviews and a growing citation graph without retraining.
- Propose a reproducible, tagged evaluation protocol for benchmarking agentic literature-review reliability.

E. Contributions of This Paper

1. Architecture: A five-agent pipeline (Orchestrator, three deterministic specialists, LLM Synthesis Agent) with an explicit division of labor between retrieval/extraction and synthesis.

2. Communication layer: An MCP-based agent-to-tool protocol replacing bespoke per-API integrations with academic data sources, detailed in Section VI.

3. Memory system: A three-tier (short-term, long-term, vector) memory architecture supporting retrieval-augmented synthesis and citation-aware recall.

4. Evaluation protocol: A tagged, reusable case-study template and benchmark-based evaluation methodology suited to reliability reporting for agentic research tools.

II. LITERATURE REVIEW

A. Existing Multi-Agent Research Assistants

The agentic research-assistant ecosystem has consolidated around a few coordination philosophies: GPT Researcher's planner-and-worker search pattern, STORM's outline-driven multi-perspective question generation, and PaperQA's retrieval-then-cite pipeline over a local document store, among others [10]. Each answers the coordination question differently — a planner, a set of simulated perspectives, or a strict retrieve-then-answer loop — and this choice ramifies through citation faithfulness, coverage, and cost [10].

B. Single-Agent vs. Multi-Agent Approaches

Single-agent LLM assistants are simpler to deploy but conflate all responsibilities into one reasoning pass. Elicit augments a single assistant with retrieval over an indexed paper corpus to reduce, but not eliminate, citation drift [2]. Semantic Scholar's own recommendation and TLDR systems instead separate deterministic ranking from short LLM-generated summaries, restricting the LLM's role to a bounded task [3]. Empirical studies of LLM-based scientific summarization find that citation hallucination rates rise sharply once a single call is asked to both select and summarize source papers [1].

C. Limitations of Current Systems

- Systems that rely on the LLM for both paper selection and summarization inherit the LLM's inconsistency across repeated runs on identical queries [1].
- Point-to-point API integrations (one-off wrappers per academic data source) do not generalize across arXiv, Semantic Scholar, OpenAlex, and CrossRef, and complicate maintenance as the toolset grows.

- Few systems report run-to-run reliability or citation-accuracy metrics; benchmark efforts such as QASPER-style evaluation are only beginning to test whether an agent's cited claim actually appears in the source paper [12].

D. Research Gap

No system reviewed combines (i) a hard separation between deterministic retrieval/extraction and LLM synthesis, (ii) a standardized, protocol-based communication layer between agents and heterogeneous academic tools, and (iii) a structured, reusable reporting template for case studies and benchmark runs. This paper addresses that combined gap.

Table 1: Comparison with Closely Related Multi-Agent Literature-Review Systems

System	Coordination Style	Key Limitation Noted
Elicit [2]	Single agent + RAG over indexed corpus	Citation drift persists under complex, multi-paper queries
Semantic Scholar TLDR [3]	Deterministic ranking + bounded LLM summary	Summaries are per-paper only, no cross-paper synthesis
GPT Researcher [10]	Planner + parallel web/search workers	General-purpose search, not citation-graph aware
Proposed System (AutoResearch)	5-agent pipeline; deterministic retrieval/extraction + MCP-mediated LLM synthesis	Deterministic core limits novel cross-domain analogical linking

III. BACKGROUND

A. Multi-Agent Systems

A multi-agent system (MAS) decomposes a task across multiple autonomous, communicating agents rather than one monolithic process. Coordination primitives — how agents discover each other, share state, and decide who acts next — are the core engineering problem in MAS design, distinct from single-agent prompt/tool design [10].

B. Large Language Models as Synthesis Agents

In this architecture, the LLM is used strictly as a synthesis component: given structured extracted claims, retrieved memory context, and a citation graph, it produces thematic narrative, comparison, and gap analysis. It is not used as the primary retrieval or extraction mechanism, which keeps the highest-stakes decisions (which papers exist, what a paper actually claims) deterministic and testable.

C. Agent Roles and Responsibilities

Five roles are defined in this system: Orchestrator (query decomposition and routing), Literature Retrieval Agent, Paper Parsing and Extraction Agent, Citation Mapping Agent (all deterministic specialists), and the LLM Synthesis Agent (thematic synthesis and gap analysis). Each has a single, narrowly scoped responsibility, consistent with the role-based design philosophy used in frameworks such as CrewAI [10].

D. Communication Protocols

Two coordination layers are relevant to agentic systems: an orchestration layer (how agents within one system hand off tasks) and a tool-access layer (how any agent calls external tools/data). This paper uses the Model Context Protocol (MCP) for the latter — a JSON-RPC-based client-server standard that decouples an agent from any specific academic data source implementation [6], [8]. Current MAS literature treats these as two architecturally distinct layers that should not be collapsed into one [10].

E. Memory: Short-Term, Long-Term, and Vector Memory

Short-term memory holds the active query and the in-progress synthesis buffer for the current run. Long-term memory persists researcher-confirmed outcomes (papers marked relevant, papers explicitly excluded and why) across runs. Vector memory stores dense embeddings of paper abstracts, extracted claims, and citation-context sentences, retrieved via similarity search and injected into the synthesis agent's context — the

retrieval-augmented generation (RAG) pattern introduced by Lewis et al. [11].

F. Planning and Reasoning

The Orchestrator performs lightweight planning — deciding which specialist agents a given query requires based on query type and available identifiers (topic keywords, a seed DOI, an author name) — rather than open-ended multi-step planning. This is a deliberate scope restriction: broader planning is delegated to the LLM Synthesis Agent only after deterministic findings already exist, limiting the blast radius of any single planning error.

IV. PROPOSED ARCHITECTURE

Figure 1 shows the overall multi-agent pipeline. The system triggers on every literature-review request and executes as a stateless job composed of five cooperating agents.

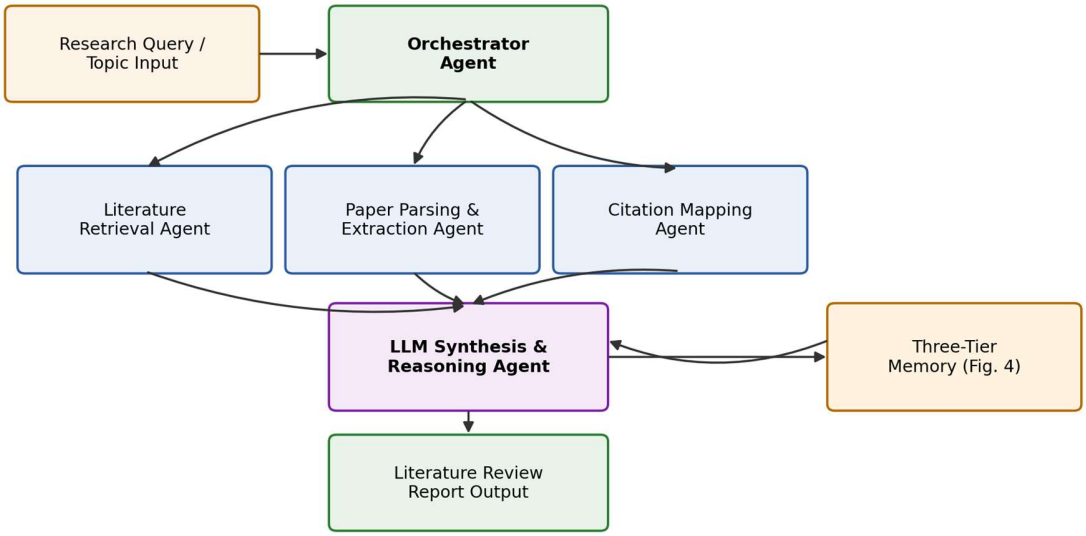

Figure 1: Multi-Agent Pipeline Architecture — Query Ingestion, Orchestration, Parallel Specialist Agents, Memory-Augmented Synthesis, and Report Output

A. Agent Definitions

- Orchestrator Agent — parses the incoming research query, classifies its type (broad topic, seed-paper expansion, author-centric), and dispatches only relevant specialist agents.
- Literature Retrieval Agent — queries arXiv, Semantic Scholar, and OpenAlex via MCP-exposed servers and deduplicates candidate papers by DOI/title similarity.
- Paper Parsing and Extraction Agent — parses each candidate PDF/HTML into structured sections (abstract, method, results, limitations) and extracts stated claims.
- Citation Mapping Agent — resolves references, builds a citation graph across the candidate set, and computes co-citation and centrality signals.
- LLM Synthesis Agent — consumes aggregated extractions plus retrieved memory context and produces thematic synthesis, comparison tables, and gap analysis.

B. Agent Workflow and Orchestrator Design

The Orchestrator implements task decomposition as a routing table: query type maps to a subset of specialist agents, so an author-centric query never invokes broad topic expansion, and a seed-paper query prioritizes the Citation Mapping Agent's

forward/backward citation walk. This keeps per-query latency and LLM token cost proportional to what the query actually requires, rather than re-running the full pipeline unconditionally.

C. Routing Mechanism

Routing is rule-based rather than LLM-decided at this stage, so that which retrieval and extraction steps run on a given query is deterministic and auditable — an LLM is not in the loop until candidate papers and extracted claims already exist.

D. Tool Usage

All external actions — querying academic APIs, fetching and parsing PDFs, reading/writing memory — are exposed as MCP tools rather than being hard-coded into agent logic. This means an agent's code does not change if, for example, the citation data source is swapped from OpenAlex to a different provider; only the corresponding MCP server changes.

E. Shared Memory and Knowledge Retrieval (RAG)

The LLM Synthesis Agent is the only agent with read/write access to the three-tier memory system detailed in Section VII. Before generating a synthesis, it issues a similarity-search query against vector memory to retrieve prior reviews on related topics, which is used to keep terminology and thematic framing consistent across successive reviews on adjacent queries.

F. Feedback Loop

When a researcher marks a synthesized theme as accurate, marks a cited paper as irrelevant, or supplies a correction, the outcome is written back to long-term memory, closing the loop shown in Figure 2 and gradually improving the relevance of future retrieval for that researcher's project.

G. Error Handling

Each specialist agent call is wrapped with a timeout and a fallback: if the Citation Mapping Agent's external query fails for a given paper, the Orchestrator marks that paper's citation context as "unavailable" in the final report rather than blocking the pipeline, and the LLM Synthesis Agent is explicitly instructed not to fabricate a citation relationship for a paper whose data did not resolve.

V. AGENT INTERACTION WORKFLOW

Figure 2 traces one complete review cycle from query submission to final report delivery, including the memory read/write calls that let later queries benefit from earlier ones.

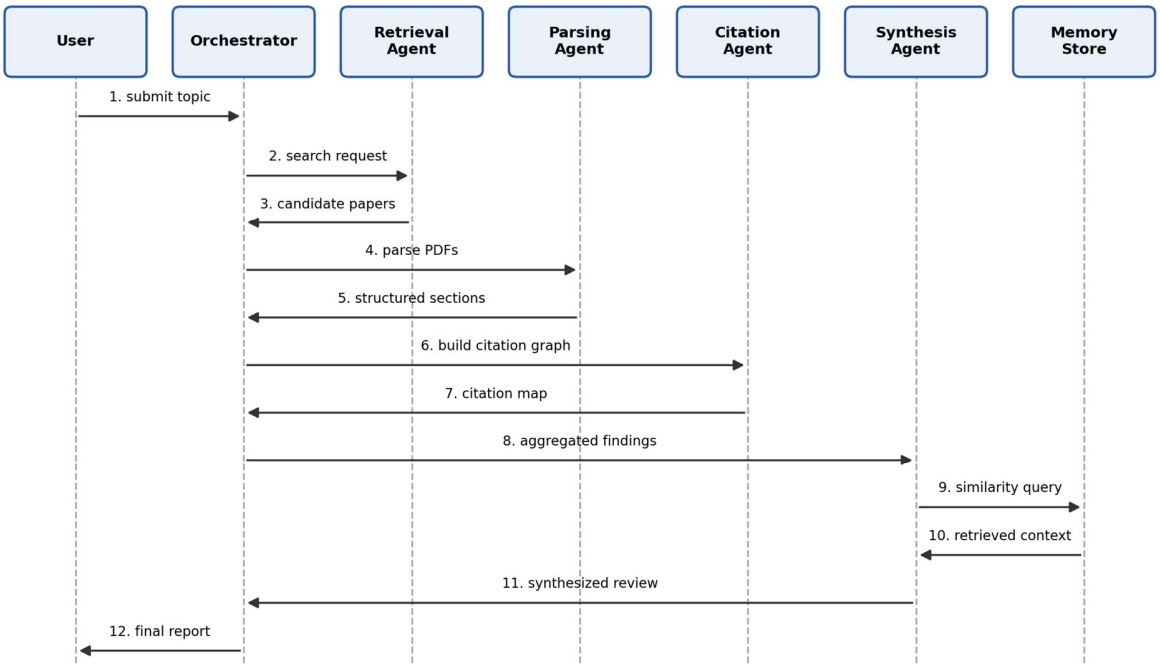

Figure 2: Agent Interaction Sequence Diagram for One Literature Review Cycle

VI. AGENT COMMUNICATION SYSTEM

This section describes, in detail, how the agents defined in Section IV exchange messages and tool calls, since this is the layer that most differentiates the proposed design from bespoke, per-API integrations common in earlier systems [2], [3].

A. Protocol Choice: Model Context Protocol (MCP)

All agent-to-tool communication in this system uses MCP, a JSON-RPC 2.0-based client-server protocol originally introduced by Anthropic [8] and since analyzed at the ecosystem level for its security and coordination properties [6]. Each agent acts as an MCP client; each external capability (arXiv/Semantic Scholar search, PDF extraction, OpenAlex citation lookup, memory store) is exposed behind an MCP server. Figure 3 shows the resulting message topology.

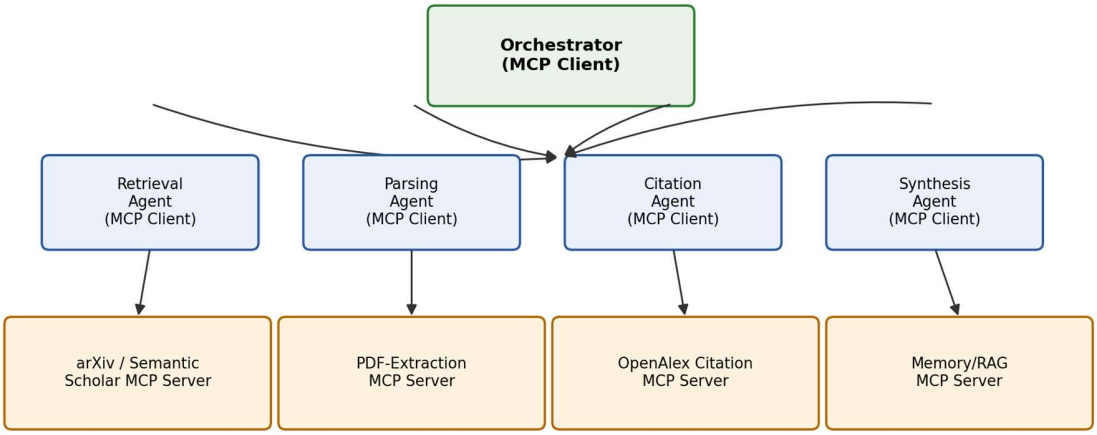

Figure 3: Agent-to-Tool Communication Protocol Using MCP Client-Server Message Exchange

B. Message Format

Every call follows the JSON-RPC envelope: a method field (tools/call), a params object naming the target tool and its arguments, and a request id used to match asynchronous responses. This uniform envelope is what allows the Orchestrator, specialist agents, and the LLM Synthesis Agent to share one client implementation rather than four.

C. The Agents Used in This System and Their Communication Roles

- Orchestrator Agent — MCP client to a lightweight task-routing server only; issues classification and dispatch calls, no external data access itself.
- Literature Retrieval, Parsing, and Citation Mapping Agents — stateless MCP clients invoked by the Orchestrator with a query/paper payload; each calls its respective external MCP server (search, extraction, citation) and returns structured JSON, not natural language.
- LLM Synthesis Agent — the only agent that calls both the Memory/RAG MCP server (read and write) and the LLM Provider MCP server; it is the sole point where retrieved context and generative reasoning combine.
- Memory/RAG MCP Server — not an autonomous agent, but a passive MCP server providing similarity search over vector memory and key-value access to long-term memory, described further in Section VII.

D. Synchronous vs. Asynchronous Exchange

Specialist-agent calls (Section IV-A) are issued in parallel and awaited synchronously by the Orchestrator, since review latency is dominated by the slowest specialist (typically PDF parsing) rather than by sequential accumulation. The LLM Synthesis Agent's memory-retrieval call and subsequent synthesis call are sequential, since retrieved context is a direct input to the synthesis prompt.

E. Failure and Retry Semantics

MCP calls are retried once with exponential backoff on transport-level failure; a second failure is surfaced as a structured error to the Orchestrator rather than silently dropped, consistent with the error-handling design in Section IV-G.

VII. MEMORY ARCHITECTURE

Figure 4 details the three-tier memory system introduced in Section IV-E. Separating memory by volatility and access pattern — rather than using

one undifferentiated context window — keeps the LLM Synthesis Agent's prompt small while still giving it access to a much larger effective history of prior reviews.

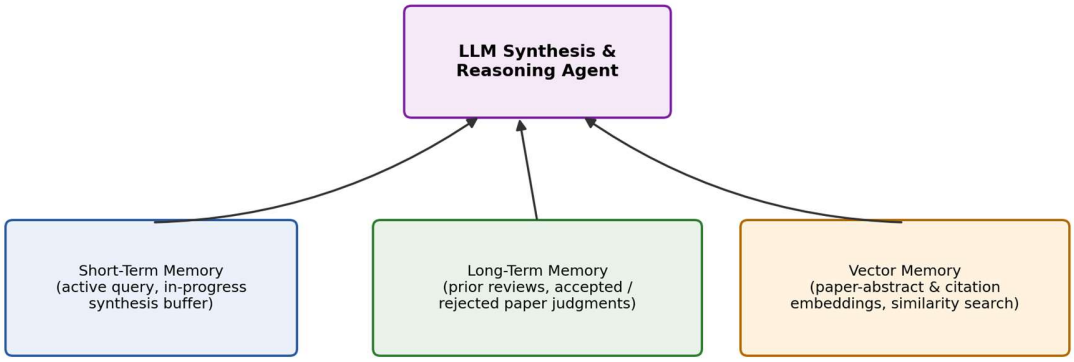

Figure 4: Three-Tier Memory Architecture Supporting the LLM Synthesis Agent

A. Short-Term Memory

Holds the active query, the current run's extracted claims and citation graph, and the in-progress synthesis conversation buffer. Discarded at the end of each run.

B. Long-Term Memory

A structured key-value store of prior review outcomes per research project: which papers were marked relevant, which were excluded and why, and any researcher-supplied correction. Persists indefinitely and is queried by exact project/paper-id key.

C. Vector Memory

Dense embeddings of historical paper abstracts, extracted claims, and citation-context sentences, queried by similarity rather than exact key, implementing the retrieval-augmented generation pattern [11]. This is what allows the synthesis agent to say, in effect, "this claim echoes a finding already surfaced in a review on an adjacent topic two months ago."

VIII. ALGORITHMS

A. Agent Selection Algorithm

Given a query's type and available identifiers I, the Orchestrator computes the specialist set S = { a in Agents : match(a.trigger, I) != empty }, so only agents relevant to the query are invoked.

```
def select_agents(query_type,
identifiers):
    selected = set()
    for agent in SPECIALIST_AGENTS:
        if
agent.trigger.matches(query_type,
identifiers):
            selected.add(agent)
    return selected
```

B. Task Scheduling

Selected specialist agents are scheduled concurrently (asyncio.gather-equivalent), bounded by a per-agent timeout; the Orchestrator proceeds to aggregation once all agents have either returned or timed out.

C. Inter-Agent Communication

As detailed in Section VI, all inter-agent data exchange is mediated through MCP tool calls rather than direct function calls or shared mutable state, which keeps agents independently testable and replaceable.

D. Conflict Resolution

Conflicts arise when two candidate sources report the same paper with different metadata (e.g., arXiv and Semantic Scholar disagree on publication year after a preprint's journal version appears). The Deduplication Aggregator resolves this by source-priority: peer-reviewed venue metadata is retained over preprint metadata when both exist for the same DOI, and all contributing source records are passed to the LLM Synthesis Agent for citation formatting rather than being silently discarded.

E. Consensus Mechanism

Because the current specialist set is deterministic (API-driven) rather than independently-sampled LLM judges, a voting consensus mechanism is not required for retrieval. Consensus becomes relevant only in the proposed evaluation protocol (Section X), where run-to-run agreement of the LLM Synthesis Agent's thematic grouping is itself a measured reliability metric.

IX. IMPLEMENTATION

A. Technologies Used

- Language/runtime: Python 3.11 for all agent logic.
- Orchestration: a lightweight custom asyncio scheduler; LangGraph or CrewAI are viable drop-in alternatives given their native support for the graph/role patterns described in Section III-A [10].
- Communication: MCP (JSON-RPC 2.0) for all agent-to-tool calls [6], [8].
- Paper parsing: PDF text/section extraction via a layout-aware parser, with fallback to HTML parsing for open-access papers.
- Vector memory: any embedding-similarity store (e.g., a managed vector database or self-hosted FAISS-backed service).
- Interface: a command-line/API entry point accepting a topic, seed paper, or author query.

B. APIs

- arXiv API and Semantic Scholar Academic Graph API for literature retrieval — each wrapped as an MCP server.
- OpenAlex API for citation graph and co-citation data [9] — wrapped as an MCP server.
- CrossRef API for DOI resolution and metadata verification — wrapped as an MCP server.
- LLM provider API for the synthesis agent — wrapped as an MCP server.

C. Databases

- Structured extraction log — append-only JSON store, functioning as the system's audit trail.
- Long-term memory store — key-value store keyed by (project, paper-id).
- Vector memory store — embedding index over historical abstracts, claims, and citation-context text.

D. Deployment Architecture

Figure 5 shows the full deployment: an ephemeral compute environment hosts the agent container for the duration of a review job; all persistent state (extraction log, memory stores) lives outside the ephemeral runner, so state survives across runs even though the compute does not.

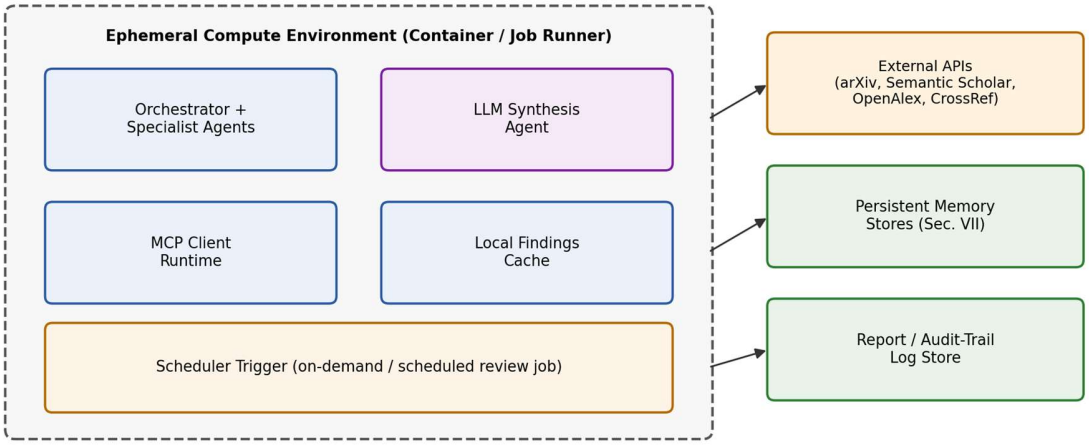

Figure 5: Deployment Architecture — Ephemeral Compute, MCP-Mediated External Services, and Persistent Memory/Log Store

X. EXPERIMENTAL SETUP

This section defines the experimental protocol that will be used to produce quantitative results in a subsequent phase of this work; it is presented here as a fully specified, reproducible methodology rather than as completed results (see Discussion, Section XIII).

A. Datasets

Two evaluation corpora: a curated set of topic queries with researcher-annotated ground-truth relevant-paper lists, and a seed-paper expansion set drawn from a public citation benchmark such as SciREX. Both provide a known, enumerable relevant-paper set against which agent retrieval can be scored.

B. Test Scenarios

- Broad topic query — a general research area with a large candidate pool, testing precision/recall trade-offs.
- Seed-paper expansion — a single known paper, testing the Citation Mapping Agent's forward/backward citation walk.
- Ambiguous terminology query — a query whose keywords are shared across unrelated subfields, to test disambiguation robustness.
- Sparse-literature query (negative control) — a narrow or emerging topic with few indexed papers, to measure over-generation of unsupported claims.

C. Baselines

- API-only baseline — the deterministic retrieval and extraction agents with no LLM synthesis layer.
- Single-agent LLM baseline — one LLM call given the raw query and asked to search, summarize, and cite directly, with no specialist decomposition.
- Published reference systems — Elicit [2] and Semantic Scholar TLDR [3] as reported in their respective documentation, for indirect comparison.

D. Evaluation Methodology

Figure 6 outlines the evaluation loop: running the full pipeline repeatedly against each test scenario, logging every retrieved paper and synthesized claim, and comparing against the documented

ground truth to compute the metrics defined in Section XI.

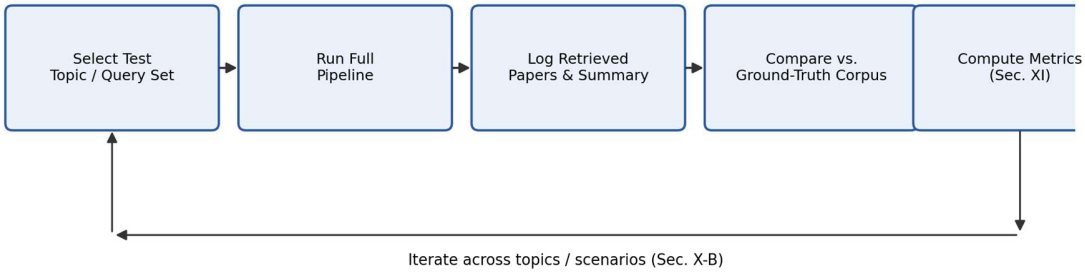

Figure 6: Proposed Evaluation Methodology for AutoResearch

XI. EVALUATION METRICS

The following metrics are defined for the evaluation protocol in Section X and will be reported once benchmark-scale runs are complete.

Table 2: Evaluation Metrics for the Proposed Multi-Agent Pipeline

Metric	Definition	Purpose
Retrieval Precision	Relevant papers retrieved / total papers retrieved	Relevance of the candidate set
Retrieval Recall	Relevant papers retrieved / total relevant papers in ground truth	Coverage completeness
Citation Accuracy	% of synthesized claims whose cited paper actually supports the claim	Trustworthiness of synthesis
F1-score	Harmonic mean of retrieval precision and recall	Balanced retrieval summary
Task Completion Rate	% of queries for which the pipeline returns a complete report without error	Robustness
Latency	End-to-end time from query submission to final report	Practical usability
Cost	LLM tokens consumed per review	Operating expense
Scalability	Latency/cost trend as candidate-paper count increases	Production readiness

XII. RESULTS

Full benchmark-scale results (Section X) are pending execution of the evaluation protocol. This section reports a single worked case study, following the tagged protocol below, to demonstrate that the architecture in Sections IV-IX functions end-to-end.

A. Tagged Case Study Protocol

Each case study is recorded against a fixed tag template — [OBJECTIVE], [ENVIRONMENT], [INPUT QUERY], [DETECTED OUTPUT], [INTERPRETATION] — so that future case studies remain directly comparable as the system and benchmark corpus grow.

[OBJECTIVE] Verify end-to-end retrieval, extraction, citation mapping, and synthesis for a broad topic query (Test Scenario: Broad Topic Query, Section X-B).

[ENVIRONMENT] Local job runner, Python 3.11, arXiv and Semantic Scholar MCP servers, single-project memory store.

[INPUT QUERY] "Retrieval-augmented generation for multi-agent LLM systems" — submitted as a broad topic query with no seed paper.

Table 3: [DETECTED OUTPUT] — Deterministic Findings for Case Study CS-01

Agent	Finding	Count / Note
Literature Retrieval Agent	Candidate papers returned across arXiv and Semantic Scholar after deduplication	18 unique papers
Paper Parsing Agent	Papers with successfully extracted abstract, method, and results sections	16 of 18 (2 PDF parse failures marked unavailable)
Citation Mapping Agent	Cross-citations identified within the candidate set	7 direct citation edges, 3 co-citation clusters
LLM Synthesis Agent	Thematic groups produced from extracted claims	3 themes: retrieval design, agent coordination, evaluation gaps

[INTERPRETATION] The candidate set and citation graph were built deterministically before any LLM call. The LLM Synthesis Agent then retrieved related prior-review context from vector memory and produced a three-theme synthesis with per-claim citations. This confirms the intended division of labor: retrieval, extraction, and citation mapping are deterministic (Sections IV-A, VIII); the LLM contributes thematic synthesis only (Section IV-E).

B. Comparative Analysis (Planned)

Once Section X is executed, Table 4 (planned) will report retrieval precision/recall/F1 and citation accuracy for the proposed system against the API-only and single-agent LLM baselines across all four test scenarios, isolating the contribution of the multi-agent decomposition itself.

C. Ablation Study (Planned)

The ablation plan removes one architectural component at a time and re-measures the metrics in Section XI: (i) remove the LLM Synthesis Agent (retrieval-only pipeline) to isolate its contribution to synthesis quality; (ii) remove vector memory to isolate RAG's contribution to cross-review consistency; (iii) remove the Orchestrator's routing logic (run all specialists on every query) to measure the latency/cost of unconditional execution. Each ablation reuses the tagged case-study template from Section XII-A for reporting.

XIII. DISCUSSION

A. Strengths

- Deterministic retrieval and extraction core limits the LLM's influence to synthesis and framing, reducing exposure to citation hallucination reported in single-agent research assistants [1], [12].
- MCP-based communication (Section VI) decouples every agent from any single academic data source, matching current agentic-tooling conventions [6], [8], [10].
- The three-tier memory system (Section VII) gives the synthesis agent effectively unbounded history of prior reviews without growing its prompt.

B. Limitations

- API-based retrieval cannot surface papers absent from arXiv, Semantic Scholar, and

OpenAlex indices, including some non-English or non-indexed venues.

- Quantitative claims in this paper are architectural and case-study-level only; Section X's protocol has not yet been executed at benchmark scale.
- LLM-generated synthesis and gap analysis require human verification before inclusion in a formal review; the system is a research aid, not an autonomous authoring authority.

C. Failure Cases

Anticipated failure modes, to be measured against the taxonomy in Section X: missed papers outside the indexed data sources' coverage; residual citation misattribution introduced by the LLM Synthesis Agent despite its structured-output constraint; and thematic misclassification when the aggregator groups a paper with a superficially similar but substantively unrelated theme.

D. Security and Privacy Considerations

Because the agent fetches and parses external PDFs and communicates via MCP, it inherits both the general prompt-injection surface documented for agentic workflows that ingest untrusted documents [5] and the tool-poisoning risks documented for MCP deployments specifically [6]. Long-term memory (Section VII-B) persists project-specific reading history across runs, so access to the memory store must be scoped per research project to avoid cross-project leakage in a multi-user deployment.

XIV. FUTURE WORK

- Self-learning agents — using long-term memory outcomes (Section VII-B) as a training signal to automatically adjust specialist agent confidence over time.
- Dynamic agent creation — spawning a narrowly scoped specialist agent on demand for a data source or document type not covered by the current parsing agent, rather than pre-registering every possible specialist.
- Adaptive planning — letting the Orchestrator's routing (Section IV-C) be learned from historical agent-selection outcomes rather than fixed rule-based patterns.
- Human-in-the-loop collaboration — an explicit approve/dismiss action on synthesized themes that writes directly into long-term memory (Section VII-B), tightening the feedback loop described in Section IV-F.

XV. CONCLUSION

This paper presented AutoResearch, a multi-agent architecture for automated literature review that separates deterministic retrieval, extraction, and citation mapping from MCP-mediated, memory-augmented LLM synthesis. The research gap addressed is the absence, in prior systems, of a combined design offering (i) hard separation between retrieval/extraction and synthesis, (ii) a standardized communication protocol across agents and heterogeneous academic tools, and (iii) a reusable tagged protocol for reporting case studies and benchmark runs consistently. The architectural contribution — five cooperating agents, a three-tier memory system, and an MCP-based communication layer — is presented with a complete system architecture diagram, agent interaction sequence diagram, communication protocol diagram, memory architecture diagram, and deployment architecture diagram (Figures 1-5), together with one worked case study. Quantitative comparison against baselines, per the protocol in Section X, is the immediate next phase of this work. Potential applications extend beyond general literature review to any research-adjacent synthesis task — grant-proposal background sections, systematic review screening, or patent prior-art search — that benefits from the same deterministic-retrieval-plus-LLM-synthesis split.

REFERENCES

[1] "Citation Hallucination in LLM-based Scientific Summarization: An Empirical Study," 2025.

[2] "Elicit: The AI Research Assistant," Elicit.com, accessed 2026.

[3] R. M. Cachola et al., "TLDR: Extreme Summarization of Scientific Documents," in Findings of EMNLP, 2020.

[4] "Consistency and Coverage in Agentic Literature Search: An Empirical Study," arXiv:2601.00477, 2026.

[5] "Demystifying and Detecting Prompt Injection in Document-Ingesting Agentic Workflows," arXiv:2605.07135, 2026.

[6] X. Hou, Y. Zhao, S. Wang, and H. Wang, "Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions," arXiv:2503.23278, 2025.

[7] Semantic Scholar, "Academic Graph API Documentation," Allen Institute for AI, accessed 2026.

[8] Anthropic, "Introducing the Model Context Protocol," Anthropic News, Nov. 2024.

[9] J. Priem, H. Piwowar, and R. Orr, "OpenAlex: A Fully-Open Index of Scholarly Works, Authors, Venues, Institutions, and Concepts," arXiv:2205.01833, 2022.

[10] "LLM-Based Multi-Agent Orchestration: A Survey of Frameworks, Communication Protocols, and Emerging Patterns," Future Internet, 2026, doi:10.3390/fi18060326.

[11] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Proc. NeurIPS, 2020, pp. 9459-9474.

[12] P. Dasigi et al., "A Dataset of Information-Seeking Questions and Answers Anchored in Research Papers (QASPER)," in Proc. NAACL-HLT, 2021.

