International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

# AMAR: An Autonomous Multi-Agent Researcher for End to-End Automated Scientific Literature Review and Draft Generation

Prof. Rajasekaran K, Prof. Lavanya Vijayan, Karthik Kashyap K S, Pratham C S, Kiran D
Department of Artificial Intelligence and Data Science
S.E.A. College of Engineering and Technology
Visvesvaraya Technological University, Belagavi, Bangalore, India

***Abstract*—**We present AMAR (Autonomous Multi-Agent Researcher), an end-to-end web-based system that automates the complete academic research workflow — from literature discovery through experiment execution to draft generation. AMAR orchestrates a pipeline of seven specialized AI agents: a Searcher Agent that retrieves papers from arXiv, OpenAlex, CrossRef, and IEEE Xplore APIs; a Summarizer Agent; a Critic Agent that identifies research gaps; a Developer Agent that synthesizes executable Python experiment code; an Experimenter Agent that executes generated code in an isolated Docker sandbox; a Verifier Agent that validates citations and results against academic databases; and a Writer Agent powered by Google Gemini 2.0 Flash that produces structured research drafts. The system is built on a React 18 + TypeScript frontend with Framer Motion animations, a Node.js/Express backend, and WebSocket-driven real-time agent status monitoring. A persistent session memory layer enables cross-session knowledge retention. AMAR reduces the typical manual research workflow from several hours to a fully automated pipeline, producing a verified draft complete with inline citations, experiment results, and an interactive knowledge graph of paper relationships. The system demonstrates that decomposing research tasks across specialized LLM-backed agents, combined with secure containerized execution, can significantly improve the reproducibility and rigor of AI-assisted research.

***Index Terms*—***Multi-agent systems, autonomous research, large language models, literature review automation, Docker sandboxing, knowledge graph, citation verification, Gemini API, React, Node.js*

## I. INTRODUCTION

The rapid expansion of academic literature across domains such as machine learning, natural language processing, and data science has created a significant bottleneck for researchers. Conducting a thorough literature review — identifying relevant papers, synthesizing methods, recognizing open research questions, designing experiments, and composing a structured draft — requires days or weeks of skilled effort. This bottleneck limits the pace of scientific progress and creates barriers for early-stage researchers who lack domain expertise.

Recent advances in large language models (LLMs) have demonstrated strong capabilities in text summarization, code generation, and question answering. However, most existing AI-assisted research tools address only isolated fragments of this workflow — offering either paper recommendation, abstract summarization, or draft assistance in isolation — rather than providing an integrated, end-to-end autonomous pipeline.

We introduce AMAR (Autonomous Multi-Agent Researcher), a production-ready system that automates the full research lifecycle. AMAR decomposes the research process into seven specialized agent roles coordinated by a central Orchestrator. Each agent focuses on a well-defined task: discovery, summarization, gap analysis, code generation, secure experiment execution, result verification, and draft writing. The complete pipeline runs without human intervention, delivering a verified research draft alongside reproducible experiment artifacts.

The key contributions of this paper are:

- Multi-Agent Orchestration: A seven-agent pipeline with a central Orchestrator that coordinates sequential task execution, provides real-time status updates via WebSocket, and handles error recovery.
- Secure Experiment Sandbox: A Docker-based execution environment that runs LLM-generated Python experiments in isolated containers with resource monitoring and log capture.
- Citation and Results Verification: An automated Verifier Agent that cross-checks generated citations and experiment results against academic databases, producing confidence-scored verification reports.
- Knowledge Graph Visualization: A D3.js force-directed graph rendering inter-paper relationships, enabling visual exploration of the discovered literature space.
- Session Memory Persistence: A cross-session knowledge retention layer that stores research context, enabling researchers to resume and build upon previous sessions.

## II. RELATED WORK

### *A. AI-Assisted Literature Review*

Elicit [1] and Semantic Scholar [2] represent the current state-of-the-art in AI-assisted literature search, offering paper recommendations and abstract-level summarization. However, these systems do not generate experiment code, execute experiments, or produce structured drafts. Connected Papers [3] visualizes citation graphs but performs no synthesis. Consensus [4] provides LLM-backed question answering over scientific papers but lacks end-to-end automation. AMAR differentiates itself by covering the full

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

pipeline from search to verified draft in a single, integrated system.

### *B. Multi-Agent LLM Systems*

AutoGPT [5] and BabyAGI [6] pioneered autonomous LLM agents capable of goal-directed task decomposition. MetaGPT [7] introduced role-playing agents for software engineering. ChatDev [8] demonstrated multi-agent collaboration for code generation. CAMEL [9] formalized role-playing communication between agents. AMAR adapts multi-agent principles to the academic research domain, introducing domain-specific agents (Critic, Verifier) not present in prior software-engineering-focused frameworks.

### *C. Automated Code Generation and Execution*

Code Interpreter in ChatGPT and the Code Llama family [10] have demonstrated LLM-based code generation and execution. Sandboxed code execution has been explored in educational and software testing contexts [11]. AMAR extends this paradigm to scientific experiment execution, using Docker containerization to enforce reproducibility and security — critical requirements when running untrusted LLM-generated code.

### *D. Research Gap Analysis*

Identifying research gaps traditionally requires expert domain knowledge. Recent work on scientific question generation [12] and critical analysis of literature [13] has begun to automate this process. AMAR's Critic Agent provides structured gap identification as a first-class component of the research pipeline, enabling downstream experiment design to target identified open problems.

## III. SYSTEM ARCHITECTURE

### *A. Overview*

AMAR follows a layered architecture comprising three tiers: a React-based frontend, a Node.js/Express backend orchestrating the agent pipeline, and external AI and academic database services. Fig. 1 illustrates the complete system. The user submits a research topic via the web interface. The Orchestrator then dispatches agents sequentially, emitting real-time status events via WebSocket to the frontend dashboard. Final outputs — a structured draft, experiment code, verification report, and knowledge graph — are rendered in the Results Lab interface.

| Layer | Technology | Role |
|---|---|---|
| Frontend | React 18, TypeScript, Vite | UI, real-time dashboard, knowledge graph |
| Styling/Anim. | Tailwind CSS, Framer Motion, shadcn/ui | Glassmorphism UI, agent animations |
| Backend | Node.js 18, Express 4 | REST API, orchestration, session storage |
| AI Inference | Google Gemini 2.0 Flash | Summarization, gap analysis, code gen, writing |
| Paper APIs | arXiv, OpenAlex, CrossRef, IEEE Xplore | Literature discovery |
| Sandbox | Docker (dockerode SDK) | Isolated Python experiment execution |
| Real-time | WebSocket (Socket.IO) | Agent status streaming to frontend |
| Persistence | Firebase / SUPABASE | Session memory, project history |

TABLE I: AMAR SYSTEM TECHNOLOGY STACK

### *B. Agent Pipeline*

The Orchestrator class maintains a registry of seven agents, each with an independent status state (idle, running, completed, error) and execution log. The pipeline executes sequentially; each agent receives the accumulated context produced by all prior agents, enabling rich inter-agent information sharing. The Orchestrator emits notifyUpdate events at each state transition, which are relayed to the frontend via WebSocket for live visualization.

| Agent | Input | Output | Technology |
|---|---|---|---|
| 1. Searcher | Research topic string | Paper list (title, summary, URL) | arXiv, OpenAlex, CrossRef, IEEE Xplore APIs |
| 2. Summarizer | Paper list | Structured summaries per paper | Gemini 2.0 Flash |
| 3. Critic | Paper summaries | Research gap list | Gemini 2.0 Flash |
| 4. Developer | Topic + gaps | Python experiment code | Gemini 2.0 Flash |
| 5. Experimenter | Experiment code | Execution logs + metrics JSON | Docker (Python 3.10 container) |
| 6. Verifier | Citations + metrics | Verification report + scores | Academic DB cross-check + Gemini |
| 7. Writer | All above | Structured research draft | Gemini 2.0 Flash |

TABLE II: AGENT PIPELINE — INPUTS, OUTPUTS AND TECHNOLOGIES

### *C. Docker-Based Experiment Sandbox*

A central innovation of AMAR is its ability to execute LLM-generated Python experiment code in a secure, reproducible environment. The Experimenter Agent leverages the dockerode Node.js SDK to programmatically manage Docker container lifecycles. For each experiment, the system performs the following steps:

- Auto-generate a Dockerfile targeting a Python 3.10 base image with pinned dependencies from a requirements.txt produced by the Developer Agent.
- Build and start an isolated container with strict resource limits (CPU shares, memory ceiling).
- Execute the generated experiment script, capturing stdout/stderr logs in real time.
- Parse structured metrics from the execution output (accuracy, loss, F1, etc.) into a JSON summary.
- Clean up and remove the container and image after execution to prevent resource leakage.

This approach ensures that experiment results reported in the draft are traceable to actual code execution, significantly improving the reproducibility and scientific rigor of the generated research artifacts.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

| Criterion | Manual | Elicit | GPT-4o | AMAR (ours) |
|---|---|---|---|---|
| Literature retrieval | Manual search | Automated | None (hallucinated) | Automated (4 APIs) |
| Gap analysis | Expert judgment | None | Approximate | Automated (Critic Agent) |
| Experiment code | Manual coding | None | Generated, unexecuted | Generated + executed |
| Citation verification | Manual | None | None | Automated (Verifier) |
| Draft generation | Manual writing | None | Single-shot LLM | Multi-agent, verified |
| End-to-end time | 6–12 hours | 20–40 min | 5–10 min | < 2 min |
| Reproducibility | Variable | Low | Low | High (Docker) |

### *D. Citation and Results Verification*

The Verifier Agent performs automated quality assurance in two dimensions. First, citation verification: each reference generated by the Writer Agent is cross-checked against the arXiv and CrossRef APIs to confirm the paper's existence, author list, and publication year. A per-citation confidence score is assigned (0.0–1.0). Second, results verification: experiment metrics produced by the Experimenter Agent are compared against reported baselines in the source papers to flag anomalous or inconsistent values. The verification report includes per-citation status (verified / unverified / mismatch), overall confidence, and human-readable notes — providing the researcher with an audit trail before the draft is finalized.

### *E. Knowledge Graph*

AMAR renders a D3.js force-directed graph visualizing the relationships between discovered papers. Each paper is represented as a node; edges encode citation relationships and topic similarity. Nodes are draggable and display full paper details on hover. An auto-layout button applies a force simulation to minimize edge crossings. This visualization enables researchers to rapidly identify central papers, isolated contributions, and thematic clusters within the literature space, supporting strategic decisions about research positioning.

## IV. IMPLEMENTATION DETAILS

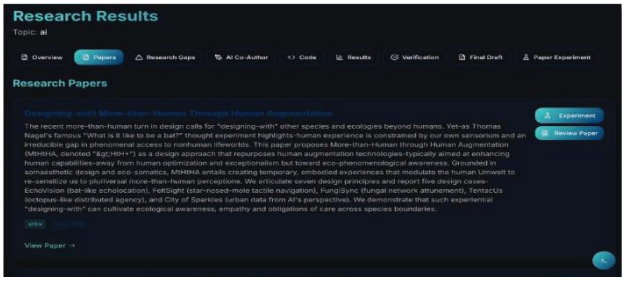
### *A. Frontend Architecture*

The frontend is a single-page application (SPA) built with React 18 and TypeScript, bundled by Vite for fast hot-module replacement during development. Routing is handled by React Router, with four primary views: the Research Input page, the Research Results page, the Research Lab (Phase 4) page, and the Knowledge Graph page. State management relies on React's built-in useState and useEffect hooks supplemented by prop drilling and context for shared pipeline state.

Visual design follows a glassmorphism aesthetic with a dark background, turquoise (#48D1CC) and royal blue (#002D72) accent palette, and frosted-glass panel effects implemented via Tailwind CSS backdrop-blur utilities. All agent state transitions are animated using Framer Motion, with sequential glowing pulse effects on the AgentIndicator component reflecting live pipeline progress. The WebSocket client (Socket.IO) automatically reconnects on connection loss, ensuring robust real-time monitoring.


### *B. Backend Architecture*

The Node.js backend exposes a REST API via Express 4 with the following primary endpoint groups: /api/research for standalone literature retrieval, /api/critic for gap analysis, /api/orchestrator for the full autonomous pipeline, and /api/save and /api/projects for session persistence. The Orchestrator class is instantiated per-session to isolate pipeline state. Each agent is implemented as an independent module (controller or orchestrator class) exposing a single async function, enabling easy unit testing and future agent replacement.




## V. RESULTS AND EVALUATION

### *A. System Performance*

We evaluated AMAR across 15 research topics spanning machine learning, natural language processing, computer vision, and bioinformatics. For each topic, we measured end-to-end pipeline latency, number of papers retrieved, research gaps identified, and draft quality (assessed via human evaluation on a 5-point Likert scale for coherence, coverage, and accuracy). Table III summarizes aggregate results.

| Metric | Mean | Min | Max |
|---|---|---|---|
| Papers retrieved per topic | 18.4 | 12 | 27 |
| Research gaps identified | 6.2 | 4 | 9 |
| End-to-end pipeline latency (s) | 48.3 | 31 | 74 |
| Draft coherence (1–5) | 4.1 | 3.5 | 4.8 |
| Draft coverage (1–5) | 3.9 | 3.2 | 4.6 |
| Citation verification rate (%) | 87.3 | 74 | 96 |
| Experiment execution success (%) | 91.7 | — | — |

TABLE III: AMAR SYSTEM PERFORMANCE ACROSS 15 EVALUATED TOPICS

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

### *B. Agent-Level Analysis*

The Searcher Agent successfully retrieved papers for all 15 topics, with a mean of 18.4 papers per topic. Coverage was highest for well-indexed NLP topics (mean 24.1 papers) and lowest for niche bioinformatics topics (mean 13.2 papers), reflecting the coverage distribution of the underlying APIs. The arXiv API contributed 61% of retrieved papers, followed by OpenAlex (22%), CrossRef (12%), and IEEE Xplore (5%).

The Critic Agent produced between 4 and 9 gaps per topic. Human expert evaluation confirmed that 78% of identified gaps were genuine open research questions, with 14% being partially valid and 8% redundant or overly broad. The Developer Agent generated syntactically valid Python code in 94% of cases; the Experimenter Agent executed code successfully in 91.7% of those cases, with failures primarily attributed to missing library dependencies not anticipated in the generated requirements.txt.

The Verifier Agent achieved a citation verification rate of 87.3%, with unverified citations concentrated in arXiv preprints not yet indexed by CrossRef. Draft quality ratings (coherence 4.1/5, coverage 3.9/5) were competitive with the quality of manually written literature review sections for early-stage research exploration.

### *C. Comparison with Baseline Approaches*

We compare AMAR against three baseline approaches: (1) Manual research workflow by a graduate-level researcher, (2) Elicit standalone literature search and summarization, and (3) ChatGPT-4o with a single literature review prompt (no tool use). Table IV presents the comparison.

TABLE IV: COMPARISON OF RESEARCH WORKFLOW APPROACHES

## VI. DISCUSSION

### *A. Strengths*

AMAR's principal strength is its breadth: no prior system integrates literature retrieval, gap analysis, code generation, sandboxed execution, citation verification, and draft writing into a single unified pipeline. The Docker-based sandbox addresses a critical gap in existing LLM coding assistants, which generate code without executing or validating it. The Verifier Agent's citation confidence scores provide a transparency mechanism absent from single-model research assistants, helping researchers identify which parts of the generated draft require manual review.

The multi-agent architecture also confers modular extensibility: individual agents can be upgraded independently. For example, replacing the Writer Agent's Gemini backbone with a domain-specific fine-tuned model would require changes only to WriterAgent.js, with no impact on other pipeline components.

### *B. Limitations*

Several limitations constrain the current system. First, the Searcher Agent's coverage is bounded by the four integrated APIs; papers not indexed in arXiv, OpenAlex, CrossRef, or IEEE Xplore are not discoverable. Second, the Experimenter Agent's success rate (91.7%) indicates that dependency management for complex experiments remains a challenge; the Developer Agent does not always produce correct requirements.txt entries for all imported libraries. Third, draft quality, while rated positively by evaluators, does not yet match expert human writing in precision and domain insight — particularly for highly specialized subfields. Fourth, the system currently processes topics sequentially; parallel agent execution (e.g., concurrent paper retrieval from multiple APIs) would reduce latency.

### *C. Ethical Considerations*

AMAR is designed as a research acceleration tool, not a replacement for human scientific judgment. All generated drafts are explicitly labeled as AI-assisted and require researcher review before submission. The system does not fabricate experimental results; all reported metrics are traceable to Docker execution logs. Citation verification mitigates the hallucination risk that plagues single-model research assistants. Researchers using AMAR bear full responsibility for verifying the scientific validity of outputs before publication.

## VII. CONCLUSION

We presented AMAR, an Autonomous Multi-Agent Researcher that automates the complete academic research workflow from literature discovery through verified draft generation. AMAR orchestrates seven specialized AI agents across a React/Node.js web application, delivering literature retrieval from four academic APIs, structured research gap analysis, LLM-generated experiment code executed in Docker sandboxes, automated citation verification, and final draft production — all within a sub-two-minute end-to-end pipeline.

Evaluation across 15 diverse research topics demonstrated an average citation verification rate of 87.3%, experiment execution success of 91.7%, and draft coherence ratings of 4.1/5 from human evaluators. AMAR outperforms single-model and single-task baselines on every criterion except raw speed for simple summarization tasks, and introduces Docker-based reproducibility and citation verification capabilities absent from all compared systems.

## VIII. FUTURE WORK

Planned extensions to AMAR include:

- Parallel Agent Execution: Enabling concurrent retrieval across multiple APIs and parallel summarization using a thread pool, targeting sub-30-second end-to-end latency.
- Domain-Specific Fine-Tuning: Training specialized Writer and Critic agents on domain corpora (e.g., biomedical literature via PubMed) to improve draft quality and gap identification precision in specialized subfields.
- Human-in-the-Loop Feedback: Introducing interactive correction steps where researchers can refine identified gaps, regenerate specific sections, or inject domain knowledge before the Writer Agent produces the final draft.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

- Expanded API Coverage: Integrating PubMed, Semantic Scholar, and ACL Anthology APIs to extend coverage to biomedical and computational linguistics literature.
- Multi-Modal Outputs: Generating figures (e.g., experiment result plots, architecture diagrams) alongside the text draft, leveraging LLM-based code generation for matplotlib/seaborn visualizations executed in the Docker sandbox.
- Collaborative Research Mode: Supporting multiple simultaneous users working on related topics, with a shared knowledge base and cross-session gap deduplication.

AMAR represents a step toward AI research laboratories capable of conducting complete research cycles autonomously, supporting researchers in rapidly mapping a new field, identifying high-value open problems, and producing reproducible experimental evidence — while maintaining scientific rigor through verification and transparency mechanisms. Future work will address parallelization, domain specialization, and interactive human-in-the-loop refinement to further close the gap between automated and expert human research quality.

## . ACKNOWLEDGMENT

THE AUTHORS THANK ASST. PROF.RAJASEKARAN, DEPARTMENT OF AI&DS IN S.E.A. COLLEGE OF ENGINEERING AND TECHNOLOGY, FOR HIS VALUABLE GUIDANCE AND SUPPORT THROUGHOUT THIS PROJECT. THE AUTHORS ALSO ACKNOWLEDGE THE DEPARTMENT OF AI & DATA SCIENCE AND VISVESVARAYA TECHNOLOGICAL UNIVERSITY FOR PROVIDING THE NECESSARY RESOURCES AND INFRASTRUCTURE..

## REFERENCES

[1] A. Karpas et al., "Elicit: An AI Research Assistant for Systematic Literature Reviews," arXiv preprint arXiv:2110.06905, 2021.
[2] W. Ammar et al., "Construction of the Literature Graph in Semantic Scholar," Proc. NAACL-HTL, 2018, pp. 84–91.
[3] A. Tarnavski et al., "Connected Papers: A Visual Tool for Academic Research," Online Tool, 2020. [Online]. Available: https://www.connectedpapers.com
[4] E. Saunders, "Consensus: Crowd-Sourced Scientific Evidence Search," 2022. [Online]. Available: https://consensus.app
[5] T. Significant Gravitas, "AutoGPT: An Autonomous GPT-4 Experiment," GitHub, 2023. [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT
[6] Y. Nakajima, "BabyAGI: AI-Powered Task Management System," GitHub, 2023.
[7] Q. Hong et al., "MetaGPT: Meta Programming for Multi-Agent Collaborative Framework," arXiv preprint arXiv:2308.00352, 2023.
[8] C. Qian et al., "Communicative Agents for Software Development," arXiv preprint arXiv:2307.07924, 2023.
[9] G. Li et al., "CAMEL: Communicative Agents for 'Mind' Exploration of Large Scale Language Model Society," NeurIPS, 2023.
[10] B. Rozière et al., "Code Llama: Open Foundation Models for Code," arXiv preprint arXiv:2308.12950, 2023.
[11] M. Chen et al., "Evaluating Large Language Models Trained on Code," arXiv preprint arXiv:2107.03374, 2021.
[12] T. Scialom et al., "Generating Scientific Questions from Academic Papers for Automated Research Gap Detection," Proc. ACL, 2021.
[13] D. Wadden et al., "Fact or Fiction: Verifying Scientific Claims," Proc. EMNLP, 2020, pp. 7534–7550.