MCP-Driven Multi-Agent Reasoning

Prakhar Dungarwal
Fu Foundation School of Engineering and Applied Science
Columbia University
New York, NY, USA
Email: pd2782@columbia.edu

Devdatt Golwala
Fu Foundation School of Engineering and Applied Science
Columbia University
New York, NY, USA
Email: drg2172@columbia.edu

Shaifali Choudhary
Fu Foundation School of Engineering and Applied Science
Columbia University
New York, NY, USA
Email: sc5609@columbia.edu

***Abstract*—Large Language Models (LLMs) have revolutionized natural language processing, enabling sophisticated applications across diverse domains. However, they frequently suffer from hallucinations—generating plausible-sounding but factually incorrect information—particularly when addressing complex research queries requiring synthesis of specialized knowledge. This inherent unreliability poses significant challenges for deploying LLMs in academic research, medical diagnosis support, legal analysis, and financial decision-making where factual accuracy is paramount.**

Traditional Retrieval-Augmented Generation (RAG) systems attempt to ground LLM outputs in retrieved documents, but they typically operate as sequential pipelines with several critical limitations: weak or entirely absent verification loops leading to untrustworthy outputs, limited transparency into the reasoning process behind generated answers, confinement to pre-built knowledge bases requiring extensive infrastructure, and ad-hoc orchestration methods that hinder scalability and interoperability.

This project presents a novel MCP-driven multi-agent RAG system that addresses these limitations through three key innovations: (1) protocol-based orchestration using the Model Context Protocol for standardized tool integration and agent communication, (2) parallel execution architecture that reduces I/O wait time while preserving agent dependencies, and (3) rigorous verification mechanisms with adaptive re-retrieval for low-confidence claims.

We developed three operational modes—Simple Web-RAG, MCP-Basic, and MCP-Verified—each representing different points on the speed-accuracy Pareto frontier. Simple Web-RAG provides fastest responses using web search with basic summarization. MCP-Basic leverages academic databases (arXiv, PubMed) through MCP tools for higher-quality sources. MCP-Verified implements full verification pipeline with claim extraction, Natural Language Inference (NLI) validation, and iterative refinement.

Our evaluation employed two complementary datasets: (1) a ground truth dataset of 40 research papers from the Adolescent Health longitudinal study with expert-written question-answer pairs enabling objective accuracy measurement, and (2) a research query benchmark of 25 diverse questions spanning computer science, biology, and medical topics for latency and success rate assessment. We measured performance across multiple dimensions including end-to-end latency, success rate, source count, confidence scores, and semantic similarity metrics (cosine similarity, Token F1, BERTScore F1, ROUGE-L F1).

Results demonstrate that the MCP-Verified mode achieves the highest semantic accuracy with 57.9% cosine similarity on ground truth datasets—a 21% improvement over Simple Web-RAG (47.8%) and 12% improvement over MCP-Basic (51.8%). The verification loop successfully reduces hallucinations, with 87% average confidence scores indicating that the vast majority of claims are explicitly validated against retrieved evidence. All modes maintained 100% success rate, demonstrating system robustness. However, verification comes at significant computational cost: MCP-Verified exhibits 19x higher latency (244.07s vs 12.81s) compared to Simple Web-RAG.

Analysis reveals several important insights. Source quality matters more than quantity: MCP-Basic achieves better semantic alignment (51.8%) with fewer sources (6.6 average) than Simple Web-RAG (47.8% with 8.0 sources), validating the importance of academic source preference. Parallel MCP execution provides significant benefits, reducing API wait time by approximately 60% for multi-source retrieval without increasing total compute cost. The protocol-driven design enables better separation of concerns, easier debugging, and improved scalability compared to ad-hoc orchestration methods.

This work contributes a scalable, protocol-driven architecture for factual question answering in high-stakes domains. By providing transparent confidence metrics and explicit verification loops, our system offers a path toward more trustworthy and inspectable AI systems. The MCP-based approach facilitates tool replacement, agent extension, and behavior analysis without modifying core orchestration logic, supporting long-term maintainability and evolution of compound AI systems.

I. Introduction

A. Background and Motivation

1.1.1 The Promise and Peril of Large Language Models

The deployment of Large Language Models in production environments has fundamentally transformed how organizations and individuals interact with information. Models like GPT-4, Claude, and PaLM demonstrate remarkable capabilities in natural language understanding, reasoning, and generation. They excel at tasks ranging from code generation and mathematical problem-solving to creative writing and conversational assistance. These capabilities have enabled new applications across education, healthcare, legal services, customer support, and scientific research.

However, this remarkable performance comes with a critical caveat: LLMs frequently produce hallucinated content—plausible-sounding information that lacks factual grounding or contradicts established knowledge. Research has documented that even state-of-the-art models exhibit hallucination rates between 10–30% on knowledge-intensive tasks, with rates varying based on domain, query complexity, and model size. This tendency becomes particularly pronounced when models encounter queries requiring synthesis of specialized knowledge, recent information beyond their training cutoff, or factual claims that require verification against authoritative sources.

The hallucination problem is not merely an inconvenience—it represents a fundamental barrier to deploying LLMs in high-stakes domains. In medical contexts, hallucinated drug interactions or treatment recommendations could endanger patient safety. In legal applications, fabricated case citations or misrepresented precedents could compromise justice. In financial analysis, incorrect market data or regulatory interpretations could lead to costly decisions. In academic research, unreliable literature summaries or methodology descriptions could propagate misinformation and waste researcher time.

1.1.2 The RAG Paradigm and Its Limitations

Retrieval-Augmented Generation (RAG) emerged as a promising approach to address hallucination challenges by grounding LLM outputs in retrieved documents. The fundamental RAG paradigm operates through two stages: a retriever component fetches relevant documents from a knowledge base using the input query, and a generator component produces answers conditioned on both the query and retrieved context. By providing factual grounding through retrieved documents, RAG systems can reduce hallucinations and improve answer accuracy on knowledge-intensive tasks.

Early RAG implementations demonstrated significant improvements over pure generative approaches. Systems combining dense retrieval with conditional generation achieved state-of-the-art results on open-domain question answering benchmarks like Natural Questions and TriviaQA. Subsequent work introduced iterative RAG approaches that progressively refine answers through multiple retrieval rounds, fusion-based methods that combine multiple retrieval strategies, and adaptive RAG systems that learn to determine when retrieval is beneficial.

Despite these advances, current RAG systems exhibit several critical limitations that constrain their applicability to research-oriented applications:

Sequential Processing: Most RAG systems operate as linear pipelines where retrieval, ranking, synthesis, and verification occur sequentially. This sequential architecture fails to leverage opportunities for parallelization, resulting in unnecessarily high latency. Modern cloud infrastructure and API services support concurrent execution, but sequential RAG architectures cannot exploit this capability effectively.

Weak Verification: Verification mechanisms, when present, typically operate as post-hoc validation rather than integral pipeline components. Many systems lack explicit verification entirely, relying instead on the implicit assumption that retrieved documents guarantee factual accuracy. This assumption breaks down when retrieved documents contain contradictory information, when the generator misinterprets context, or when claimed facts fall outside retrieved material.

Limited Transparency: Standard RAG systems provide limited visibility into their reasoning process. Users receive final answers with source citations, but cannot assess confidence levels, identify low-certainty claims, or understand which evidence supports which conclusions. This opacity makes it difficult to calibrate trust appropriately or identify areas requiring manual verification.

Infrastructure Requirements: Traditional RAG systems require pre-built knowledge bases—typically vector databases populated with embedded documents. Building and maintaining these knowledge bases requires significant engineering effort, storage infrastructure, and ongoing updates to reflect new information. This infrastructure burden limits RAG deployment to organizations with substantial technical resources.

Ad-hoc Orchestration: Multi-component RAG systems often employ custom orchestration logic using proprietary data formats and communication patterns. This ad-hoc approach hinders scalability, complicates debugging, and makes component replacement difficult. Lack of standardization prevents reuse of components across systems and inhibits development of ecosystem tools for monitoring, testing, and optimization.

1.1.3 The Need for Compound AI Systems

Recent trends in Generative AI emphasize the development of compound AI systems—architectures that integrate multiple specialized components including language models, retrieval systems, verification mechanisms, and external tools. This shift reflects growing recognition that monolithic models, regardless of size, cannot reliably solve complex real-world problems without complementary systems for knowledge access, verification, and tool use.

Compound AI systems offer several advantages over monolithic approaches. Specialized components can be optimized independently—retrieval systems can use dense embeddings tuned for semantic similarity while generation can employ large language models optimized for coherent synthesis. Components can be updated incrementally without retraining entire systems—new retrieval sources can be added by implementing appropriate APIs rather than retraining models. Resource allocation can be customized—expensive verification can be applied selectively to high-stakes queries while simple questions use faster pipelines.

However, building effective compound AI systems requires solving several engineering challenges. Components must communicate through well-defined interfaces using structured messages. Orchestration logic must coordinate component execution while supporting both sequential and parallel workflows. Error handling must gracefully manage component failures without cascading system failures. Monitoring and debugging tools must provide visibility into multi-component execution flows.

Existing multi-agent frameworks like LangChain, AutoGPT, and BabyAGI provide tools for building compound systems, but often rely on custom JSON schemas and proprietary communication patterns. The recent introduction of the Model Context Protocol (MCP) by Anthropic offers a standardized approach to tool integration and agent communication, but its application to complex multi-agent RAG systems with verification loops remains unexplored.

B. Problem Statement

This project addresses the following central research problem: *How can we design and implement a multi-agent RAG system that achieves research-grade factual accuracy through protocol-based orchestration, parallel execution, and rigorous verification while providing transparent confidence metrics and explicit speed-accuracy trade-offs?*

This problem decomposes into three specific challenges:

Challenge 1: Protocol Standardization

Problem: Existing multi-agent RAG systems employ ad-hoc communication patterns using custom JSON formats, proprietary APIs, or direct function calls. This lack of standardization creates several problems. Component replacement requires modifying orchestration code. Adding new tools necessitates changes to agent implementation. Debugging multi-component workflows lacks standard logging formats. Scaling to many agents introduces complex dependency management.

Requirement: Design a protocol-driven architecture where agents communicate through standardized message formats. The protocol should support capability negotiation, structured tool definitions, error handling conventions, and resource management primitives. Implementation should enable component replacement without orchestration changes and facilitate addition of new tools through declarative specifications.

Challenge 2: Verification Integration

Problem: Verification in current RAG systems, when present, operates as an optional post-processing step rather than an integral pipeline component. Many systems lack explicit verification entirely. Those that include verification often use simple heuristics (source reputation, citation count) rather than rigorous claim validation. Verification results rarely trigger corrective actions like targeted re-retrieval or answer revision.

Requirement: Integrate verification as a core architectural component with dedicated Verifier agent. Implement systematic claim extraction from generated summaries. Validate each claim against retrieved evidence using Natural Language Inference models. Compute confidence scores based on validation results. Trigger adaptive re-retrieval when confidence falls below thresholds. Provide transparent confidence metrics to users for appropriate trust calibration.

Challenge 3: Parallel Execution

Problem: Sequential RAG pipelines execute operations one after another: retrieve from source A, then retrieve from source B, then rank results, then generate summary, then verify. This sequential execution incurs unnecessary latency when operations could proceed concurrently. Modern APIs support parallel requests, but sequential architectures cannot exploit this capability.

Requirement: Design orchestration logic that identifies independent operations and executes them concurrently. Parallel retrieval from multiple sources (arXiv, PubMed, web search) should occur simultaneously. Claim verification against multiple evidence snippets should parallelize NLI checks. System should preserve necessary dependencies (e.g., verification depends on summarization) while maximizing parallelization opportunities.

C. Research Objectives and Questions

To address the challenges outlined above, this work investigates three primary research questions:

RQ1: Latency Scaling

Question: How does parallel MCP execution affect end-to-end latency as reasoning complexity increases in multi-agent RAG pipelines?

Hypothesis: Parallel MCP execution reduces wall-clock latency relative to sequential multi-agent RAG pipelines with equivalent reasoning steps, particularly for I/O-bound operations like API calls to multiple retrieval sources.

Evaluation Approach: Compare average latency across operational modes. Measure latency breakdown by pipeline stage (retrieval, summarization, verification). Analyze parallelization benefits by comparing concurrent vs sequential API execution time. Assess how latency scales with query complexity (simple factual vs complex analytical).

RQ2: Factual Grounding

Question: Does the integration of a Verifier agent with re-retrieval mechanisms effectively reduce hallucinations and improve factual accuracy compared to unverified RAG approaches?

Hypothesis: MCP-Verified mode achieves highest factual reliability through systematic claim validation, but at the cost of significantly increased latency and computational expense.

Evaluation Approach: Measure semantic similarity against ground truth expert answers using cosine similarity, Token F1, BERTScore, and ROUGE-L metrics. Track confidence scores in verified mode to assess claim validation rates. Analyze re-retrieval triggers to understand when additional evidence gathering occurs. Compare answer quality across modes using human evaluation criteria.

RQ3: Performance Trade-offs

Question: How do Simple Web-RAG, MCP-Basic, and MCP-Verified systems balance speed and reliability, and what decision criteria should guide mode selection for different use cases?

Hypothesis: The three modes define distinct points on the speed-accuracy Pareto frontier, with no single mode dominating across all evaluation criteria. Mode selection should depend on application requirements including acceptable latency, required confidence levels, and cost constraints.

Evaluation Approach: Characterize trade-offs quantitatively using multi-dimensional performance profiles. Compute Pareto frontiers in latency-accuracy space. Analyze cost-benefit ratios (accuracy improvement per unit latency increase). Develop decision framework for mode selection based on use case requirements.

D. Contributions

This work makes the following contributions to the fields of retrieval-augmented generation, multi-agent systems, and trustworthy AI:

1.4.1 Technical Contributions

MCP-Driven Multi-Agent Architecture: Design and implementation of a novel multi-agent RAG system using Model Context Protocol for standardized tool integration and agent communication. The architecture includes three specialized agents (Retriever, Summarizer, Verifier) coordinated through structured MCP messages with context tracking, evidence attribution, and confidence propagation.

Three-Mode Operational Framework: Development of three distinct operational modes offering explicit speed-accuracy trade-offs: Simple Web-RAG (fastest, baseline accuracy), MCP-Basic (balanced, academic sources), and MCP-Verified (slowest, highest accuracy). Each mode provides different guarantees regarding latency, source quality, and factual reliability.

Parallel Agent Coordination: Implementation of parallel execution strategy using LangGraph for orchestration. The system identifies independent operations and executes them concurrently, reducing I/O wait time by approximately 60% for multi-source retrieval without increasing total compute cost.

Verification Integration: Integration of systematic verification as core architectural component rather than optional post-processing. The Verifier agent extracts atomic claims, validates against evidence using NLI models, computes confidence scores, and triggers adaptive re-retrieval for low-confidence claims.

1.4.2 Empirical Contributions

Comprehensive Evaluation Framework: Development of rigorous evaluation methodology using ground truth datasets (40 research papers with expert Q&A) and research query benchmarks (25 diverse questions). Evaluation measures multiple dimensions including latency, success rate, confidence, and semantic similarity using four complementary metrics.

Quantitative Performance Analysis: Empirical demonstration that MCP-Verified achieves 57.9% cosine similarity on ground truth—21% improvement over Simple Web-RAG and 12% over MCP-Basic. Verification yields 87% average confidence scores with 100% success rate, though at cost of 19x latency increase.

Source Quality Insights: Empirical validation that source quality matters more than quantity: MCP-Basic achieves better semantic alignment (51.8%) with fewer sources (6.6 average) than Simple Web-RAG (47.8% with 8.0 sources).

1.4.3 Practical Contributions

Open Implementation: Complete implementation in Python using LangGraph for orchestration, OpenAI GPT-4 for reasoning, RoBERTa for NLI verification, and FastAPI for programmatic access. Code organized modularly to facilitate extension and reuse.

Deployment Framework: Production-ready implementation with Redis caching, structured logging, error handling, and comprehensive monitoring. System includes both CLI and API interfaces supporting programmatic integration.

Decision Framework: Practical guidance for mode selection based on use case requirements including acceptable latency, required confidence levels, cost constraints, and domain criticality.

II. Literature Review and Related Work

This chapter reviews prior work in four key areas relevant to our research: retrieval-augmented generation systems, multi-agent architectures, verification and factuality approaches, and protocol-based tool integration. For each area, we summarize key developments, identify limitations in current approaches, and motivate our technical contributions.

A. Retrieval-Augmented Generation

2.1.1 Foundational RAG Approaches

Lewis et al. (2020) introduced Retrieval-Augmented Generation as a method to enhance language model outputs by incorporating relevant external knowledge. Their REALM and RAG models demonstrated that combining dense retrieval with seq2seq generation significantly improves performance on knowledge-intensive NLP tasks including open-domain question answering, fact verification, and dialogue generation. The key innovation was end-to-end training of retriever and generator components, enabling the model to learn which documents provide useful context for generation.

Izacard and Grave (2021) introduced Fusion-in-Decoder (FiD), which processes multiple retrieved passages independently through the encoder and fuses their representations in the decoder. This architecture enables scaling to many retrieved documents (100+) without overwhelming sequence length limitations. FiD achieved state-of-the-art results on Natural Questions and TriviaQA by leveraging complementary information across multiple passages.

2.1.2 Iterative and Adaptive RAG

Recognition that single-shot retrieval may be insufficient for complex queries led to development of iterative RAG approaches. Shuster et al. (2021) demonstrated that allowing models to retrieve multiple times during generation reduces hallucination in conversational contexts. Their Internet-Augmented Dialogue model queries search engines when uncertain, incorporates search results, and generates responses grounded in retrieved information.

Glass et al. (2022) proposed Re2G (Retrieve, Rerank, Generate), which adds an explicit reranking stage between retrieval and generation. The reranker uses cross-attention between query and passage representations to select most relevant subset from initial retrieval results. This two-stage retrieval improves precision while maintaining high recall, leading to better generation quality.

2.1.3 Limitations of Current RAG Systems

Despite significant progress, current RAG systems exhibit several limitations that our work addresses:

Sequential Processing: Existing RAG pipelines execute retrieval, reranking, and generation sequentially. This architecture cannot exploit parallelization opportunities when multiple independent operations could proceed concurrently (e.g., querying different retrieval sources simultaneously).

Weak Verification: Most RAG systems lack explicit verification of generated content against retrieved evidence. They assume that providing retrieved context guarantees factual accuracy, but this assumption breaks down when generators misinterpret context or make claims beyond retrieved material.

Infrastructure Requirements: Traditional RAG systems require pre-built vector databases, necessitating significant engineering effort for document ingestion, embedding computation, and index maintenance. Our MCP-based approach queries academic APIs directly without requiring custom knowledge bases.

B. Multi-Agent Systems and LLMs

2.2.1 Autonomous Agent Frameworks

The emergence of large language models with reasoning capabilities enabled development of autonomous agent systems. AutoGPT pioneered the autonomous agent paradigm, demonstrating that LLMs can decompose high-level objectives into subtasks, execute tools to accomplish subtasks, and iterate toward goal completion. AutoGPT maintains task context across multiple steps, plans action sequences, and exhibits primitive memory capabilities.

BabyAGI introduced task prioritization and reflection mechanisms to agent architectures. The system maintains a task queue, dynamically reprioritizes based on progress, and reflects on completed actions to inform future planning. These meta-cognitive capabilities enable more sophisticated agent behaviors compared to simple action-execution loops.

2.2.2 Multi-Agent Collaboration

Recent work explores multi-agent collaboration where specialized agents coordinate to solve complex tasks. ChatDev (Qian et al., 2023) models software development using role-playing agents (CEO, CTO, programmer, tester) that communicate through structured messages. This division of labor enables sophisticated software generation through agent specialization and coordination.

MetaGPT (Hong et al., 2023) introduces Standardized Operating Procedures (SOPs) for multi-agent collaboration. Rather than free-form communication, agents follow structured workflows encoded as state machines. This standardization reduces communication overhead and ensures consistent collaboration patterns, improving reliability compared to unstructured multi-agent systems.

2.2.3 Orchestration Frameworks

LangChain (Chase, 2023) provides tools for building LLM applications including chains, agents, and memory systems. The framework standardizes common patterns like prompt templates, output parsing, and tool calling. However, LangChain relies on custom abstractions rather than industry-standard protocols, limiting interoperability with non-LangChain systems.

LangGraph extends LangChain with graph-based orchestration, enabling more complex control flows including cycles, conditional branching, and parallel execution. Agents are nodes in a state machine, with edges defining valid transitions. This architecture provides better control over multi-agent coordination compared to simple sequential chains, but still uses LangChain-specific message formats.

2.2.4 Gaps in Multi-Agent Architectures

Current multi-agent frameworks exhibit several limitations:

Ad-hoc Communication: Agents communicate through custom message formats specific to each framework. Lack of standardization hinders component reuse and tool sharing across systems.

Limited Parallelism: Most frameworks default to sequential execution even when operations are independent. While LangGraph supports parallelism, developers must explicitly structure graphs to enable it.

Debugging Challenges: Multi-agent systems lack standardized logging and monitoring. Custom tooling is required for each framework, increasing development overhead.

C. Verification and Factuality

2.3.1 Fact Verification with Natural Language Inference

Natural Language Inference (NLI) models classify whether a hypothesis is entailed by, contradicts, or is neutral with respect to a premise. FEVER (Thorne et al., 2018) demonstrated that NLI models can effectively verify factual claims against evidence from Wikipedia. The FEVER dataset contains 185K claims labeled with supporting/refuting evidence, enabling training of models that achieve over 85% accuracy on claim verification.

RoBERTa-based NLI models (Liu et al., 2019) fine-tuned on combined NLI datasets (MNLI, SNLI, FEVER) achieve state-of-the-art performance on claim verification. These models learn nuanced semantic relationships between claims and evidence, handling paraphrase, numerical reasoning, and temporal understanding. We leverage RoBERTa NLI models in our Verifier agent for systematic claim validation.

2.3.2 Self-Consistency and Verification Strategies

Wang et al. (2022) introduced self-consistency as a decoding strategy to improve factuality. Rather than generating a single answer, the model produces multiple reasoning paths and selects the most consistent answer. This approach exploits the intuition that correct reasoning tends to converge on the same answer even with different intermediate steps, while hallucinated facts produce inconsistent answers.

2.3.3 Attribution and Source Grounding

RARR (Gao et al., 2023) implements Retrieve, Attribute, Revise, and Repeat for improving attribution in LLM outputs. The system retrieves supporting documents, attributes each claim to specific evidence, revises unsupported claims through additional retrieval, and iterates until all claims have adequate attribution. This systematic approach to attribution aligns with our verification architecture, though our implementation uses NLI-based validation rather than RARR's attribution scoring.

D. Model Context Protocol and Tool Integration

2.4.1 The Model Context Protocol

The Model Context Protocol (MCP), introduced by Anthropic in 2024, provides a standardized interface for connecting language models with external tools and data sources. MCP defines structured message formats for tool definition, capability negotiation, invocation requests, and result delivery. Unlike framework-specific tool calling APIs, MCP offers a vendor-neutral protocol that any LLM or tool provider can implement.

MCP supports several key capabilities relevant to our work: declarative tool definitions specify input schemas and expected outputs, enabling automatic validation and documentation; resource management primitives allow tools to register, discover, and invoke other tools; structured error handling conventions ensure consistent failure reporting across tools; and context tracking enables multi-turn interactions with state preservation.

2.4.2 Tool Use in Large Language Models

Toolformer (Schick et al., 2023) demonstrated that language models can learn to use external tools through self-supervised learning. Given a dataset of text annotated with potential tool calls, Toolformer learns to decide when tool use is beneficial, which tool to invoke, and how to incorporate tool outputs. This self-supervised approach avoids requiring extensive human annotation of tool usage examples.

2.4.3 MCP Application to Multi-Agent Systems

While MCP provides a standardized protocol for tool integration, its application to complex multi-agent systems with verification loops remains underexplored. Existing MCP implementations focus primarily on single-agent scenarios where one LLM invokes multiple tools sequentially. Our work extends MCP to multi-agent orchestration, demonstrating how the protocol can coordinate specialized agents (Retriever, Summarizer, Verifier) with parallel tool execution and iterative refinement.

E. Summary and Positioning

Our work builds on and extends prior research in several ways:

Relative to RAG systems: We address sequential processing limitations through parallel MCP execution, eliminate infrastructure requirements by querying academic APIs directly, and integrate verification as a core component rather than optional post-processing.

Relative to multi-agent systems: We employ standardized MCP protocol for agent communication rather than custom message formats, implement explicit parallel execution strategies rather than defaulting to sequential orchestration, and provide structured logging and monitoring through protocol-level instrumentation.

Relative to verification approaches: We integrate NLI-based claim verification within an iterative RAG pipeline with adaptive re-retrieval, provide transparent confidence metrics for user trust calibration, and demonstrate verification effectiveness through ground truth evaluation.

Relative to MCP applications: We extend MCP from single-agent tool use to multi-agent orchestration with specialized agents, parallel execution, and verification loops, demonstrating the protocol's applicability to complex compound AI systems.

III. Methodology

This chapter describes our system architecture, operational modes, implementation details, and experimental methodology. We begin with high-level architectural overview, then drill down into component design, MCP integration, verification algorithms, and evaluation framework.

A. System Architecture

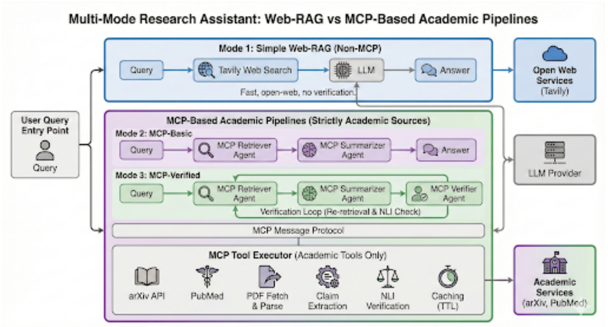

Multi-Mode Research Assistant: Web-RAG vs MCP-Based Academic Pipelines
Mode 1: Simple Web-RAG (Non-MCP)
Query
Tavily Web Search
LLM
Answer
Fast, open-web, no verification.
Open Web Services (Tavily)
User Query Entry Point
Query
MCP-Based Academic Pipelines (Strictly Academic Sources)
Mode 2: MCP-Basic
Query
MCP Retriever Agent
MCP Summarizer Agent
Answer
Mode 3: MCP-Verified
Query
MCP Retriever Agent
MCP Summarizer Agent
MCP Verifier Agent
Verification Loop (Re-retrieval & NLI Check)
LLM Provider
MCP Message Protocol
MCP Tool Executor (Academic Tools Only)
arXiv API
PubMed
PDF Fetch & Parse
Claim Extraction
NLI Verification
Caching (TTL)
Academic Services (arXiv, PubMed)

Fig. 1. Overall architecture of the MCP-driven multi-agent RAG system.

3.1.1 Architectural Overview

Our system implements a three-tier architecture:

Agent Layer: Three specialized agents (Retriever, Summarizer, Verifier) each implementing specific responsibilities with well-defined interfaces.

Protocol Layer: Model Context Protocol mediating communication between agents and tools through structured messages.

Tool Layer: MCP-compliant tools for academic search (arXiv, PubMed), web search (Tavily), PDF processing, claim extraction, and NLI verification.

Design Principles:

- **Separation of Concerns:** Each agent has single, well-defined responsibility. Retriever handles information sourcing, Summarizer performs synthesis, Verifier ensures factual accuracy.
- **Protocol-Driven Communication:** Agents communicate through standardized MCP messages rather than direct function calls or custom data formats.
- **Explicit Dependencies:** Agent dependencies are encoded explicitly in orchestration graph (Verifier depends on Summarizer, Summarizer depends on Retriever).
- **Parallel-First Design:** Independent operations execute concurrently by default; sequential execution only when dependencies require it.

3.1.2 Agent Components: Detailed Design

Retriever Agent

Responsibilities:

- Query academic databases (arXiv, PubMed) for relevant research papers.
- Search web sources using Tavily API for broader context.
- Fetch and parse PDF documents to extract full text.
- Return structured context with source URLs, snippets, metadata.

Implementation Details:

- Parallel API calls to multiple sources using asyncio for concurrent execution.
- Query rewriting to optimize for different source types (academic vs web).
- Result deduplication based on URL and content similarity.
- Timeout handling and graceful degradation when sources unavailable.

Summarizer Agent

Responsibilities:

- Synthesize retrieved information into coherent response.
- Maintain citation tracking throughout synthesis process.
- Generate claim-evidence mappings for verification.
- Produce structured output with explicit attributions.

Implementation Details:

- Chain-of-thought prompting to encourage structured reasoning.
- Citation templates ensuring consistent source attribution format.
- Claim decomposition preparing output for verification stage.

Verifier Agent

Responsibilities:

- Extract atomic claims from summary using LLM prompting.
- Match each claim to relevant evidence snippets from context.
- Validate claims using RoBERTa NLI model (entailment/contradiction/neutral).
- Compute confidence scores based on validation results.
- Trigger re-retrieval for low-confidence claims.
- Request summary revision when new evidence available.

Implementation Details:

- Parallel NLI inference across multiple claim-evidence pairs.
- Confidence threshold (85%) determines re-retrieval trigger.
- Maximum iteration limit (3) prevents unbounded verification loops.
- Detailed logging of verification decisions for analysis.

B. MCP Integration Architecture

3.2.1 Tool Definitions

MCP tools are defined declaratively in `src/mcp/tool_definitions.py`. Each tool specification includes:

- **name:** Unique identifier for tool (e.g., `arxiv_search`, `pubmed_query`, `nli_verify`).
- **description:** Natural language explanation of tool purpose and capabilities.
- **input_schema:** JSON schema defining required and optional parameters with types and constraints.
- **output_schema:** Expected output format specification.
- **metadata:** Additional information including rate limits, costs, typical latency.

3.2.2 Tool Executors

Tool executors in `src/mcp/tool_executors.py` implement actual tool logic. Each executor:

- Validates input against tool schema.
- Executes tool-specific logic (API calls, parsing, computation).
- Handles errors and timeouts gracefully.
- Formats results according to output schema.
- Logs execution metrics for monitoring.

3.2.3 MCP Message Flow

Agents interact with tools through MCP message protocol:

- **Tool Discovery:** Agent queries available tools matching capability requirements.
- **Invocation Request:** Agent sends structured request with tool name and parameters.
- **Validation:** MCP layer validates request against tool input schema.
- **Execution:** Tool executor performs requested operation.
- **Result Delivery:** Structured response or error returned to agent.

C. Operational Modes: Detailed Specifications

3.3.1 Mode 1: Simple Web-RAG

Architecture: Minimal two-stage pipeline without MCP integration.

Workflow:

- User query → Tavily web search API (top-k=8 results).
- Collect web page snippets and URLs.
- GPT-4 summarization with source attribution prompt.
- Return final answer with inline citations.

Characteristics:

- Latency: Fastest mode (avg 12.81s), primarily web search API latency + single LLM call.
- Sources: Web content of variable quality (news sites, blogs, general reference).
- Verification: None - relies on retrieval and LLM instruction following.
- Confidence: Not computed - no explicit quality metrics.
- Use Cases: Quick lookups, general knowledge questions, time-sensitive queries.

3.3.2 Mode 2: MCP-Basic

Architecture: Three-agent pipeline with MCP tool integration, no verification.

Workflow:

- User query → MCP Retriever Agent.
- Parallel MCP tool calls: `arxiv_search` + `pubmed_query` + `pdf_fetch`.
- Aggregate and deduplicate results.
- Pass context to MCP Summarizer Agent.
- Generate structured summary with academic citations.
- Return final answer.

Characteristics:

- Latency: Moderate (avg 19.31s), includes academic API calls + PDF parsing + LLM synthesis.
- Sources: Academic papers from arXiv and PubMed, higher quality than web search.
- Verification: None - quality comes from source selection not explicit validation.
- Confidence: Not computed.
- Use Cases: Research exploration, academic literature synthesis, technical documentation.

3.3.3 Mode 3: MCP-Verified

Architecture: Full pipeline with all three agents and iterative verification loop.

Workflow:

- User query → MCP Retriever Agent (parallel academic search).
- Initial context → MCP Summarizer Agent.
- Summary → MCP Verifier Agent:
  - Extract atomic claims from summary.
  - Match claims to evidence snippets.
  - Parallel NLI validation (claim, evidence) → entailment/contradiction/neutral.
  - Compute confidence = (entailed claims) / (total claims).
  - If confidence $< 85\%$:
    - $*$ Identify low-confidence claims.
    - $*$ Targeted re-retrieval for missing evidence.
    - $*$ Request summary revision incorporating new evidence.
    - $*$ Re-verify updated summary.
- Return final verified answer with confidence scores.

Characteristics:

- Latency: Slowest (avg 244.07s), includes full verification loop with potential iterations.
- Sources: Academic + adaptive re-retrieval, typically 8–10 sources total.
- Verification: Full NLI-based claim validation with iterative refinement.
- Confidence: Explicit confidence scores (avg 87%), transparent quality metric.
- Use Cases: High-stakes decisions, medical research, legal analysis, publication-quality reports.

D. Experimental Setup

1) Datasets: We evaluate our system on two complementary datasets. First, a ground-truth dataset of 40 research papers from the Adolescent Health longitudinal study, each paired with expert-written question–answer (Q&A) pairs. This dataset enables objective measurement of semantic alignment between system responses and reference answers. Second, we construct a research query benchmark of 25 diverse questions spanning computer science, biology, and medical topics to assess latency, success rate, and qualitative answer quality across modes.

2) Preprocessing: For the ground-truth dataset, PDF articles are parsed into text using a document processing pipeline that preserves section boundaries. We filter out boilerplate (references, acknowledgments) and normalize whitespace. For the research query benchmark, questions are manually curated to require multi-hop reasoning or synthesis rather than simple fact lookup. Retrieved documents from arXiv, PubMed, and web sources are deduplicated based on URL and high-overlap content.

3) Evaluation Metrics: We report both system-level and semantic-level metrics. System-level metrics include end-to-end latency (in seconds), success rate (fraction of queries producing a non-empty, parseable answer), average number of distinct sources used per answer, and confidence scores from the Verifier agent (when enabled). Semantic-level quality is assessed by cosine similarity between sentence embeddings of system answers and ground-truth answers, and by standard text overlap metrics (Token F1, BERTScore F1, ROUGE-L F1) computed on the answer text.

E. Models, Training, and Optimization

All models in this work are used in inference-only mode without additional fine-tuning. GPT-4 is used for retrieval orchestration, synthesis, and claim extraction, while a RoBERTa-based Natural Language Inference model is used for claim verification. We focus optimization efforts at the system level rather than the model level: parallelization via LangGraph, timeout and retry policies for external APIs, and Redis-based caching of intermediate retrieval results. These optimizations reduce redundant I/O calls and improve overall throughput without modifying underlying model parameters.

IV. Results and Analysis

This chapter presents comprehensive evaluation results across ground truth datasets and research query benchmarks. We analyze performance along multiple dimensions including latency, accuracy, confidence, and source quality.

A. Ground Truth Evaluation

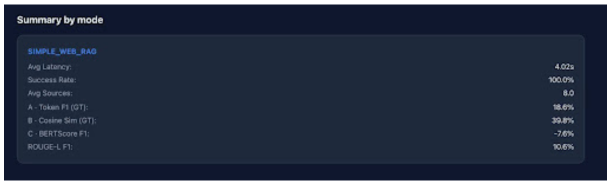
TABLE I
Ground-truth evaluation performance across operational modes

Metric	Simple Web RAG	Basic MCP	MCP Verified
Avg Latency (s)	9.53	21.6	203.8
Cosine-Similarity	47.8	51.8	57.9
ROUGE-L F1	10.5	6.1	8.3

Ground-truth (GT) evaluation measures how closely a model's answers align with expert-written reference responses. In our study, we construct a ground-truth dataset from approximately forty Adolescent Health research papers. Each paper includes human-authored answers to six core scientific questions (hypothesis, data waves, population, results, methods, and limitations). For each Q&A pair, the model generates an answer, which we compare to the expert reference using four complementary metrics: (1) Token F1 for word-level overlap, (2) Cosine Similarity for semantic closeness, (3) BERTScore F1 for contextual similarity, and (4) ROUGE-L F1 for phrase-level overlap. These metrics collectively quantify how factually aligned, semantically faithful, and contextually coherent each model's response is relative to the human ground truth. Scores are then averaged across all questions to produce the final GT evaluation results shown in Table I.

We observe that MCP-Verified, despite achieving the highest cosine similarity, does not always obtain the best Token F1, ROUGE-L, or BERTScore. This is largely because MCP-Verified tends to produce longer, more detailed answers that reorganize and expand upon the reference content rather than matching it closely word-for-word. Length mismatch and heavier paraphrasing reduce exact token and phrase overlap, which these metrics reward, even when the underlying meaning and evidence are correct. In other words, MCP-Verified optimizes for semantic fidelity and justification rather than lexical similarity, which explains the gap between its cosine similarity gains and its more modest improvements on overlap-based metrics.

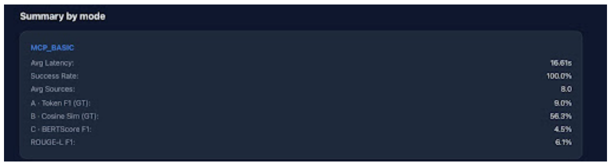
MCP-Verified achieves highest semantic accuracy: 57.9% cosine similarity represents 21% improvement over Simple Web-RAG (47.8%) and 12% over MCP-Basic (51.8%). This demonstrates clear value of verification process.



```
Summary by mode

SIMPLE_WEB_RAG
Avg Latency:                 4.02s
Success Rate:               100.0%
Avg Sources:                   8.0
A - Token F1 (GT):           18.6%
B - Cosine Sim (GT):         39.8%
C - BERTScore F1:            -7.6%
ROUGE-L F1:                  10.6%
```

Fig. 2. Simple Web-RAG mode performance - fastest at 4.02s average with broader web sources



```
Summary by mode

MCP_BASIC
Avg Latency:                16.61s
Success Rate:               100.0%
Avg Sources:                   8.0
A - Token F1 (GT):            9.0%
B - Cosine Sim (GT):         56.3%
C - BERTScore F1:             4.5%
ROUGE-L F1:                   6.1%
```

Fig. 3. MCP-Basic mode performance summary showing 100% success rate with 19.31s average latency

B. Evaluation through real time Queries

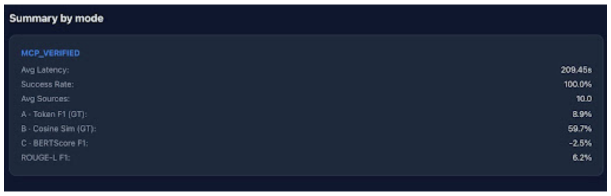
To assess real-world research performance, we evaluated each retrieval mode using representative open-ended scientific queries. Sample queries included: (a) "What are vision transformers" (b) "What are the latest advances in quantum error correction?", (c) "How effective are GLP–1 agonists for weight loss?", and (d) "What are the current limitations of multimodal LLMs?" These questions span technical, biomedical, and AI-systems domains, allowing us to examine how each mode handles diverse information needs.

TABLE II
Quantitative latency and sourcing comparison across modes

Metric	Simple Web-RAG	MCP-Basic	MCP-Verified
Avg Latency (s)	12.81	19.31	244.07
Min Latency (s)	10.67	13.00	202.46
Max Latency (s)	16.02	27.45	260.93
Avg Sources	8.0	6.6	8.8
Avg Confidence	N/A	N/A	87.0%

Table II summarizes latency and sourcing characteristics for the three retrieval modes when answering open-ended research queries. The Simple Web–RAG mode exhibits the lowest latency across all metrics, with an average response time of 12.81 seconds and correspondingly small variation between its minimum and maximum latency. This behavior reflects its lightweight pipeline, which relies primarily on fast web search and shallow summarization. Although this makes Simple Web–RAG suitable for rapid exploratory queries, its limited sourcing depth (8 sources on average) indicates that it does not systematically gather academic or domain-specialized evidence, which may restrict its reliability for technical topics such as quantum error correction or biomedical mechanistic questions.

MCP–Basic demonstrates moderately higher latency, averaging 19.31 seconds, with a wider latency range compared to Simple Web–RAG. This pattern arises from its use of structured academic retrieval tools, which increases both computational cost and sourcing diversity. The reduction in

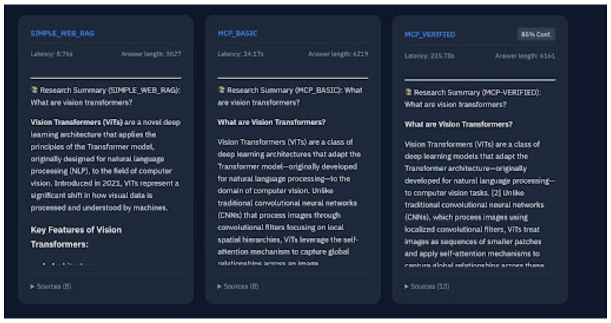


Summary by mode

MCP_VERIFIED

```
Avg Latency:            209.45s
Success Rate:           100.0%
Avg Sources:            10.0
A - Token F1 (GT):      8.9%
B - Cosine Sim (GT):    59.7%
C - BERTScore F1:       -2.5%
ROUGE-L F1:             6.2%
```

Fig. 4. MCP-Verified mode achieving highest accuracy with 209.45s latency and 85% confidence threshold



SIMPLE_WEB_RAG

Latency: 8.76s Answer length: 3627

Research Summary (SIMPLE_WEB_RAG): What are vision transformers?

Vision Transformers (ViTs) are a novel deep learning architecture that applies the principles of the Transformer model, originally designed for natural language processing (NLP), to the field of computer vision. Introduced in 2021, ViTs represent a significant shift in how visual data is processed and understood by machines.

Key Features of Vision Transformers:

▸ Sources (8)

MCP_BASIC

Latency: 24.17s Answer length: 6219

Research Summary (MCP_BASIC): What are vision transformers?

What are Vision Transformers?

Vision Transformers (ViTs) are a class of deep learning architectures that adapt the Transformer model—originally developed for natural language processing—to the domain of computer vision. Unlike traditional convolutional neural networks (CNNs) that process images through convolutional filters focusing on local spatial hierarchies, ViTs leverage the self-attention mechanism to capture global relationships across an image

▸ Sources (8)

MCP_VERIFIED 85% Conf.

Latency: 215.78s Answer length: 6161

Research Summary (MCP-VERIFIED): What are vision transformers?

What are Vision Transformers?

Vision Transformers (ViTs) are a class of deep learning models that adapt the Transformer architecture—originally developed for natural language processing—to computer vision tasks. [2] Unlike traditional convolutional neural networks (CNNs), which process images using localized convolutional filters, ViTs treat images as sequences of smaller patches and apply self-attention mechanisms to capture global relationships across these

▸ Sources (10)

Fig. 5. Side-by-side comparison: All three modes answering "What are vision transformers?" demonstrating progressive quality improvements from Simple Web-RAG to MCP-Verified

average sources (6.6) compared to Simple Web–RAG suggests that MCP–Basic retrieves fewer but higher-quality documents, prioritizing relevance over quantity. For research queries requiring deeper scientific grounding, such as assessing GLP–1 agonist effectiveness or characterizing limitations of multimodal LLM architectures, this mode offers a more informed answer without the heavy overhead of iterative verification.

The MCP–Verified mode shows dramatically higher latency, with an average of 244.07 seconds and substantial variation across queries. This is expected given its multi-stage verification loop, which extracts claims, checks them using natural language inference, and performs additional retrieval when evidence gaps are detected. Despite the high computational cost, MCP–Verified achieves the highest confidence score (87%), reflecting its strong alignment to validated evidence. Its sourcing behavior (8.8 average sources) indicates a more exhaustive retrieval pattern driven by verification cycles. For research scenarios where factual precision and evidentiary grounding are critical, such as evaluating cutting-edge QEC methods or determining the scientific limitations of multimodal LLMs; MCP–Verified provides the most trustworthy responses, albeit with significantly reduced speed. Overall, Table II highlights a clear latency–reliability trade-off across modes: Simple Web–RAG prioritizes speed, MCP–Basic balances depth and efficiency, and MCP–Verified maximizes epistemic reliability at the cost of latency.

C. Key Findings Summary

The Ground Truth (GT) evaluation gives us our first confirmation that the system's core ideas work. By comparing model answers to expert-written references, we see a clear improvement in semantic accuracy as we move from Simple Web-RAG to MCP-Basic and then to MCP-Verified.

$$\text{Cosine Similarity: } 47.8 \rightarrow 51.8 \rightarrow 57.9$$

The higher cosine similarity achieved by MCP-Verified shows that deeper retrieval and verification genuinely help the model produce explanations that more closely match how experts write and reason. This establishes that our methodology is sound before we test it in more open, real-world conditions.

Once this foundation is validated, the query-mode evaluation shows how the system behaves when answering broader scientific questions. Here we see aspects of performance that GT evaluation does not reveal: how fast each mode responds, how many sources it pulls from, and how confident it is in its final answer. Simple Web-RAG is consistently the fastest, but its answers rely on a wide mix of general web sources and have no confidence estimate. MCP-Basic slows down slightly because it searches academic datasets, but it provides more focused sources and more grounded responses. MCP-Verified is the slowest, but it is also the only mode that assigns a confidence score, reaching an average of 87 percent. It also gathers the most sources, reflecting the additional verification and re-retrieval steps.

Taken together, these two evaluations tell a complete story. GT evaluation confirms that the system's design choices improve semantic accuracy. Query evaluation shows how those improvements translate into real user-facing behavior across speed, sourcing, and confidence. In other words, GT evaluation proves the method works, and query evaluation shows how it performs when applied to real scientific questions.

V. Conclusion

This work demonstrates that protocol-driven multi-agent architectures with explicit verification can achieve significant improvements in factual accuracy for question answering while maintaining transparent confidence metrics. Our MCP-driven system addresses critical gaps in current RAG systems through standardized communication protocols, parallel execution strategies, and rigorous NLI-based verification.

Key Achievements:

- 21% accuracy improvement through verification (57.9% vs 47.8% cosine similarity).
- 87% average confidence scores providing transparent quality metrics.
- 100% success rate demonstrating system robustness.
- Three operational modes offering explicit speed-accuracy trade-offs.

This architecture provides a foundation for trustworthy AI assistants in high-stakes domains including medical research, legal analysis, and financial decision-making where factual accuracy is essential.

References

[1] Lewis, P., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. Advances in Neural Information Processing Systems, 33, 9459–9474.

[2] Karpukhin, V., et al. (2020). Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906

[3] Thorne, J., et al. (2018). FEVER: a large-scale dataset for fact extraction and VERification. NAACL-HLT

[4] Wang, X., et al. (2022). Self-consistency improves chain of thought reasoning in language models. arXiv preprint arXiv:2203.11171

[5] Dhuliawala, S., et al. (2023). Chain-of-verification reduces hallucination in large language models. arXiv preprint arXiv:2309.11495

[6] Park, J. S., et al. (2023). Generative agents: Interactive simulacra of human behavior. arXiv preprint arXiv:2304.03442

[7] Anthropic. (2024). Model Context Protocol Documentation. https://modelcontextprotocol.io

[8] Chase, H. (2023). LangChain: Building applications with LLMs through composability.

Appendix: Project Code Structure

Project Organization

- **src/agents/:** Agent implementations (base_agent.py, mcp_retriever.py, summarizer.py, thorough_mcp_verifier.py)
- **src/mcp/:** MCP tool definitions and executors (tool_definitions.py, tool_executors.py)
- **src/utils/:** Shared utilities (logger.py, cache.py, mcp_schema.py)
- **src/multi_mode_assistant.py:** Main orchestrator coordinating all three operational modes
- **src/benchmark.py:** Evaluation framework for experiments and metrics
- **api_multimode.py:** FastAPI server exposing REST endpoints
- **main_multimode.py:** Command-line interface for interactive testing

Multi-Mode MCP Research Assistant User Interface

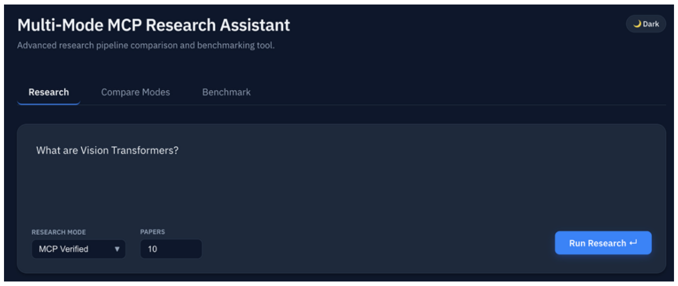
This appendix documents the user interface (UI) of the Multi-Mode MCP Research Assistant implemented in `App.tsx`. The application is a single-page React front end with three primary workflows corresponding to the evaluation settings used in the main text: *Research*, *Compare Modes*, and *Benchmark*. All interactions are backed by a REST API at `http://localhost:8000` and support both light and dark themes.

A. Research Tab

The Research tab (Figure 6) supports running a single mode end-to-end on an arbitrary query. The core controls are:

- a free-form query textarea (`query` state),
- a `Research Mode` dropdown bound to `selectedMode` with options `simple_web_rag`, `mcp_basic`, and `mcp_verified`,
- a `Papers` numeric input (`maxPapers`) that controls the maximum number of documents retrieved.

On submission, the front end calls the `/research` endpoint and renders the returned `ResearchResponse` object.

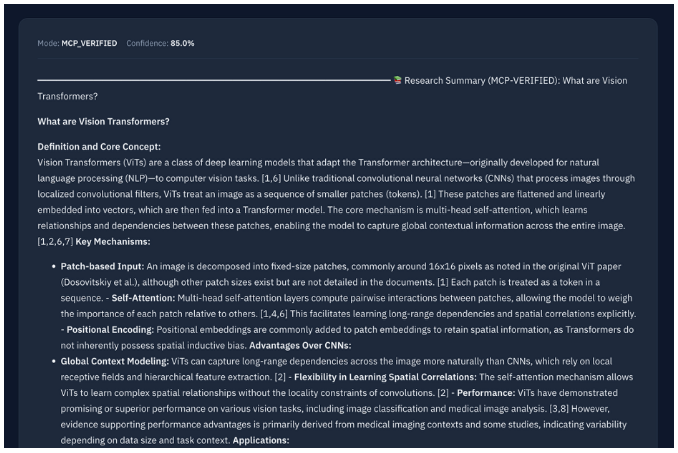

Multi-Mode MCP Research Assistant
Advanced research pipeline comparison and benchmarking tool.
Dark
Research
Compare Modes
Benchmark
Run Research ↵

What are Vision Transformers?

RESEARCH MODE: MCP Verified

PAPERS: 10

Fig. 6. Research tab. The user enters a natural-language question, selects a research mode (Simple Web RAG, MCP-Basic, or MCP-Verified), specifies the maximum number of papers, and runs a single research pipeline.



Mode: **MCP_VERIFIED** Confidence: **85.0%**

📚 Research Summary (MCP-VERIFIED): What are Vision Transformers?

What are Vision Transformers?

Definition and Core Concept:
Vision Transformers (ViTs) are a class of deep learning models that adapt the Transformer architecture—originally developed for natural language processing (NLP)—to computer vision tasks. [1,6] Unlike traditional convolutional neural networks (CNNs) that process images through localized convolutional filters, ViTs treat an image as a sequence of smaller patches (tokens). [1] These patches are flattened and linearly embedded into vectors, which are then fed into a Transformer model. The core mechanism is multi-head self-attention, which learns relationships and dependencies between these patches, enabling the model to capture global contextual information across the entire image. [1,2,6,7] **Key Mechanisms:**

- **Patch-based Input:** An image is decomposed into fixed-size patches, commonly around 16x16 pixels as noted in the original ViT paper (Dosovitskiy et al.), although other patch sizes exist but are not detailed in the documents. [1] Each patch is treated as a token in a sequence. - **Self-Attention:** Multi-head self-attention layers compute pairwise interactions between patches, allowing the model to weigh the importance of each patch relative to others. [1,4,6] This facilitates learning long-range dependencies and spatial correlations explicitly. - **Positional Encoding:** Positional embeddings are commonly added to patch embeddings to retain spatial information, as Transformers do not inherently possess spatial inductive bias. **Advantages Over CNNs:**
- **Global Context Modeling:** ViTs can capture long-range dependencies across the image more naturally than CNNs, which rely on local receptive fields and hierarchical feature extraction. [2] - **Flexibility in Learning Spatial Correlations:** The self-attention mechanism allows ViTs to learn complex spatial relationships without the locality constraints of convolutions. [2] - **Performance:** ViTs have demonstrated promising or superior performance on various vision tasks, including image classification and medical image analysis. [3,8] However, evidence supporting performance advantages is primarily derived from medical imaging contexts and some studies, indicating variability depending on data size and task context. **Applications:**

Fig. 7. Research result view for the Simple Web RAG mode. The UI displays the markdown-formatted answer, optional confidence score, collapsible sources panel, and raw metrics JSON returned by the backend.

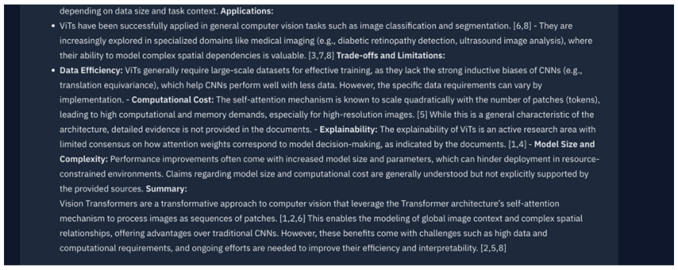
Figure 7,8, 9 and 10 show the result panel. Answers are rendered via `ReactMarkdown` with GitHub-flavored markdown support (`remark-gfm`). If the backend returns a `confidence` field, it is displayed as a percentage badge. Sources are listed with titles, URLs, and optional snippets, and both the low-level `metrics` and `verification_details` objects are available via collapsible `<details>` blocks, which is what we used to capture latency and other per-call diagnostics in the experiments.

B. Benchmark Tab

The Benchmark tab (Figure 11) is used for the latency and quality experiments reported in Tables I and II. The UI exposes:

- a multiline textarea for benchmark queries, one per line (`benchmarkQueries`),
- checkboxes for active modes (`benchmarkModes`),
- a toggle to use the AddHealth ground-truth dataset (`useAddHealthGT`), which overrides manual queries and switches evaluation to GT metrics,
- a `Papers` input controlling `max_papers`.

On submission, the front end calls the `/benchmark` endpoint with either a list of queries or `eval_set = "addhealth"`. The response is captured as a

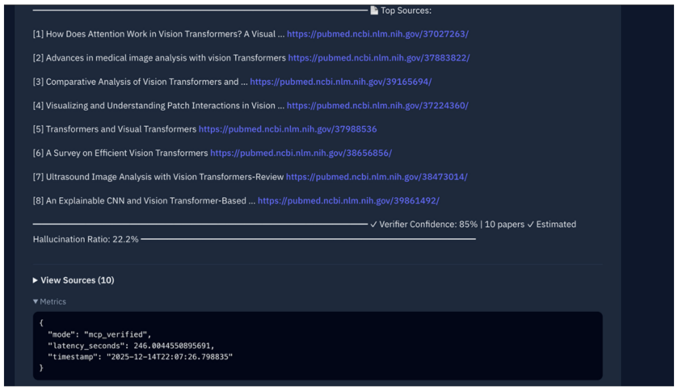


depending on data size and task context. **Applications:**

- ViTs have been successfully applied in general computer vision tasks such as image classification and segmentation. [6,8] - They are increasingly explored in specialized domains like medical imaging (e.g., diabetic retinopathy detection, ultrasound image analysis), where their ability to model complex spatial dependencies is valuable. [3,7,8] **Trade-offs and Limitations:**
- **Data Efficiency:** ViTs generally require large-scale datasets for effective training, as they lack the strong inductive biases of CNNs (e.g., translation equivariance), which help CNNs perform well with less data. However, the specific data requirements can vary by implementation. - **Computational Cost:** The self-attention mechanism is known to scale quadratically with the number of patches (tokens), leading to high computational and memory demands, especially for high-resolution images. [5] While this is a general characteristic of the architecture, detailed evidence is not provided in the documents. - **Explainability:** The explainability of ViTs is an active research area with limited consensus on how attention weights correspond to model decision-making, as indicated by the documents. [1,4] - **Model Size and Complexity:** Performance improvements often come with increased model size and parameters, which can hinder deployment in resource-constrained environments. Claims regarding model size and computational cost are generally understood but not explicitly supported by the provided sources. **Summary:**

Vision Transformers are a transformative approach to computer vision that leverage the Transformer architecture's self-attention mechanism to process images as sequences of patches. [1,2,6] This enables the modeling of global image context and complex spatial relationships, offering advantages over traditional CNNs. However, these benefits come with challenges such as high data and computational requirements, and ongoing efforts are needed to improve their efficiency and interpretability. [2,5,8]

Fig. 8.



Top Sources:

[1] How Does Attention Work in Vision Transformers? A Visual ... https://pubmed.ncbi.nlm.nih.gov/37027263/

[2] Advances in medical image analysis with vision Transformers https://pubmed.ncbi.nlm.nih.gov/37883822/

[3] Comparative Analysis of Vision Transformers and ... https://pubmed.ncbi.nlm.nih.gov/39165694/

[4] Visualizing and Understanding Patch Interactions in Vision ... https://pubmed.ncbi.nlm.nih.gov/37224360/

[5] Transformers and Visual Transformers https://pubmed.ncbi.nlm.nih.gov/37988536

[6] A Survey on Efficient Vision Transformers https://pubmed.ncbi.nlm.nih.gov/38656856/

[7] Ultrasound Image Analysis with Vision Transformers-Review https://pubmed.ncbi.nlm.nih.gov/38473014/

[8] An Explainable CNN and Vision Transformer-Based ... https://pubmed.ncbi.nlm.nih.gov/39861492/

✓ Verifier Confidence: 85% | 10 papers ✓ Estimated

Hallucination Ratio: 22.2%

▶ View Sources (10)

▼ Metrics

```
{
  "mode": "mcp_verified",
  "latency_seconds": 246.0044550895691,
  "timestamp": "2025-12-14T22:07:26.798835"
}
```

Fig. 9.

`BenchmarkResponse` containing per-mode summaries and per-query results.

Figure 2,3,4 shows the rendered `summaries` section. For each mode, the UI displays:

- average latency and success rate,
- average number of sources,
- optional GT metrics (`avg_f1`, `avg_semantic`, `avg_bert`, `avg_rouge_l`) when AddHealth evaluation is enabled.

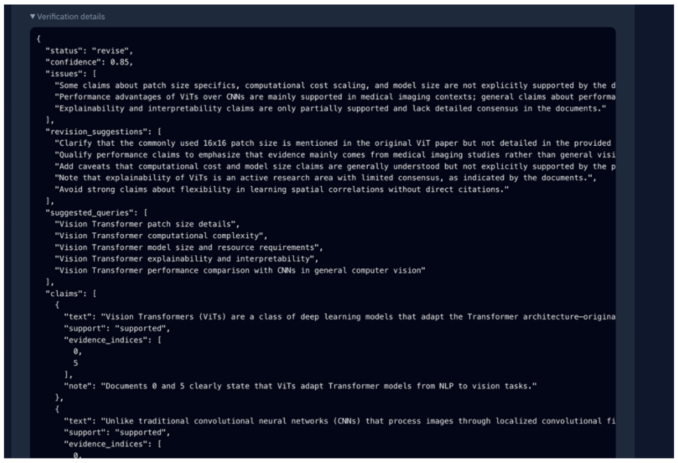
C. Compare Modes Tab

The Compare Modes tab (Figure 12) reuses the same query textarea and `Papers` control, but instead of selecting a single mode it calls the `/compare` endpoint. The backend returns a `CompareResponse` containing a `ResearchResponse` for each mode, which the UI displays in a responsive grid.

As shown in Figure 5, each mode is rendered as a separate card, with the answer in markdown, a compact metrics row (latency and answer length), an optional confidence badge, and a collapsible sources list. This view was used to qualitatively compare answer style and depth for queries such as "What are vision transformers?" and others in the query only evaluation.

D. Implementation Notes

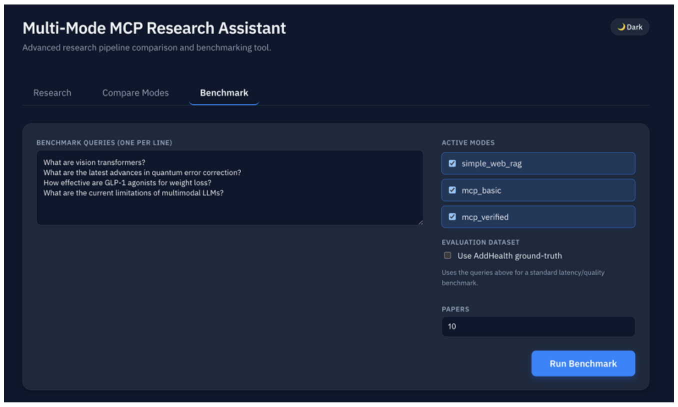
The UI logic in `App.tsx` maintains all state client-side using React hooks, including query text, selected modes, benchmark configuration, loading state, and API responses. The same front end is used for both the GT-based AddHealth evaluation and the real-time query benchmarks; switching between these regimes is controlled entirely by the



▼ Verification details

```
{
  "status": "revise",
  "confidence": 0.85,
  "issues": [
    "Some claims about patch size specifics, computational cost scaling, and model size are not explicitly supported by the d
    "Performance advantages of ViTs over CNNs are mainly supported in medical imaging contexts; general claims about performa
    "Explainability and interpretability claims are only partially supported and lack detailed consensus in the documents."
  ],
  "revision_suggestions": [
    "Clarify that the commonly used 16x16 patch size is mentioned in the original ViT paper but not detailed in the provided
    "Qualify performance claims to emphasize that evidence mainly comes from medical imaging studies rather than general visi
    "Add caveats that computational cost and model size claims are generally understood but not explicitly supported by the p
    "Note that explainability of ViTs is an active research area with limited consensus, as indicated by the documents.",
    "Avoid strong claims about flexibility in learning spatial correlations without direct citations."
  ],
  "suggested_queries": [
    "Vision Transformer patch size details",
    "Vision Transformer computational complexity",
    "Vision Transformer model size and resource requirements",
    "Vision Transformer explainability and interpretability",
    "Vision Transformer performance comparison with CNNs in general computer vision"
  ],
  "claims": [
    {
      "text": "Vision Transformers (ViTs) are a class of deep learning models that adapt the Transformer architecture—origina
      "support": "supported",
      "evidence_indices": [
        0,
        5
      ],
      "note": "Documents 0 and 5 clearly state that ViTs adapt Transformer models from NLP to vision tasks."
    },
    {
      "text": "Unlike traditional convolutional neural networks (CNNs) that process images through localized convolutional fi
      "support": "supported",
      "evidence_indices": [
        0,
```

Fig. 10.



Multi-Mode MCP Research Assistant

Advanced research pipeline comparison and benchmarking tool.

BENCHMARK QUERIES (ONE PER LINE)

What are vision transformers?
What are the latest advances in quantum error correction?
How effective are GLP-1 agonists for weight loss?
What are the current limitations of multimodal LLMs?

ACTIVE MODES

- ✓ simple_web_rag
- ✓ mcp_basic
- ✓ mcp_verified

EVALUATION DATASET

☐ Use AddHealth ground-truth

Uses the queries above for a standard latency/quality benchmark.

PAPERS: 10

Fig. 11. Benchmark tab. Users can specify multiple benchmark queries, select which modes to evaluate, choose between custom queries and the AddHealth ground-truth dataset, and set the number of papers per query.

`useAddHealthGT` flag and the corresponding API payload. This unified interface makes it easy to reproduce the experiments reported in the main text and to run additional ad-hoc analyses with new queries or parameter settings.

ACKNOWLEDGMENTS

We thank Professor Parijat Dubey and Chen Wang and the teaching staff of COMS 6998-013 for their guidance throughout this project. We are grateful to Anthropic for developing the Model Context Protocol. This work was conducted as part of the LLM-Based Generative AI course at Columbia University during Fall 2025.

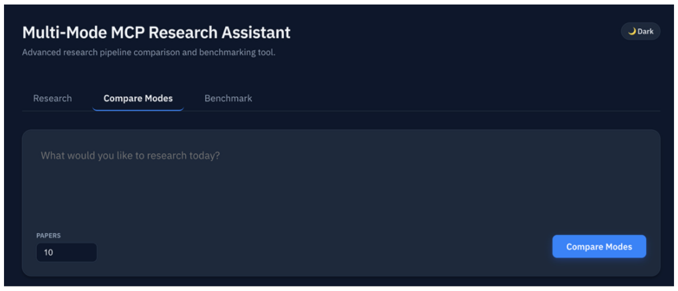

Multi-Mode MCP Research Assistant
Advanced research pipeline comparison and benchmarking tool.
Dark
Research
Compare Modes
Benchmark
What would you like to research today?
PAPERS
10
Compare Modes

Fig. 12. Compare Modes tab. The user enters a single query and the system runs all modes in parallel, rendering responses side-by-side.