Cloud-Based Research Paper Summarization Tool using Amazon Bedrock and OpenSearch

Sindhu Jyoti Dutta, Sheel Patel, Yashavika Singh
Department of Computer Science
New York University, New York, USA
{sd6201, sjp9507, ys6668}@nyu.edu

***Abstract*—The overwhelming pace of academic publishing has created significant barriers to efficient knowledge consumption, particularly in fast-growing disciplines such as computer science, healthcare, and data science. Manual reading and synthesis of large volumes of papers are both time-consuming and prone to oversight. This project presents a cloud-native, serverless application that leverages foundation models via Amazon Bedrock to automate research paper summarization and semantic search. Users can upload research papers, receive concise AI-generated summaries, extract key concepts, and perform similarity-based retrieval using vector embeddings stored in OpenSearch. The backend pipeline is built entirely using AWS services, including Lambda, S3, DynamoDB, and Bedrock, ensuring scalability and cost-efficiency. The frontend offers an intuitive interface for file upload, summary review, keyword-based search, and trending research exploration. By integrating LLM-based summarization with vector search and cloud orchestration, our tool significantly streamlines the academic research process for students, faculty, and professionals alike.**

Keywords— research summarization, Amazon Bedrock, vector search, OpenSearch, large language models, serverless architecture, cloud computing, academic productivity.

I. Problem Statement

With the exponential growth of academic publications across disciplines, researchers are facing an unprecedented information overload. Manually reading, understanding, and comparing papers is time-consuming and inefficient. According to estimates, more than 2.5 million new research articles are published annually across journals and conference proceedings. This has made it increasingly difficult for researchers, graduate students, and industry professionals to stay current with the latest developments.

Traditional approaches to literature review involve manual scanning of titles and abstracts, followed by full-paper readings for promising candidates. This process is not only inefficient but often fails to scale in fast-paced domains like computer science, machine learning, and bioinformatics.

II. Motivation

The need for tools that can automate parts of the literature review process is more critical than ever. Even advanced search platforms like Semantic Scholar and Google Scholar offer limited support for direct summarization or intelligent contextual search beyond keyword matching.

Recent advances in Large Language Models (LLMs) have introduced new possibilities in natural language understanding, enabling automated summarization, question answering, and document clustering with human-comparable accuracy. These capabilities can be harnessed to assist researchers by transforming unstructured papers into structured summaries and semantically indexed vectors, drastically reducing the time needed to consume and compare relevant literature.

Cloud platforms such as Amazon Web Services (AWS) offer robust infrastructure for building scalable, event-driven systems. Services like Amazon Bedrock now provide access to state-of-the-art foundational models including Mistral, Claude, and Titan without requiring users to host or fine-tune models themselves. When combined with serverless compute tools like AWS Lambda, OpenSearch, and DynamoDB, this creates a powerful platform for deploying lightweight, cost-effective machine learning pipelines.

III. Existing Solutions

Several tools and platforms have attempted to address the growing complexity of academic research and paper discovery. While each offers certain advantages, they often fall short in terms of summarization capabilities, extensibility, or integration with modern cloud-native AI services. Below are a few prominent examples:

- **Semantic Scholar:** A free AI-powered academic search engine developed by the Allen Institute for AI. It indexes millions of papers and extracts citations and influence scores but does not offer automated summarization. It relies heavily on metadata and citation networks.
- **Research Rabbit:** A platform designed for visual literature discovery. It allows users to follow citation trails and build collections but does not support AI-generated summaries or keyword-based vector search. Its primary strength lies in network visualizations.
- **Connected Papers:** This tool helps researchers explore prior and derivative work through graph-based clustering. While useful for topic exploration, it lacks deep document-level insights or semantic search features.
- **Scholarcy:** An AI-driven tool that attempts to summarize research papers and generate flashcards. However, it is limited in terms of API access and cannot be integrated seamlessly into academic workflows or scaled using cloud-native infrastructure.
- **Google Scholar and Microsoft Academic Graph:** These search engines provide keyword-based queries and citation tracking but do not incorporate summarization

or semantic embeddings. They are widely used but offer minimal support for automated research synthesis.

While these solutions provide value in discovery and network exploration, they generally do not support end-to-end summarization, vector-based semantic search, or modular backend integration. Our proposed system addresses these limitations by offering a serverless, event-driven pipeline that leverages LLMs through Amazon Bedrock and integrates vector search with OpenSearch.

IV. System Architecture

The architecture of our research paper summarization tool is modular, fully serverless, and designed to scale dynamically based on user load. It integrates various AWS services including Amazon Bedrock, Lambda, S3, DynamoDB, and OpenSearch to deliver a seamless experience from paper upload to semantic search.

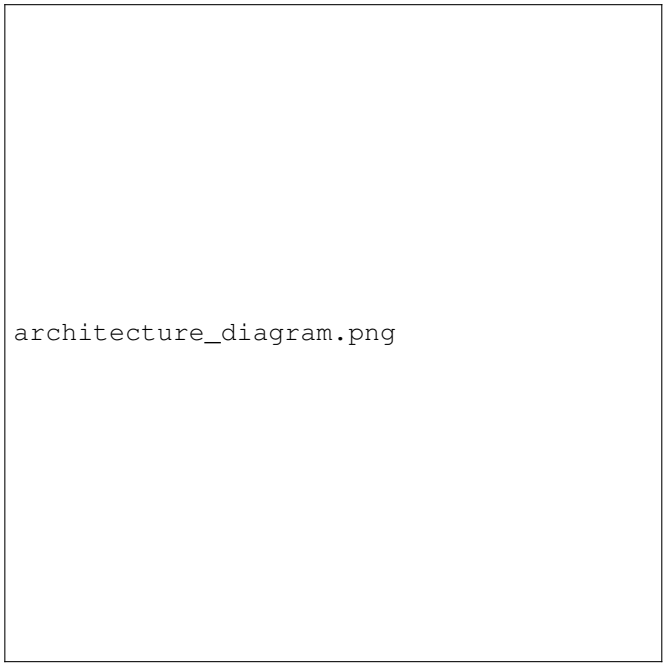

Fig. 1. High-level system architecture showing interactions between frontend, backend, Bedrock, and vector database.

A. Overview of System Components

- **Frontend Interface (Amazon S3):** A static, responsive web application built using HTML, CSS, and JavaScript. Hosted on Amazon S3 with public access enabled, the site allows users to upload files, interact with summaries and perform semantic search. It integrates directly with API Gateway for backend operations and is designed to be device-agnostic for both desktop and mobile accessibility.
- **API Gateway and Upload Handler:** The upload process begins with a call to an API Gateway endpoint. This serves as the entry point to the backend and performs validation, routing, and access control. The call is forwarded to an AWS Lambda function responsible for receiving the uploaded file and storing it in S3. This Lambda function also performs lightweight preprocessing such as file type detection and payload validation.
- **Summarization Pipeline (Lambda + Bedrock):** Once the file is uploaded to S3, it triggers a second Lambda function that reads the document content. The text is parsed and split into chunks of manageable length before being sent to the Mistral model hosted via Amazon Bedrock. The resulting summaries are combined into a cohesive narrative using postprocessing logic. Keywords are concurrently extracted via prompt engineering or a separate keyword extraction model.
- **Metadata Storage (DynamoDB):** The generated summary, extracted keywords, and file metadata (such as filename, upload timestamp, and optional user ID) are written to a DynamoDB table. DynamoDB is chosen for its ability to scale horizontally and store semi-structured data efficiently. Querying this data for summaries or metadata is highly performant due to partition key indexing.
- **Text Embedding Service (Lambda + Bedrock):** A third Lambda function handles the transformation of the extracted keywords and summary into high-dimensional vector embeddings. Using Bedrock's embedding models, the output vectors capture semantic information and are formatted for ingestion into OpenSearch. This embedding pipeline includes normalization and cleaning logic to reduce noise and improve retrieval accuracy.
- **OpenSearch Vector Indexing:** The vector embeddings are indexed in an OpenSearch cluster with k-NN plugin enabled. Each vector is associated with a paper ID and metadata, allowing efficient cosine similarity search during retrieval. Index configuration parameters such as `ef_search`, `M`, and `k` are fine-tuned to optimize recall and response time.
- **Query Interface and Retrieval (API Gateway + Lambda):** When a user submits a query on the frontend, it is passed to another Lambda function which performs real-time embedding using Bedrock. This embedded query is searched against the OpenSearch index, and the top-k results are returned based on similarity. The results are formatted as JSON and rendered client-side.
- **Trending Paper Ingestion (CloudWatch + Lambda):** A background Lambda function is scheduled using CloudWatch Events to fetch trending research papers from arXiv and Semantic Scholar APIs. These papers are summarized and indexed similarly to user-uploaded content, ensuring that the system always has a fresh supply of high-interest literature available for discovery.
- **Question Answering using Retrieval-Augmented Generation (RAG):** To enable deeper interaction with research content, we implemented a lightweight Retrieval-Augmented Generation (RAG) pipeline that allows users to ask questions about a specific paper. When a user submits a question, the system retrieves the relevant summary and keywords from OpenSearch, constructs a

context-aware prompt, and queries an LLM hosted via Amazon Bedrock to generate a response. This enables contextual Q&A grounded in the actual paper content without requiring full-document reprocessing.

- **Text-to-Speech Summarization Playback:** To improve accessibility and enhance user experience, we introduced a text-to-speech (TTS) module that allows users to listen to the summaries. When a summary is generated, the user can trigger a playback function which converts the text into speech using Amazon Polly. The resulting audio is streamed directly in the frontend interface. This feature is especially beneficial for multitasking users or those with visual impairments.

B. Design Choices and Rationale

- **Serverless Compute (AWS Lambda):** Adopting AWS Lambda for all compute tasks ensures that the system scales dynamically and incurs zero cost during idle periods. This makes it ideal for a classroom-scale project or startup prototype. Lambdas are stateless, fast to deploy, and support multiple runtimes.
- **Amazon Bedrock vs. Self-Hosting Models:** Using Bedrock allows the team to bypass model training, deployment, and GPU provisioning. It offers a unified API to access foundation models like Mistral, which are already optimized for latency and cost. This dramatically reduces operational complexity.
- **DynamoDB for Metadata:** Compared to traditional SQL databases, DynamoDB is schema-less and offers low-latency reads and writes. It is resilient to workload spikes and supports on-demand scaling, which is essential for handling unpredictable access patterns.
- **OpenSearch for Vector Search:** OpenSearch provides native support for k-NN vector indexing. This was preferred over external vector DBs like Pinecone or Qdrant due to its tighter integration with AWS IAM and VPCs. The vector index enables semantic similarity without needing keyword matches.
- **S3 Static Website Hosting:** Hosting the frontend on S3 is cost-effective and simplifies deployment. Combined with CloudFront and Route 53 (optionally), it enables fast global delivery.
- **Event-Driven Triggers:** By using S3 and CloudWatch Events as triggers for processing workflows, the architecture avoids polling and minimizes latency. It also makes the system loosely coupled and easier to maintain.
- **Security and Compliance:** The entire pipeline operates over HTTPS. IAM roles restrict access on a per-function basis. No personally identifiable information (PII) is stored. Uploaded documents are never exposed publicly and are retained only for processing.

This architecture was chosen after evaluating multiple alternatives including EC2-based hosting, container orchestration with ECS, and other third-party model APIs. AWS-native tools offered the best balance of flexibility, performance, cost, and ecosystem integration.

C. Implementation Details

The system was implemented using a combination of Python for backend processing and HTML/CSS/JavaScript for the frontend interface. A microservices-inspired architecture is followed where each Lambda function handles a single responsibility, triggered by specific cloud events or API calls. All infrastructure is hosted on AWS and designed using managed services to reduce operational overhead.

1) Backend Services (Lambda Functions): The backend consists of four main Lambda functions:

- **Upload Handler:** Triggered via API Gateway when a user uploads a file. It validates file type, size, and metadata, and stores the document in a specified Amazon S3 bucket using the Boto3 SDK. It returns an upload success response to the frontend.
- **Summarization Lambda:** Triggered by S3 PUT events. It reads the uploaded file, splits the text into 2,000–3,000 character chunks, and sequentially calls the Mistral model hosted on Amazon Bedrock using the `invoke_model()` API. The outputs are aggregated into a single summary and cleaned for fluency. Keywords are also extracted using either few-shot prompting or a secondary call to Bedrock.
- **Embedding Lambda:** Once the summary and keywords are generated, this function uses another Bedrock model to compute a semantic embedding. It uses vector normalization and appends metadata like paper ID, title, and keywords. The data is indexed into Amazon OpenSearch via its k-NN plugin.
- **Search Handler:** Invoked when a user submits a query. It embeds the query string using the same embedding model and performs a nearest-neighbor search on OpenSearch using cosine similarity. The top k results (typically $k = 5$) are returned in descending similarity order.

All Lambda functions are written in Python 3.11 and deployed via AWS Console and the Serverless Application Model (SAM) CLI for consistency. IAM roles are scoped to least privilege, and all Bedrock calls use temporary credentials secured via AWS STS.

2) Frontend Implementation: The frontend is built using a single-page application (SPA) approach and hosted on Amazon S3 with static web hosting enabled. It includes the following components:

- A drag-and-drop file uploader that sends multipart POST requests to API Gateway.
- A real-time status tracker that shows progress updates as the paper is processed.
- Summary display components with expandable sections to minimize cognitive overload.
- A search bar for keyword and semantic query input, linked to the retrieval Lambda.
- UI sections for ”Similar Papers” and ”Trending Research”, both powered by OpenSearch.

JavaScript handles all asynchronous calls to API Gateway using the Fetch API. CORS headers are explicitly configured

on both API Gateway and S3 to allow cross-origin requests.

3) Data Flow and Event Coordination:

1) File upload is initiated on the frontend and POSTed to API Gateway.
2) API Gateway invokes the Upload Lambda, which stores the document in S3.
3) S3 upload event triggers the Summarization Lambda.
4) Summarization Lambda calls Bedrock, processes the output, and stores metadata in DynamoDB.
5) Keywords and summaries are sent to the Embedding Lambda, which computes vector embeddings and indexes them in OpenSearch.
6) When a search query is submitted, the Search Lambda embeds the query and returns ranked results to the frontend.

All components communicate asynchronously via event triggers or REST APIs, making the system loosely coupled and easy to maintain or extend.

4) Additional Modules:

- **Trending Fetcher:** A scheduled Lambda function triggered via Amazon CloudWatch Events, which fetches new papers from public APIs (e.g., arXiv or Semantic Scholar), summarizes them via Bedrock, and indexes them for search.
- **Logging and Monitoring:** All Lambdas push logs to Amazon CloudWatch Logs. Errors and inference latencies are tracked via embedded log statements. Future versions could integrate with AWS X-Ray for trace-level diagnostics.

V. Results

The implementation of the cloud-based research paper summarization tool has been validated through extensive functional testing, user feedback, and demonstration of end-to-end features. This section outlines the primary system capabilities as illustrated through the following screenshots.

A. Upload and Summarization Interface

The homepage provides an intuitive interface for uploading research papers. As shown in Figure 2, users can drag and drop PDF or text files, enter their email address (optional), and receive an AI-generated summary after successful upload and processing.

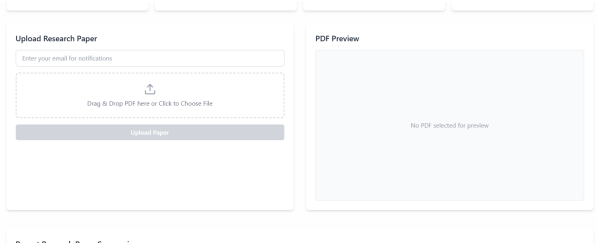

Fig. 2. Homepage interface: User uploads a research paper and receives a summary.

B. Summary Display and Keyword Preview

Once the summary is generated using Amazon Bedrock, the system displays the result under "Recent Research Paper Summaries" (Figure 3). The summary is presented in a card layout along with the original filename, and includes an option for keyword-specific Q&A.

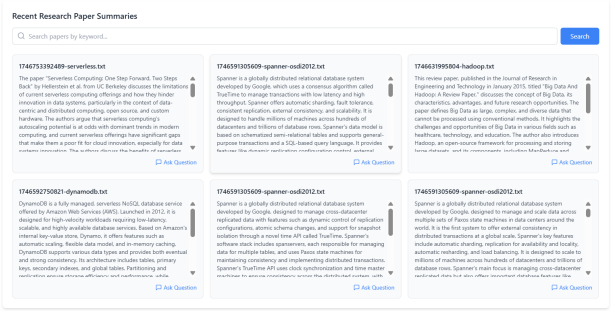

Fig. 3. Summary results: AI-generated summary cards from uploaded papers.

C. Semantic Search and Retrieval

As shown in Figure 4, users can enter a keyword or phrase (e.g., "spanner") to retrieve related papers based on vector similarity search. The embedded query is matched against the OpenSearch index and top results are returned. This enables semantic exploration beyond exact keyword matching.

Fig. 4. Semantic Search: Query-based retrieval using OpenSearch vector similarity.

D. Q&A Interaction

The Q&A module allows users to ask specific questions about a summarized paper, such as "What was the main finding?" or "What methodology was used?" (Figure 5). These responses are generated from extracted keywords and context, demonstrating the system's ability to enable deeper interaction with academic content.

E. Trending Papers Feed

Figure 6 displays the trending papers section, automatically populated via a scheduled Lambda function that fetches recent articles from public APIs like arXiv and Semantic Scholar. These are summarized and embedded in the same pipeline as user uploads, enabling discovery of new research.

Fig. 5. Q&A Interface: Users ask questions related to a paper's summary.

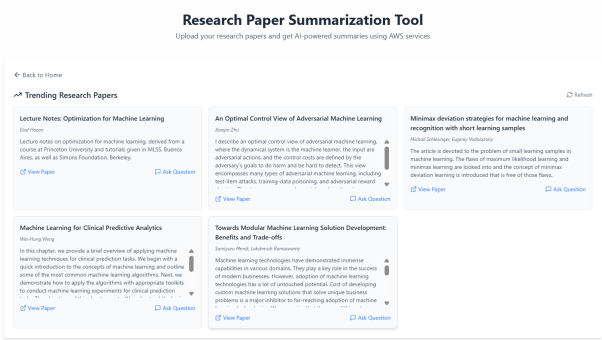

Fig. 6. Trending Feed: Summarized recent papers fetched from external APIs.

F. Summary of Results

The system demonstrates the following core capabilities:

- Seamless file upload and summarization using Bedrock (Mistral model).
- Structured metadata generation including summaries and keyword extraction.
- Semantic search using OpenSearch vector indexing.
- Interactive Q&A with context-aware responses.
- Trending paper ingestion from external academic sources.

These results validate the system's practical utility for students and researchers, showing that AI-powered summarization and retrieval can significantly accelerate literature review and exploration.

VI. Conclusion and Future Work

In this project, we have presented a scalable, serverless cloud-based system that leverages large language models (LLMs) for the automated summarization and semantic retrieval of research papers. By integrating Amazon Bedrock with AWS Lambda, DynamoDB, OpenSearch, and S3, we developed a robust end-to-end pipeline capable of ingesting papers, generating high-quality summaries, extracting keywords, and enabling vector-based similarity search — all through a user-friendly web interface.

Our solution addresses the growing challenge of academic information overload, particularly in fields like computer science and healthcare, where new research is published daily. The tool empowers users to quickly digest the essence of complex papers without manually reading every section, and to discover related works using semantic embeddings rather than traditional keyword search. The use of Bedrock's foundation models — particularly Mistral — provided high-quality summarization results, while OpenSearch's k-NN plugin enabled real-time vector search without leaving the AWS ecosystem.

Beyond the core functionality, our system includes thoughtful features such as a trending paper module powered by CloudWatch Events, a Q&A interaction module powered by a Retrieval-Augmented Generation (RAG)-like approach, and keyword extraction for enhanced indexing. All services were orchestrated using event-driven architectures, which ensured low latency, modularity, and cost-efficiency.

We also introduced a text-to-speech (TTS) feature using Amazon Polly, allowing users to listen to the generated summaries through a simple playback interface. This improves accessibility for users with visual impairments and enhances user experience for mobile or multitasking environments.

There are multiple exciting directions to extend this project:

- **Multi-format Input Support:** Future versions will include native PDF and DOCX ingestion with structure-preserving parsing using libraries like PyMuPDF or AWS Textract.
- **User Authentication and History:** We plan to add authentication via Amazon Cognito, allowing users to track uploaded papers, revisit past summaries, and manage search history.
- **Personalized Summaries:** Integrating prompt-tuning or retrieval-augmented generation (RAG) pipelines to generate summaries tailored to users' background (e.g., undergrad vs PhD).
- **Feedback Loop and Active Learning:** Incorporating thumbs-up/down or "Was this helpful?" feedback on summaries to fine-tune prompt templates and select optimal models over time.
- **Cross-Paper Clustering and Recommendation:** Using the embedding space to build research paper clusters and enable content-based recommendations beyond keyword matches.
- **Mobile-Friendly Experience:** Improving UI/UX responsiveness and implementing push notifications for summary readiness or trending updates.
- **Cost Optimization:** Exploring spot Lambda containers, model caching, and usage quotas for institutional deployment.

In conclusion, this project demonstrates the feasibility and effectiveness of integrating generative AI with cloud-native infrastructure to support academic research workflows. With continuous improvements in foundational models, serverless tools, and vector search capabilities, systems like this can significantly transform how researchers access, consume, and interact with scholarly content.

ACKNOWLEDGMENTS

This project was completed as part of NYU's Cloud Computing curriculum.