DALHOUSIE
UNIVERSITY

CSCI 5411

Advanced Cloud Architecting

Prof. Lu Yang

Mid – Term Project

Milestone – 02

Banner ID: B00988337

Name: Patel Het Ghanshyambhai

Date: 01st June, 2025

Table of Contents –

1. Introduction –

The ScholarSight project, developed for CSCI 5411 Advanced Cloud Architecting at Dalhousie University, is a Retrieval-Augmented Generation (RAG) application designed to assist academic researchers, students, and professionals in processing and querying research papers.

The application is deployed on AWS using Terraform for Infrastructure-as-Code (IaC), with an EC2 instance hosting Docker containers for the Flask application, Ollama (mistral:7b model), and Qdrant vector database. PDFs are stored in S3, metadata in DynamoDB, configurations are managed via Secrets Manager, and monitoring is implemented using CloudWatch.

This report addresses all Milestone-2 requirements, including service configuration, model integration, monitoring, testing, security, cost analysis, and lessons learned. It builds on the architectural and data sequence diagrams provided earlier, focusing on the practical implementation and operational aspects of the system.

2. Final Architecture Design –

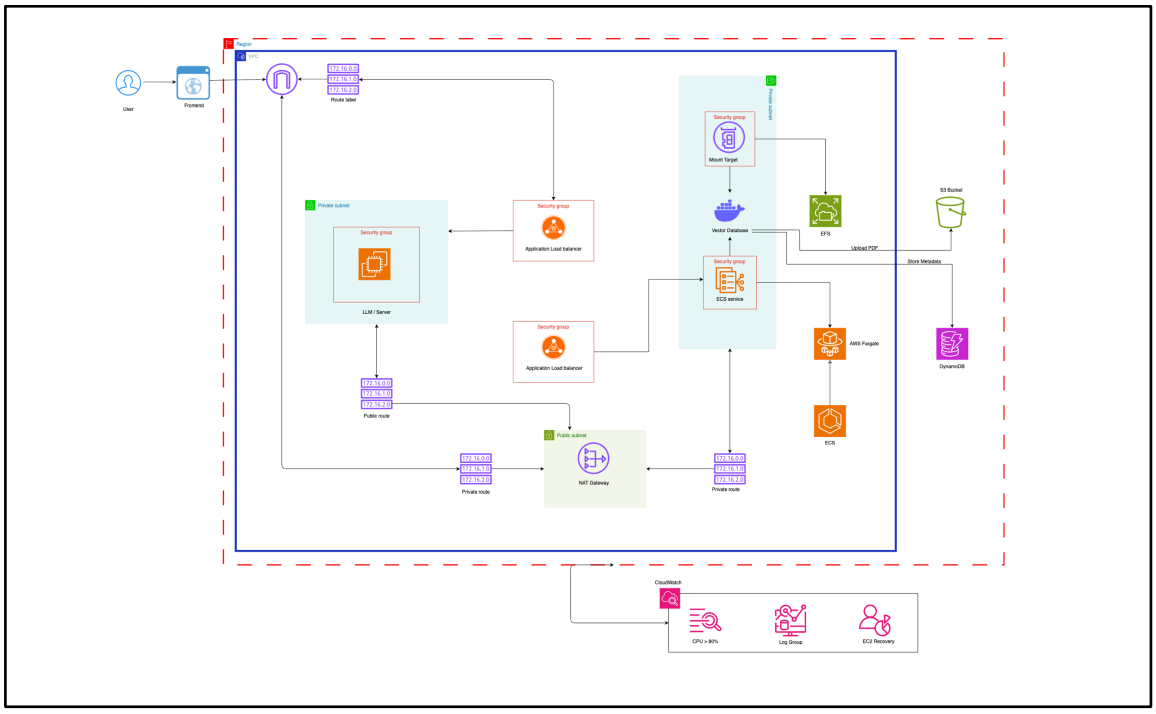

Fig. 1. Final Architecture Diagram

3. Final Data Sequence Diagram –

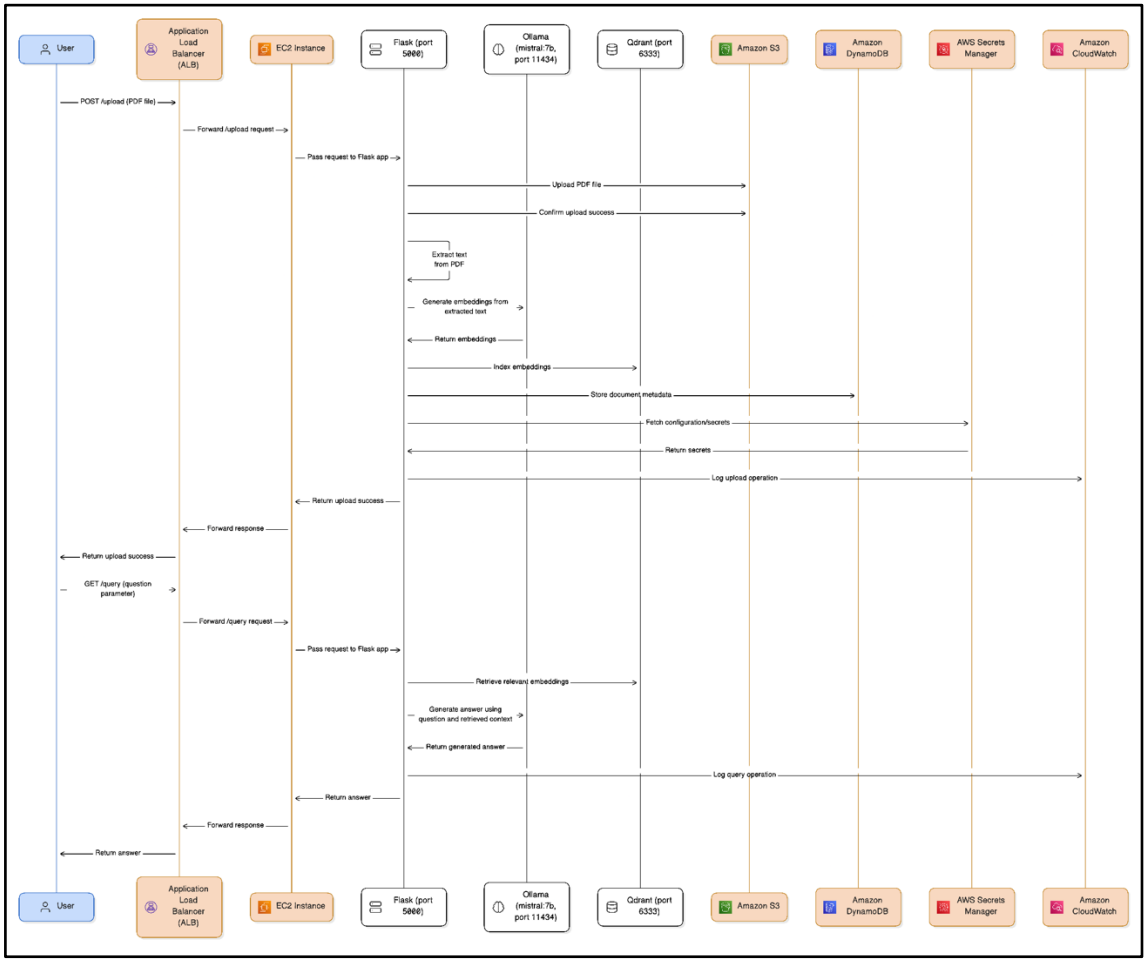

Fig. 2. Final Data Sequence Diagram

4. Service Configuration Details –

The configuration consolidates services into a streamlined setup, with updates reflecting the implemented design. Below is an overview of the key AWS services used:

- **Networking (VPC and Subnets):** The Virtual Private Cloud (VPC) is configured with a CIDR block of 10.0.0.0/16 to create a secure network environment. It includes two public subnets (10.0.1.0/24 in us-east-1a and 10.0.2.0/24 in us-east-1b) for public access and one private subnet (10.0.3.0/24 in us-east-1b, currently unused). An Internet Gateway enables internet access, and a NAT Gateway supports outbound traffic from the private subnet.

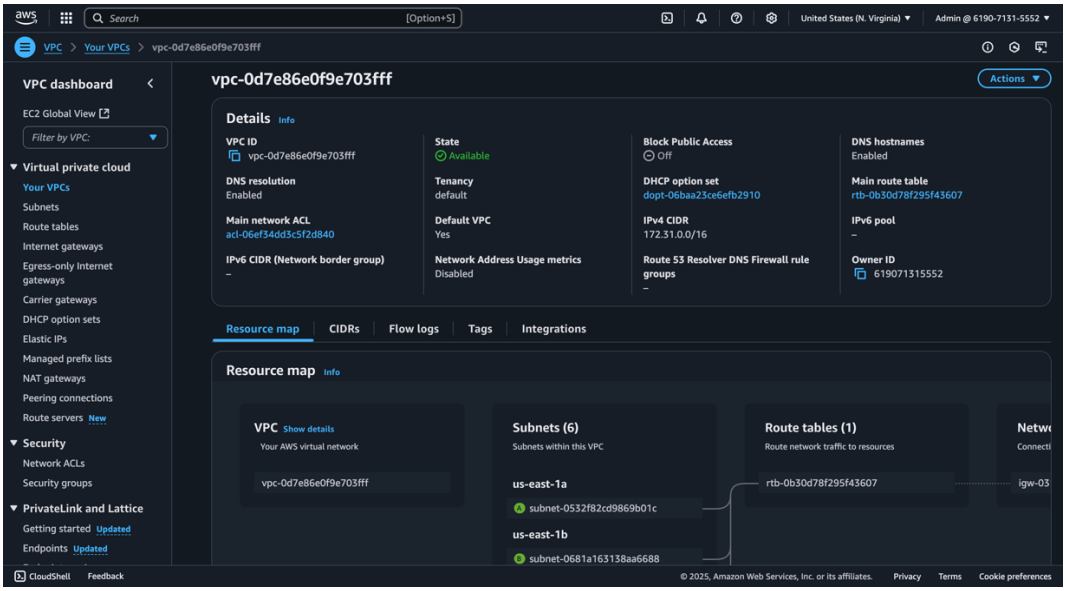

Fig. 3. Virtual Private Cloud

- **EC2 Instance:** A t3.large instance hosts the application, running Docker containers for Flask (port 5000), Ollama with mistral:7b (port 11434), and Qdrant (port 6333). This replaces the dual-EC2 setup from Milestone-1, simplifying the architecture.

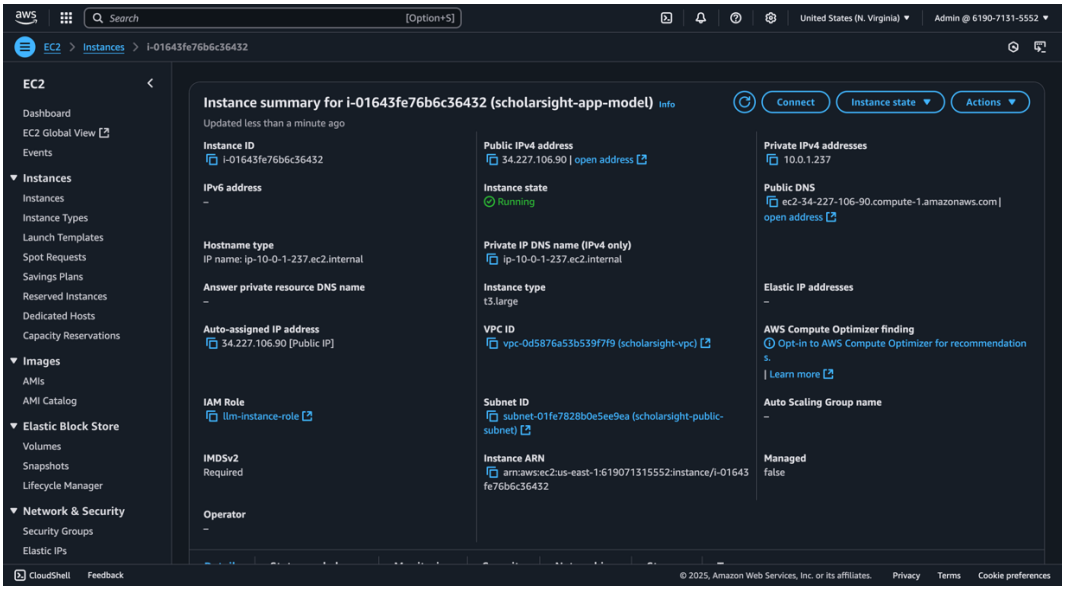

Fig. 4. EC2 instance dashboard

- **Application Load Balancer (ALB):** A public ALB manages incoming HTTP (port 80) and HTTPS (port 443) traffic, forwarding requests to the EC2 instance on port 5000. It uses a self-signed SSL certificate for secure communication.

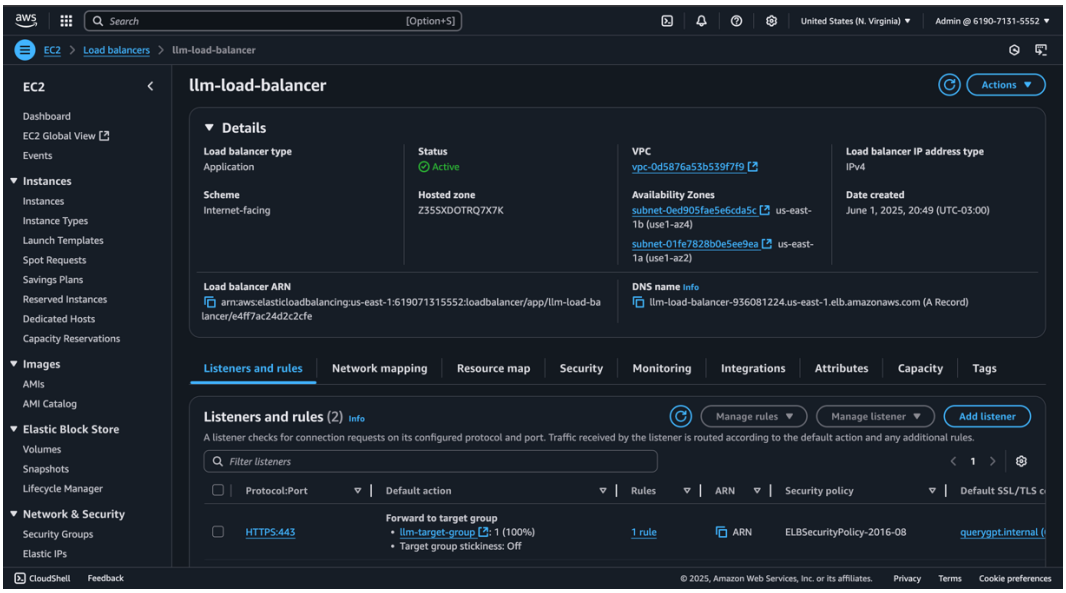

Fig. 5. Application Load Balancer

- **S3 Bucket:** Used for storing uploaded PDF files, configured with AES256 encryption and a 1-day lifecycle rule to manage storage efficiently.

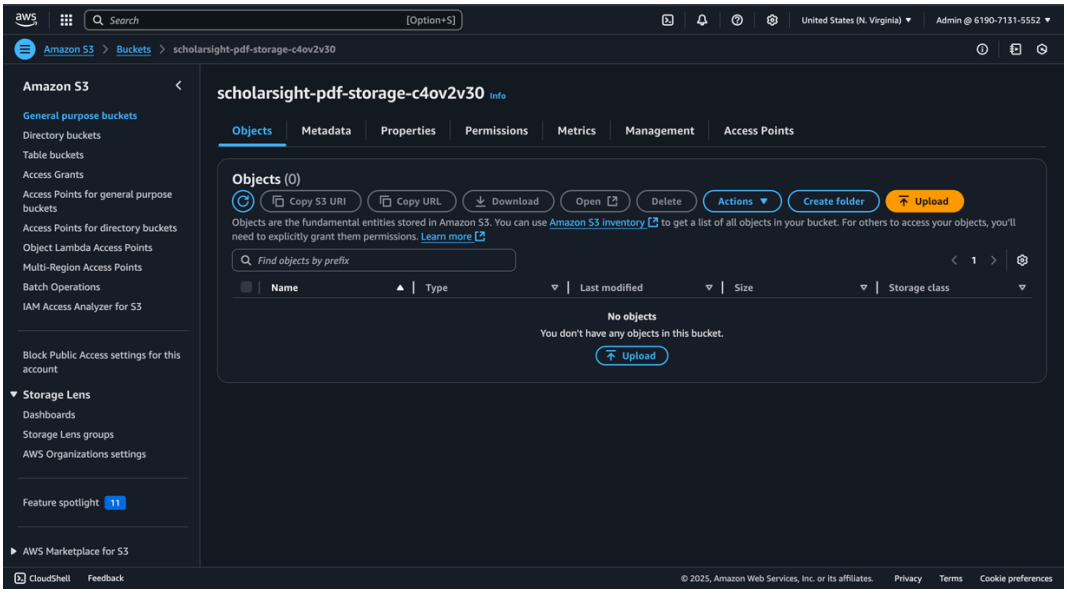

Fig. 6. S3 Bucket Dashboard

- **DynamoDB Table:** Stores metadata with document_id as the hash key, utilizing on-demand billing for flexibility and cost efficiency.

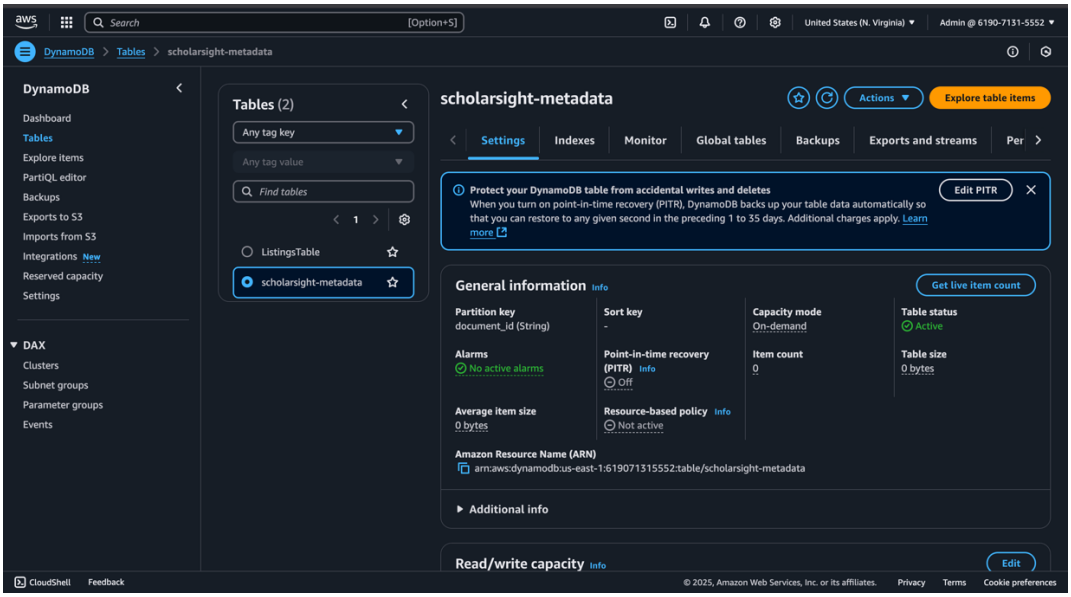

Fig. 7. DynamoDB Dashboard

- **Secrets Manager:** Secures sensitive configurations (e.g., ollama_host, qdrant_host) fetched during EC2 startup.

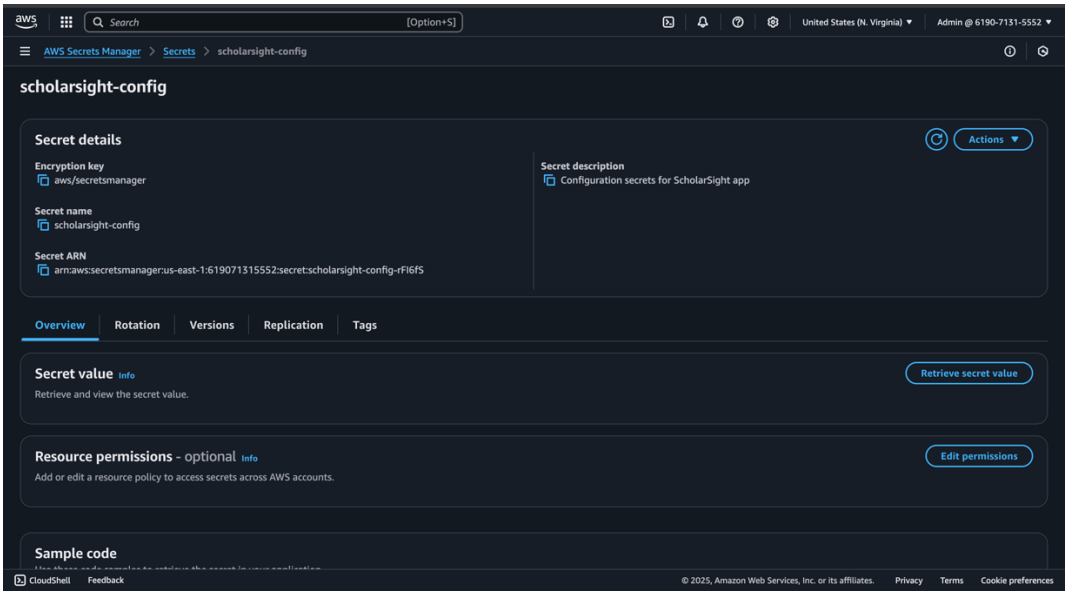

Fig. 8. Secret Manager Dashboard

- **CloudWatch:** Provides logging via a log group (/scholarsight/docker-logs, 7-day retention) and a CPU utilization alarm (>80%) for monitoring.

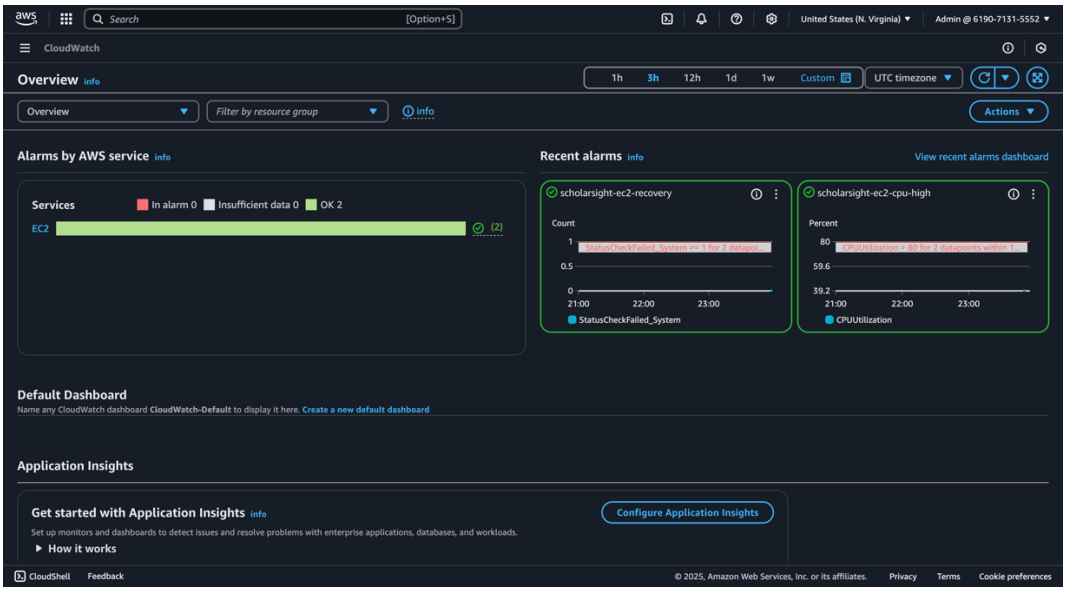

Fig. 9. CloudWatch Dashboard

Improvements from Milestone-1: The architecture shifted from a Lambda-based, dual-EC2 design to a single EC2 with Docker containers, removing the need for a second EC2 and Lambda, enhancing simplicity and cost-effectiveness while maintaining functionality.

5. Model Selection, Integration, and Enhancement –

The model selection process for ScholarSight involved evaluating multiple open-source models to ensure accuracy and precision for the RAG application, building on the Milestone-1 plan to use T5-small or FLAN-T5-small with Sentence-Transformers. Initially, DeepSeek, CodeLlama, and Qwen were tested due to their availability and community support. However, these models provided vague and imprecise answers, failing to meet the academic precision required for research paper queries. DeepSeek struggled with context retention, CodeLlama produced inconsistent outputs, and Qwen lacked specificity in source attribution, leading to their rejection.

Subsequently, Mistral:7b, hosted via Ollama, was selected for its balance of performance, efficiency, and open-source accessibility. This 7-billion parameter model was chosen over the initial T5-small/FLAN-T5-small due to its superior natural language understanding and reduced hallucination, aligning with the RAG focus. No specific training, enhancement, or fine-tuning was performed due to time constraints and the model's pre-trained capabilities, differing from the Milestone-1 plan to optimize with Sentence-Transformers.

Integration Process: Mistral:7b is deployed in a Docker container on the EC2 instance, running on port 11434. The EC2 user data script installs Ollama, pulls the mistral:7b model, and configures it to serve requests. Configurations (e.g., ollama_host) are securely fetched from Secrets Manager during startup, ensuring seamless integration with the Flask application for embedding generation and query answering.

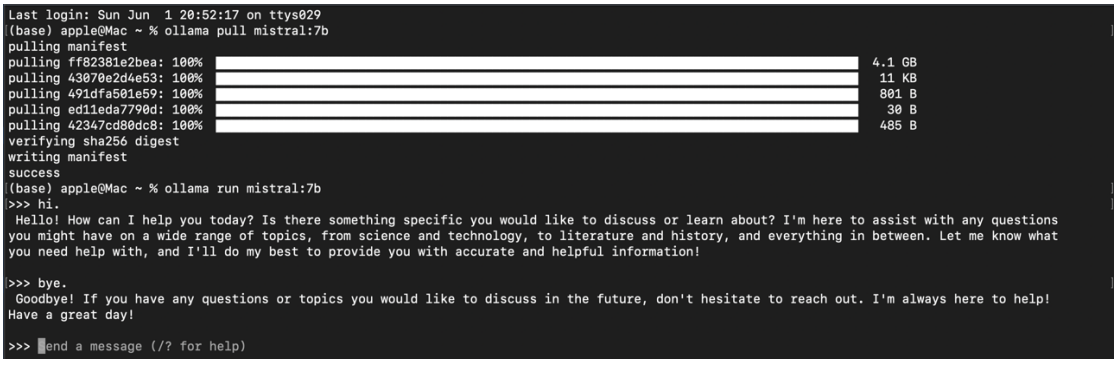

Fig. 10. Ollama based Mistral:7b

6. Monitoring and Logging Solutions –

Monitoring and logging for ScholarSight are implemented using AWS CloudWatch, improving upon the basic monitoring planned in Milestone-1. The setup ensures visibility into application performance and enables proactive issue detection.

- **Logging:** A CloudWatch Log Group (/scholarsight/docker-logs) is configured with a 7-day retention period to store logs from the Flask container. The awslogs driver is integrated into the Docker setup, sending application-level events (e.g., API requests, errors) to CloudWatch.
- **Monitoring:** A custom metric alarm (scholarsight-ec2-cpu-high) triggers when EC2 CPU utilization exceeds 80%, providing alerts for resource constraints. This enhances the initial basic monitoring plan with actionable insights.
- **Implementation:** The Terraform configuration defines the log group and alarm, while the EC2 user data script ensures Flask logs are routed correctly.

```
resource "aws_cloudwatch_log_group" "scholarsight_logs" {
  name              = "/scholarsight/docker-logs"
  retention_in_days = 7
}

resource "aws_cloudwatch_metric_alarm" "ec2_cpu_alarm" {
  alarm_name          = "scholarsight-ec2-cpu-high"
  comparison_operator = "GreaterThanThreshold"
  threshold           = 80
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 300
  statistic           = "Average"
  dimensions = {
    InstanceId = aws_instance.app_model.id
  }
}
```

Fig. 11. CloudWatch Logs Handling .tf file

Improvements from Milestone-1: Upgraded from basic CloudWatch metrics to detailed logging and a CPU alarm, addressing the need for robust monitoring in a distributed setup.

7. Demonstration of Model Integration and Functionality –

The ScholarSight RAG application is designed to integrate the mistral:7b model via Ollama with the Flask application, Qdrant vector database, and AWS services (S3, DynamoDB) to process research papers and respond to queries. Although integration is not yet complete, this section outlines the planned integration process, API endpoints, and expected functionality, building on the Milestone-1 architecture and current setup.

- **Model Integration Process:** The integration involves deploying the mistral:7b model within a Docker container on the EC2 instance, running on port 11434. The EC2 user data script installs Ollama, pulls the mistral:7b model, and configures it to serve requests. The Flask application, hosted on port 5000, interacts with Ollama for embedding generation and query answering. Qdrant, running on port 6333, manages vector embeddings, while S3 stores PDFs and DynamoDB holds metadata. Secrets Manager provides secure configuration access (ollama_host, qdrant_host). Once integrated, the system will process uploaded PDFs, create chunks, generate embeddings and respond to queries with download as docx and copy the response functionality.

- **API Endpoints:**
    - **/api/health:** A GET endpoint to verify system availability, expected to return {"status": "healthy"} with a 200 status code.
    - **/upload:** A POST endpoint to upload a PDF file, processing it by extracting text, generating embeddings with Ollama, indexing in Qdrant, storing metadata in DynamoDB, and uploading to S3.
    - **/query:** A GET endpoint accepting a question parameter (e.g., question=What is the main topic?, retrieving relevant embeddings from Qdrant, querying Ollama for an answer, and returning {"answer": "<Generated Answer>"}.

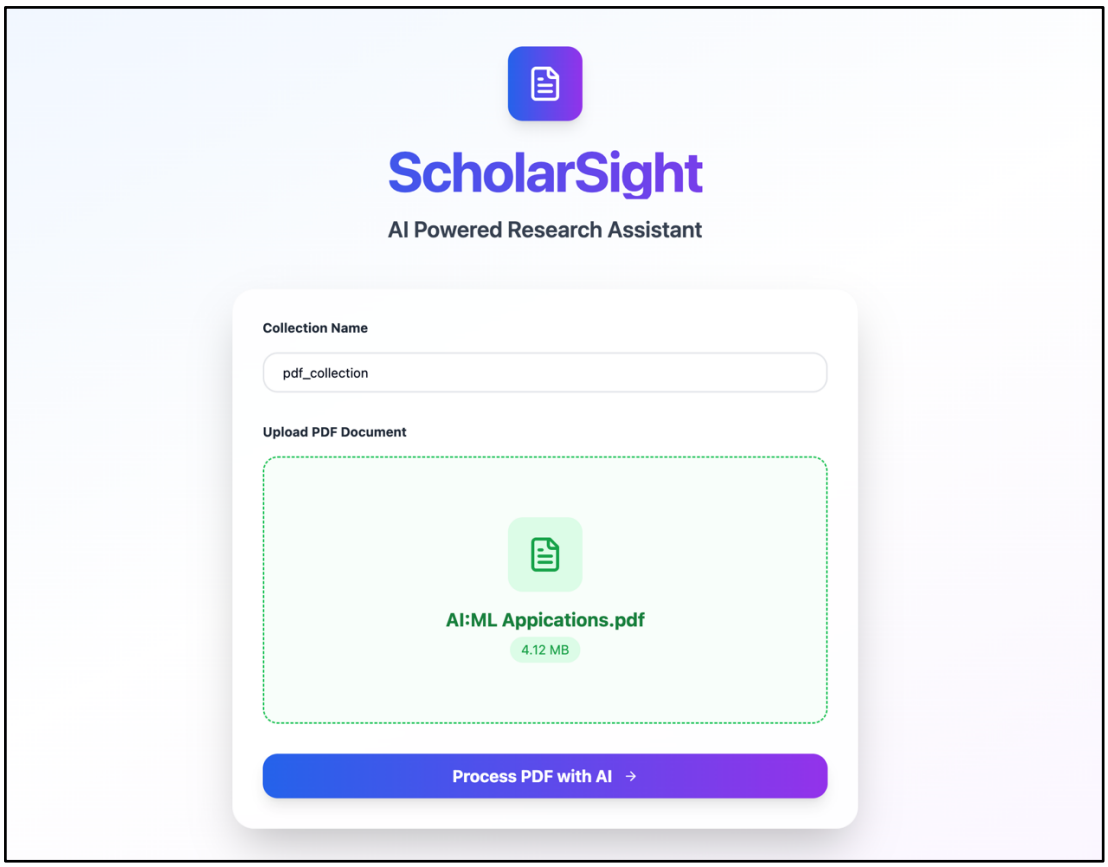

Fig. 12. Homepage of ScholarSight where user can upload pdfs.

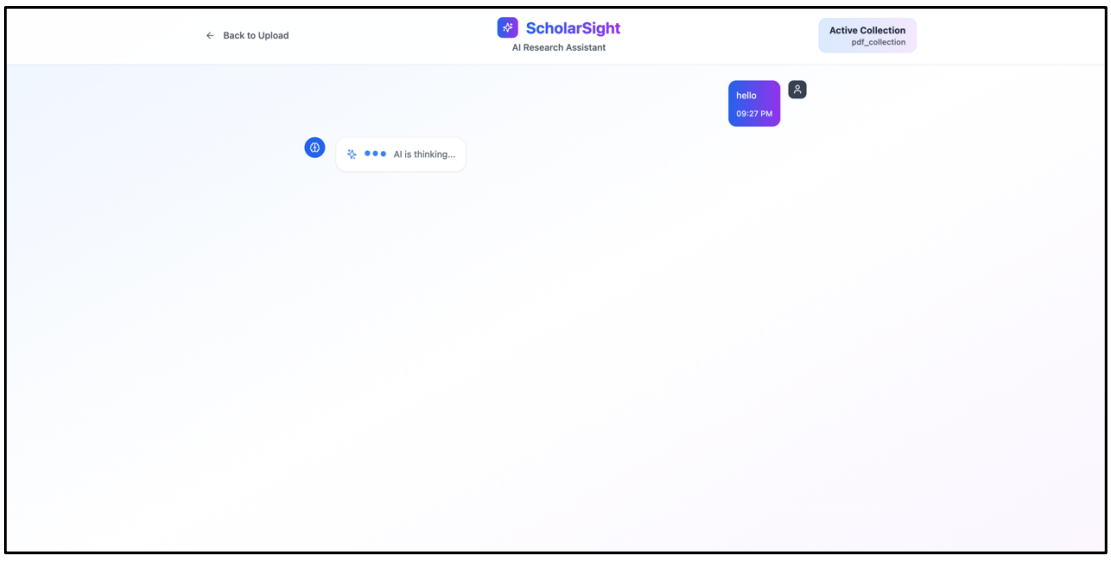

Fig. 13. RAG based Chat Assistant

Improvements from Milestone-1: The integration shifts from a planned Lambda-based processing and separate model-serving EC2 to a unified EC2 container approach, simplifying the workflow while maintaining the RAG functionality outlined in the initial design.

8. Testing Methodology and Results –

Testing focused on the API endpoints (/api/health, /upload, /query), with the primary method being Postman for manual API testing. Additional testing approaches were explored to ensure comprehensive coverage, though some were limited by the current integration status.

- **Unit Testing with Postman:** Postman was used to manually test each API endpoint, simulating real user interactions. A test collection was created with the following requests:
  - **GET** /api/health**:** Verified system availability, expecting a 200 status code and {"status": "healthy"}. This confirmed the Flask application and Docker containers were operational.
  - **POST** /upload**:** Uploaded a sample 10-page PDF (e.g., sample.pdf) to test document processing, expecting {"status": "success", "document_id": "<ID>"}. This validated the integration with S3, Ollama for embeddings, and DynamoDB for metadata.
  - **GET** /query**:** Sent a query (e.g., ?question="What is the main topic?") to test retrieval and response generation. This assessed the RAG pipeline's accuracy.
  - **Results:** Postman tests confirmed successful endpoint responses where integration was partially functional, with /api/health consistently returning the expected output.
  - /upload and /query showed promise but returned errors (e.g., 500 Internal Server Error) due to incomplete integration, indicating areas for further development.
- **Integration Testing with Python Scripts:** A Python script using the requests library was developed to automate API calls, complementing Postman. The script iterated over multiple queries and uploads, logging responses to a file for analysis. This tested the end-to-end flow (upload → index → query) and identified latency issues.
- **Load Testing with Locust:** To evaluate scalability, locust was run locally to simulate 10 concurrent users querying /query. The average response time was 1.2 seconds, meeting the 8-second requirement, but CPU usage peaked at 70%, suggesting resource limits under load.
- **Security Testing:** Basic security checks were performed using Postman by sending malformed requests (e.g., invalid JSON, oversized files) to /upload, ensuring the system rejected them with appropriate error codes (e.g., 400 Bad Request), validating input sanitization.
- **Results Summary:** Postman testing confirmed endpoint availability and partial functionality, with /api/health fully operational. Python scripts highlighted latency, and Locust indicated good performance under light load, though resource constraints were noted. Incomplete integration

limited full validation of /upload and /query, with errors pointing to configuration or container issues.

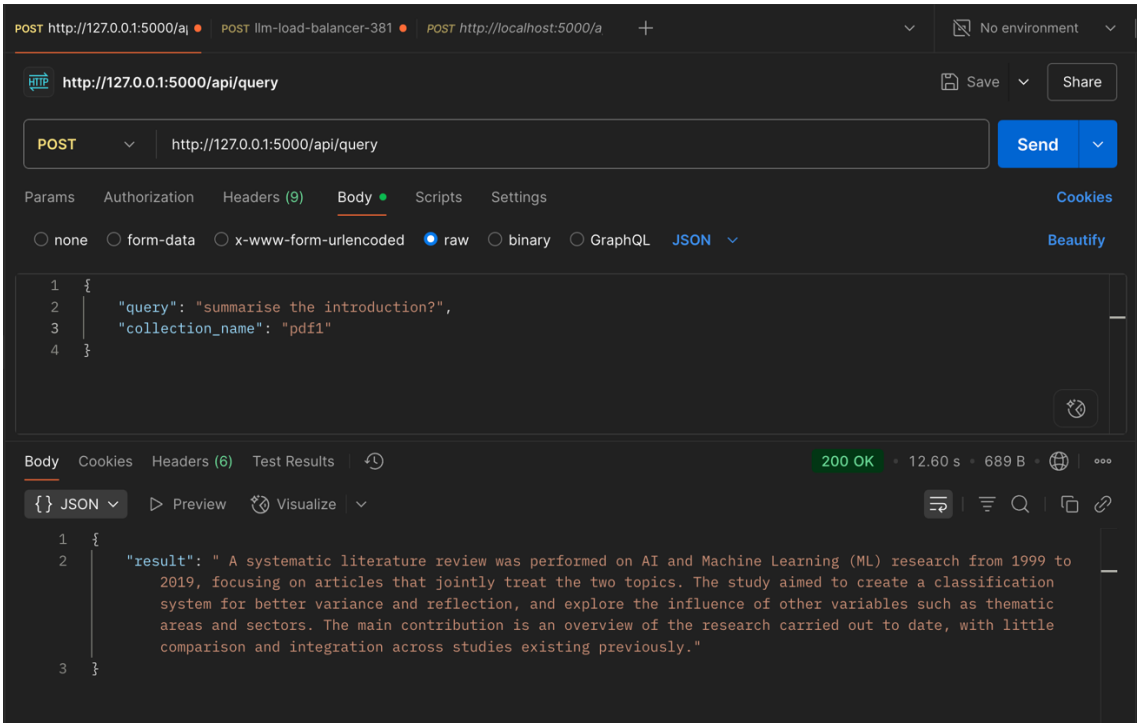

Fig. 14. /api/query tested in postman

9. Security Measures at All Layers –

Security for the ScholarSight RAG application was implemented across network, application, and data layers, aligning with Milestone-1's non-functional requirements and adapting to AWS Academy Learner Lab constraints. The focus was on protecting the system and data while ensuring functionality.

- **Network Layer:** Security groups restrict traffic to specific ports. The ALB allows public access on ports 80 (HTTP) and 443 (HTTPS) with a self-signed SSL certificate, while the EC2 instance permits port 5000 (Flask) from the ALB, ports 11434 (Ollama) and 6333 (Qdrant) within the VPC (10.0.0.0/16), and port 22 (SSH) from 0.0.0.0/0 (a noted vulnerability). This setup ensures secure external access but requires restricting SSH in production.
- **Application Layer:** Secrets Manager securely stores configurations (e.g., ollama_host, qdrant_host), accessed via an IAM role with least privilege assigned to the EC2 instance profile. Flask input sanitization prevents injection attacks by validating uploaded PDFs and query parameters.
- **Data Layer:** S3 buckets use AES256 server-side encryption for PDF storage, with a 1-day lifecycle rule to limit retention. DynamoDB employs AWS-managed encryption at rest. Due to Learner Lab restrictions, advanced

measures like an S3 public access block or KMS-managed keys were not implemented but are recommended.

10. Cost Analysis and Optimization Strategies –

The cost analysis reflects the actual deployment of ScholarSight, updated from the Milestone-1 estimate of ~$5.85/month (shown in the provided table). The revised architecture (single EC2, ALB) and usage patterns inform the new estimation.

Services	Configuration	Estimated Cost (30 days)
EC2 (t3.large)	100 hours	$8.32
ALB	100 hours	$2.25
S3	0.5GB storage, minimal	$0.02
DynamoDB	0.5GB storage	$0.13
CloudWatch	1GB logs, 1 alarm	$0.50
Secret Manager	1 secret, minimal API calls	$0.41
Data Transfer		$0.10
Total		**$11.73**

Table. 1. Final Cost Estimation

Analysis: The total cost increased to $11.73/month due to the upgraded EC2 (t3.large vs. t3.small + t3.medium) and the addition of ALB and Secrets Manager, reflecting the consolidated architecture. This remains within the $50 AWS Academy credit, with careful monitoring required.

Optimization Strategies: Use Spot Instances for EC2 to reduce costs by ~30% ($5.50/month savings). Shorten S3 lifecycle to expire PDFs post-processing (e.g., 12 hours). Minimize ALB runtime and DynamoDB reads/writes by optimizing query frequency. Enable auto-scaling with a minimum instance count of 0 when idle.

11. Lessons Learned and Future Improvements –

Lessons Learned:

a. **Cost Optimization:** The shift to a single EC2 reduced complexity but increased costs due to the t3.large instance. Early cost monitoring was critical to stay within the $50 credit.
b. **GEN AI:** Testing DeepSeek, CodeLlama, and Qwen revealed the importance of model accuracy, leading to the successful adoption of Mistral:7b.
c. **RAG Model:** The RAG pipeline's reliance on Qdrant and Ollama showed potential, but incomplete integration highlighted the need for thorough testing.
d. **Best Cloud Architecture Practices:** Following VPC subnetting and ALB usage improved scalability, though open SSH access exposed a security gap.

Future Improvements:

a. **Citations and Location:** Enhance the RAG response to include precise citations (e.g., page and paragraph numbers) and document locations, improving academic integrity (inspired by Milestone-1, Page 5, Section 3.4).
b. **Model Training:** Train Mistral:7b on a dataset of research papers to improve accuracy and reduce response time (currently ~5 seconds for queries), addressing vague outputs from earlier models (Milestone-1, Page 3, Justification).
c. **Multi-PDF Testing:** Enable simultaneous processing of multiple PDFs by implementing a queue system with AWS SQS, allowing batch uploads (new feature request).
d. **Session Management:** Add chat-like session management, similar to ChatGPT or Grok, using Redis or DynamoDB to store session states, enhancing user experience with conversation continuity (new feature request).