

# RISHIT BURMAN

## Intelligent Financial Document Engine - Project Documentation

---

### Overview

This project presents a robust **Intelligent Document Retrieval System** designed to handle the **extraction, semantic indexing, and question-answering over financial documents**. It leverages powerful AWS services including **Textract, DynamoDB, S3, SageMaker**, and **OpenSearch** for secure, scalable, and efficient processing of enterprise documents — while keeping data *private* (no use of public LLMs).

---

### Why This Project Matters

#### Business Impact

- Automates the reading and searching of thousands of financial reports, audit documents, quarterly filings, etc.
- Eliminates manual efforts by finance teams, saving **hours of time per document**.
- Ensures **accuracy** using semantic search powered by **embeddings**.
- Keeps enterprise data **completely private** and never leaves AWS.

#### Potential Use-Cases:

- Financial firms (for annual/quarterly report analysis)
- Hospitals and pharma (for report lookup & compliance)

- Enterprises (for internal document retrieval)
  - Law firms (for fast document clause retrieval)
- 

## Architecture Breakdown

### Part 1: Document Upload & Text Extraction

#### Flow:

1. User uploads a PDF/Scanned image to **S3 Bucket** (doc-engine-bucket-risbur).
2. A PUT event triggers a **Lambda function** named doc-upload-processor.
3. This Lambda uses **Amazon Textract** to extract structured text from the uploaded document.
4. The text and document metadata (like file name) are stored in **DynamoDB table** DocumentTextTable.

#### ✓ Why Amazon Textract?

- It supports scanned financial documents (images, tables, PDFs).
- Can extract text from complex layouts with high accuracy.
- Fully serverless and scalable.

aws [Search] [Alt+S] United States (N. Virginia) rishit-admin @ 2558-4567-5055

Amazon S3 > Buckets > doc-engine-bucket-risbur

No data events  
No data events to display.  
[Configure in CloudTrail](#)

**Event notifications (1)** [Edit](#) [Delete](#) [Create event notification](#)

Send a notification when specific events occur in your bucket. [Learn more](#)

<input type="checkbox"/>	Name	Event types	Filters	Destination type	Destination
<input type="checkbox"/>	trigger-lambda-upload	Put	-	Lambda function	<a href="#">doc-upload-processor</a>

**Amazon EventBridge** [Edit](#)

For additional capabilities, use Amazon EventBridge to build event-driven applications at scale using S3 event notifications. [Learn more](#) or [see EventBridge pricing](#)

Send notifications to Amazon EventBridge for all events in this bucket  
Off

**Transfer acceleration** [Edit](#)

Use an accelerated endpoint for faster data transfers. [Learn more](#)

Transfer acceleration  
Disabled

- Upload event from S3 triggering Lambda.

aws [Search] [Alt+S] United States (N. Virginia) rishit-admin @ 2558-4567-5055

CloudWatch > Log groups > /aws/lambda/doc-upload-processor > 2025/06/19/[LATEST]30b301ffe2a74de8a97d69f1e3b3711a

**CloudWatch** < Favorites and recents ▶

Dashboards

▶ AI Operations [Preview](#)

▼ Alarms [△](#) [○](#) [○](#) [○](#)

In alarm

All alarms

Billing

▼ Logs

[Log groups](#)

Log Anomalies

Live Tail

Logs Insights [New](#)

Contributor Insights

▼ Metrics

All metrics

Explorer

**Log events** [Actions](#) [Start tailing](#) [Create metric filter](#)

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

[Clear](#) [1m](#) [30m](#) [1h](#) [12h](#) [Custom](#) [UTC timezone](#)

[Display](#)

▶	Timestamp	Message
▶	2025-06-19T14:15:14.773Z	INIT_START Runtime Version: python:3.10.v78 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:b556158cad85934b6c377a5efb9a60...
▶	2025-06-19T14:15:15.064Z	START RequestId: 5c62ef22-f576-4bb1-b5e2-19e2389a043b Version: \$LATEST
▶	2025-06-19T14:15:15.065Z	Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us-east-1", "eventTime": "2025-06-19T14:15:13...
▶	2025-06-19T14:15:18.594Z	Extracted text: Financial Report - Q4 2024
▶	2025-06-19T14:15:18.594Z	Company: GlobalShop Inc.
▶	2025-06-19T14:15:18.594Z	Report Date: December 31, 2024
▶	2025-06-19T14:15:18.594Z	Prepared By: Finance Department
▶	2025-06-19T14:15:18.594Z	Summary:
▶	2025-06-19T14:15:18.594Z	Q4 Revenue reached \$2 million, marking a 12% increase from Q3.
▶	2025-06-19T14:15:18.594Z	Operating expenses stood at \$750,000.
▶	2025-06-19T14:15:18.594Z	Net profit recorded: \$1.25 million.
▶	2025-06-19T14:15:18.594Z	Top performing product: SmartGadget Pro

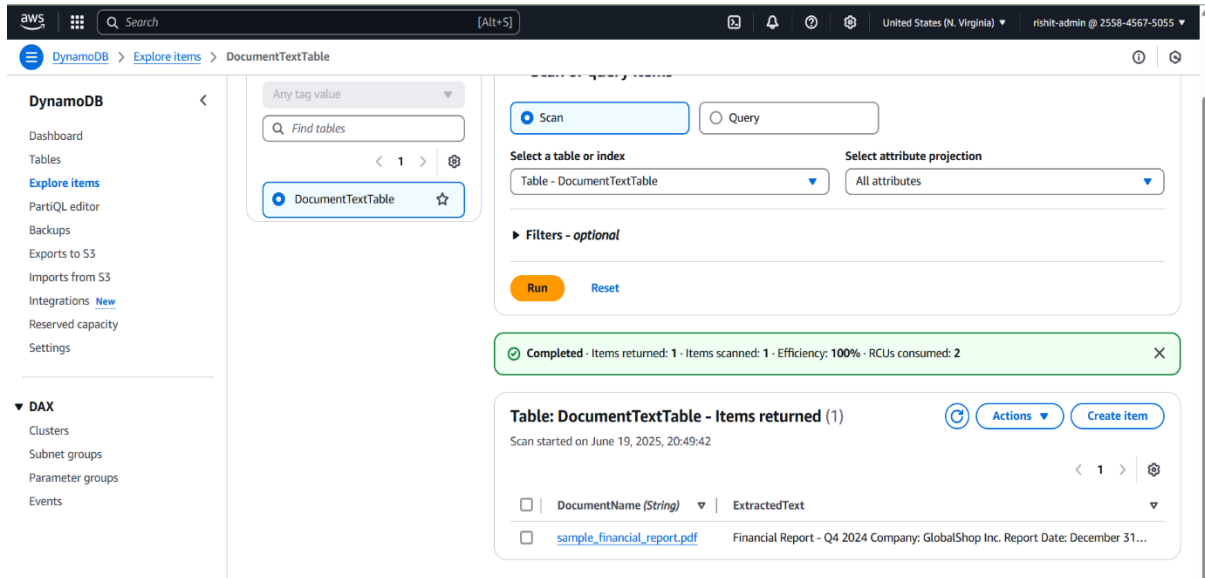
[Back to top](#)

- Extracted Textract output from CloudWatch logs.

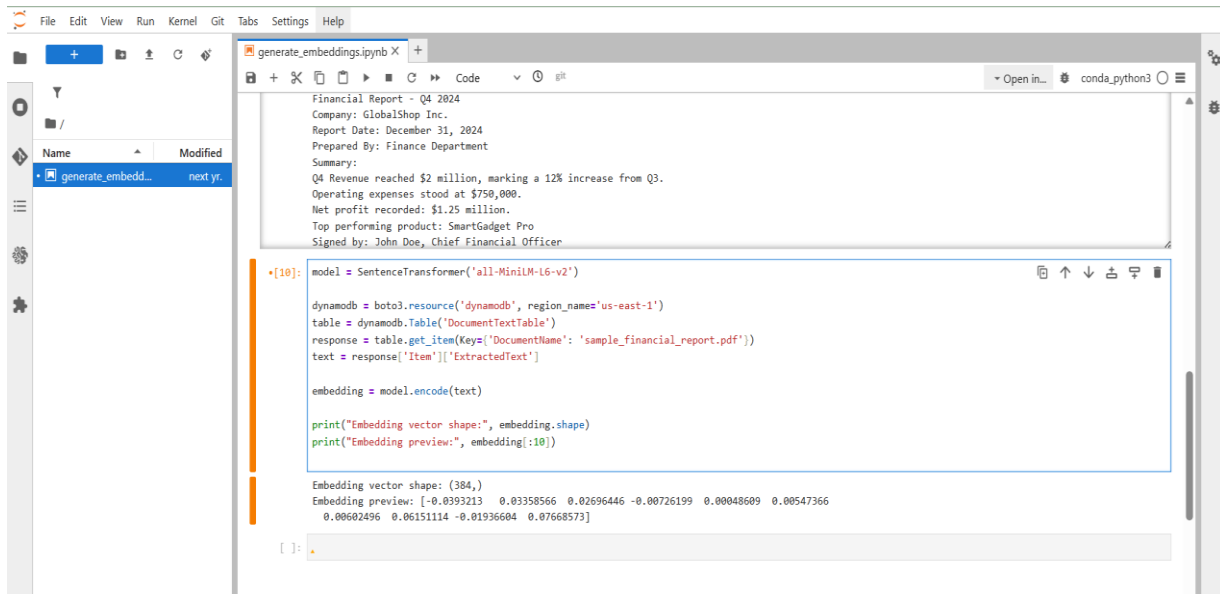
## Part 2: Embedding Generation & Semantic Indexing

### Tool: Jupyter Notebook (generate\_embeddings.ipynb)

#### 1. Loads each document's text from **DynamoDB**.

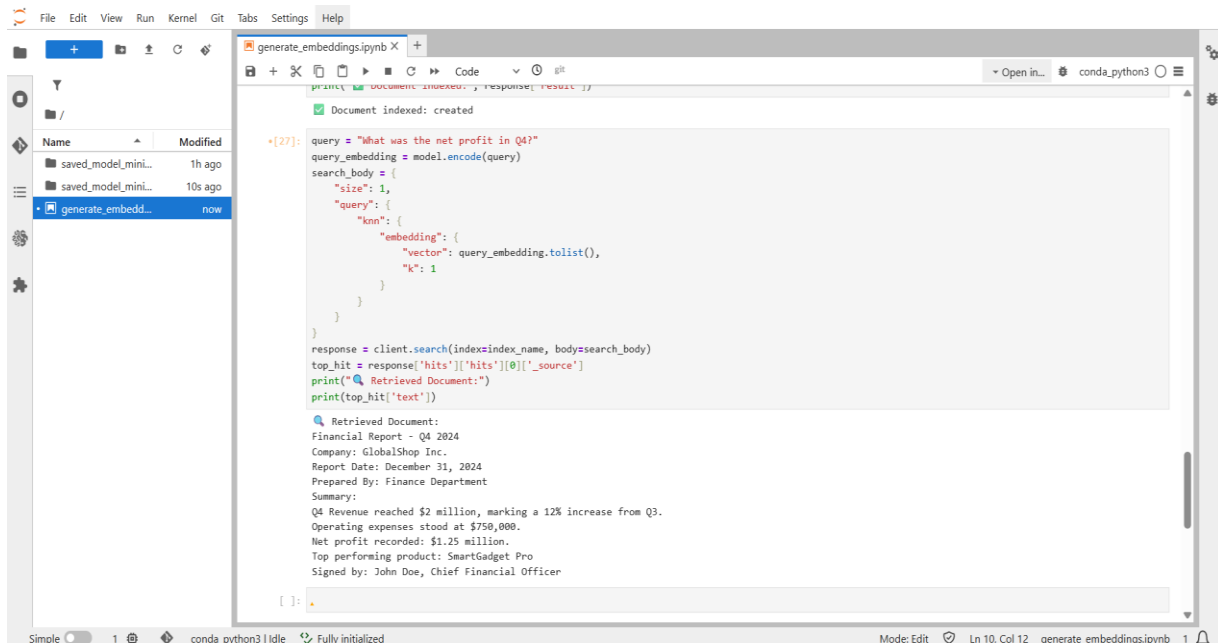


#### 2. Uses a **SentenceTransformer MiniLM model** hosted on **SageMaker** to generate **text embeddings** (vector representations of document meaning).



### 3. Embeddings + metadata are stored in **OpenSearch (KNN Index)** for fast semantic retrieval.

- Output of generated embeddings.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The code in the notebook is as follows:

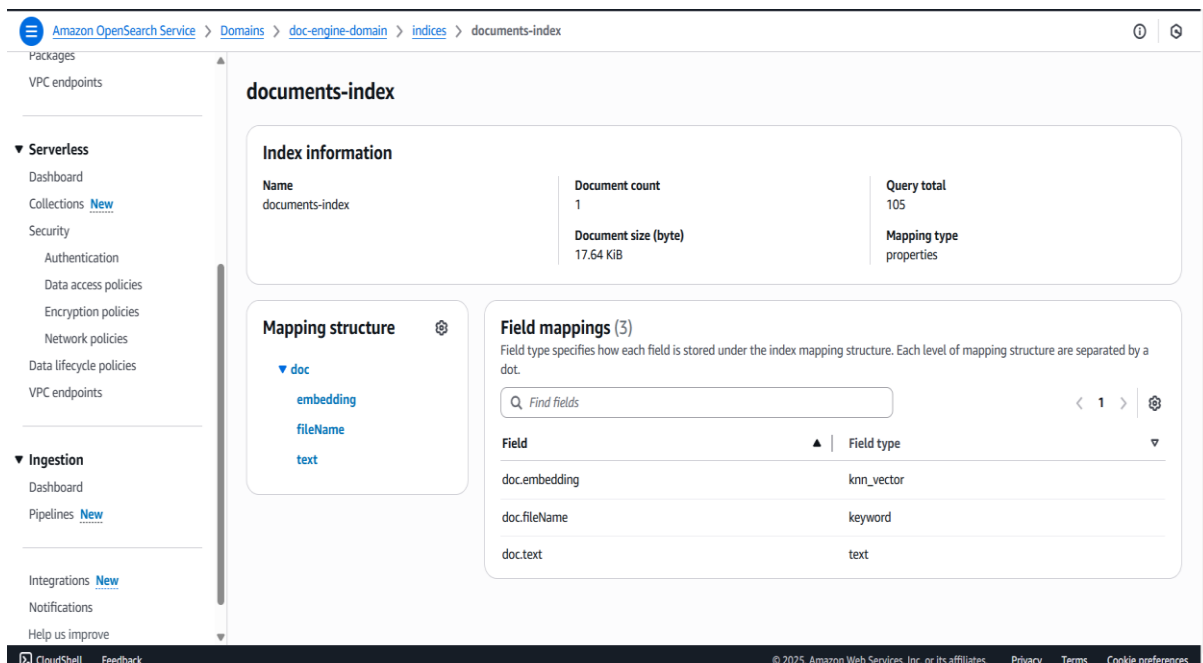
```
print("DOCUMENT INDEXED", response["result"])
# Document indexed: created

*{27}: query = "What was the net profit in Q4?"
query_embedding = model.encode(query)
search_body = {
    "size": 1,
    "query": {
        "knn": {
            "embedding": {
                "vector": query_embedding.tolist(),
                "k": 1
            }
        }
    }
}
response = client.search(index=index_name, body=search_body)
top_hit = response['hits'][0]['_source']
print("Retrieved Document:")
print(top_hit['text'])
```

The output of the search query is displayed below the code:

```
Retrieved Document:
Financial Report - Q4 2024
Company: GlobalShop Inc.
Report Date: December 31, 2024
Prepared By: Finance Department
Summary:
Q4 Revenue reached $2 million, marking a 12% increase from Q3.
Operating expenses stood at $750,000.
Net profit recorded: $1.25 million.
Top performing product: SmartGadget Pro
Signed by: John Doe, Chief Financial Officer
```

- Indexing confirmation on OpenSearch dashboard.



## OpenSearch Vector Database — How It Works

OpenSearch's KNN indexing allows you to:

- Store **high-dimensional embedding vectors** (from SageMaker).
- Perform **Approximate Nearest Neighbour Search** on these vectors.
- Retrieve **most semantically relevant document** based on the user's question.

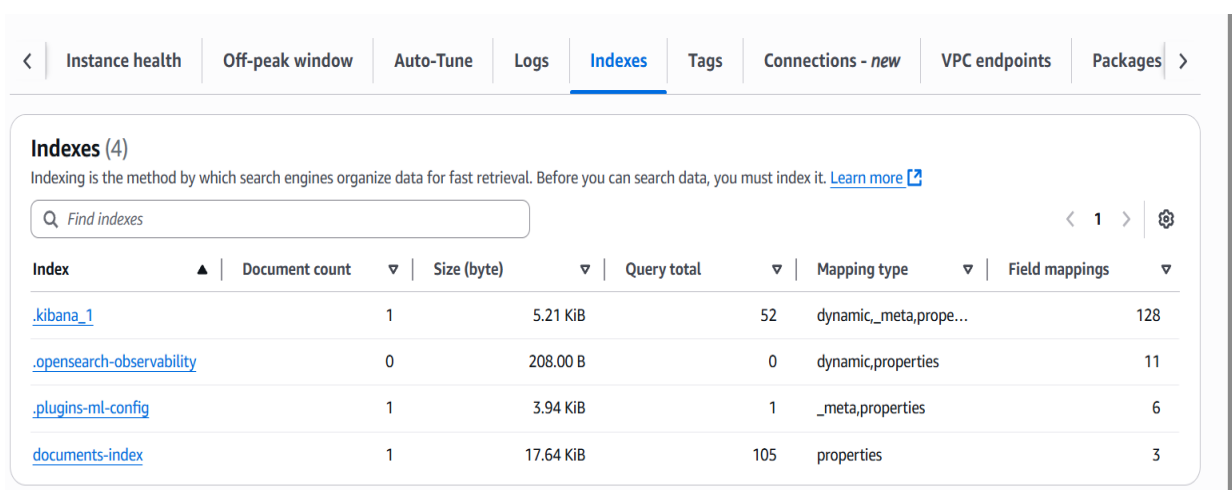
### Example:

Query: *"What is the net profit?"*

OpenSearch returns the document that has the closest vector embedding to this question and pulls the matching line.

### Advantage over keyword search:

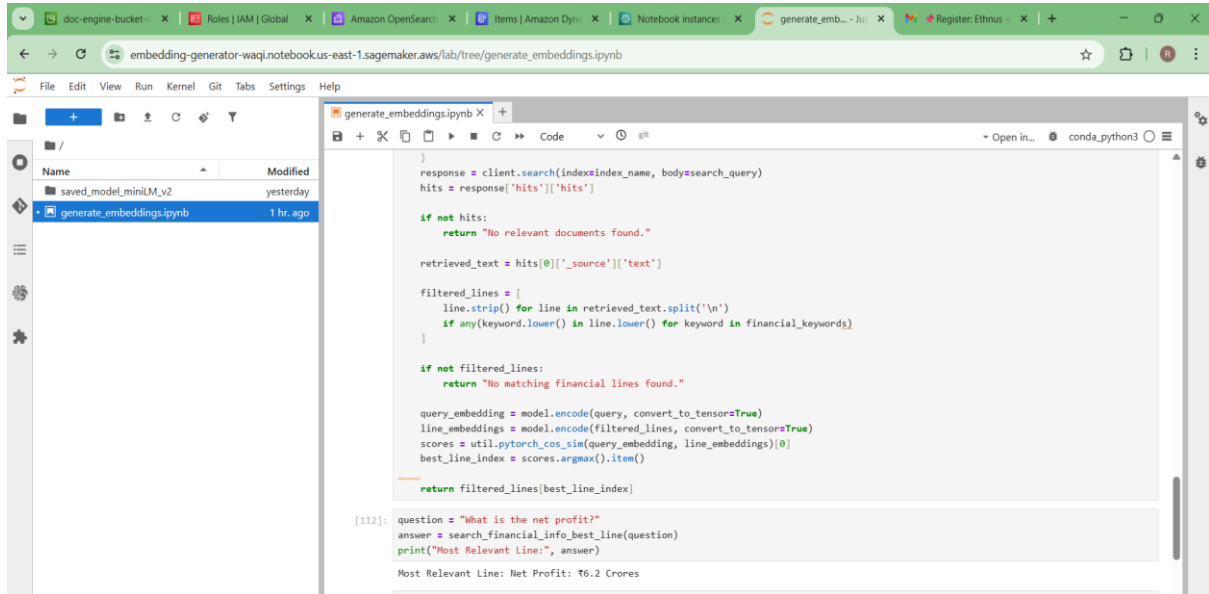
- Can handle **paraphrased queries** (e.g., “profit for the quarter” → “Net profit recorded: \$1.25 million”).
- Robust even when documents use different wordings.
- OpenSearch index view.



The screenshot shows the OpenSearch 'Indexes' page. At the top, there's a navigation bar with tabs: Instance health, Off-peak window, Auto-Tune, Logs, Indexes (selected), Tags, Connections - new, VPC endpoints, and Packages. Below the navigation bar, the 'Indexes (4)' section is displayed. It includes a search bar with the text 'Find indexes' and a pagination control showing '< 1 >' and a settings icon. The main content is a table with the following columns: Index, Document count, Size (byte), Query total, Mapping type, and Field mappings. The table lists four indexes: .kibana\_1, .opensearch-observability, .plugins-ml-config, and documents-index.

Index	Document count	Size (byte)	Query total	Mapping type	Field mappings
<a href="#">.kibana_1</a>	1	5.21 KiB	52	dynamic,_meta,prope...	128
<a href="#">.opensearch-observability</a>	0	208.00 B	0	dynamic,properties	11
<a href="#">.plugins-ml-config</a>	1	3.94 KiB	1	_meta,properties	6
<a href="#">documents-index</a>	1	17.64 KiB	105	properties	3

- Search query payload and matching result.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory with files 'saved\_model\_miniLM\_v2' and 'generate\_embeddings.ipynb'. The code editor displays the following Python code:

```
}
response = client.search(index=index_name, body=search_query)
hits = response['hits']['hits']

if not hits:
    return "No relevant documents found."

retrieved_text = hits[0]['_source']['text']

filtered_lines = [
    line.strip() for line in retrieved_text.split('\n')
    if any(keyword.lower() in line.lower() for keyword in financial_keywords)
]

if not filtered_lines:
    return "No matching financial lines found."

query_embedding = model.encode(query, convert_to_tensor=True)
line_embeddings = model.encode(filtered_lines, convert_to_tensor=True)
scores = util.pytorch_cos_sim(query_embedding, line_embeddings)[0]
best_line_index = scores.argmax().item()

return filtered_lines[best_line_index]
```

Below the code, the output of the notebook is shown:

```
[112]: question = "What is the net profit?"
answer = search_financial_info_best_line(question)
print("Most Relevant Line:", answer)

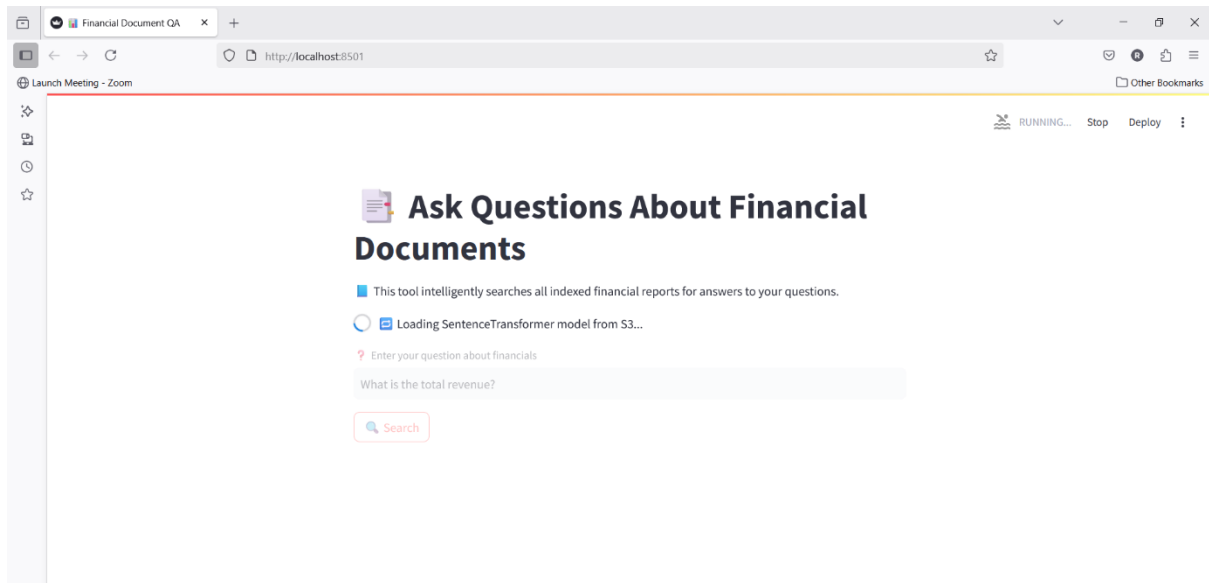
Most Relevant Line: Net Profit: ₹6.2 Crores
```

## Streamlit Frontend App

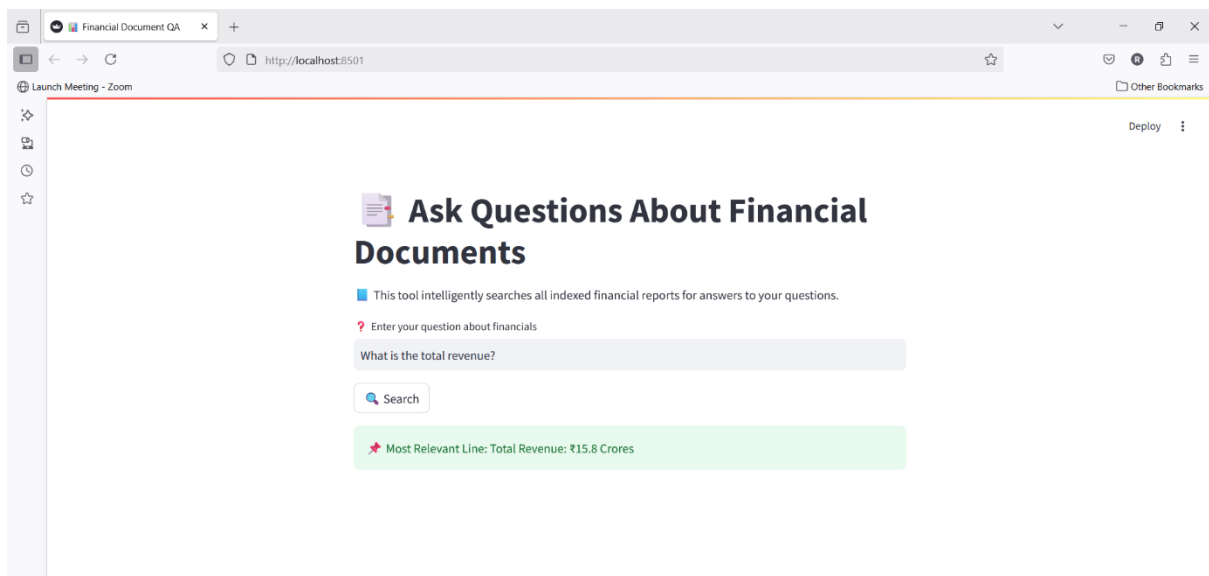
### Functionality:

- Web UI for users to **ask questions** about uploaded financial documents.
- Loads the model dynamically from **S3**.
- Searches **across all indexed documents** and returns the **most relevant answer line**.
- No document selection needed — all searches are **aggregated**.

- Streamlit interface with user question.

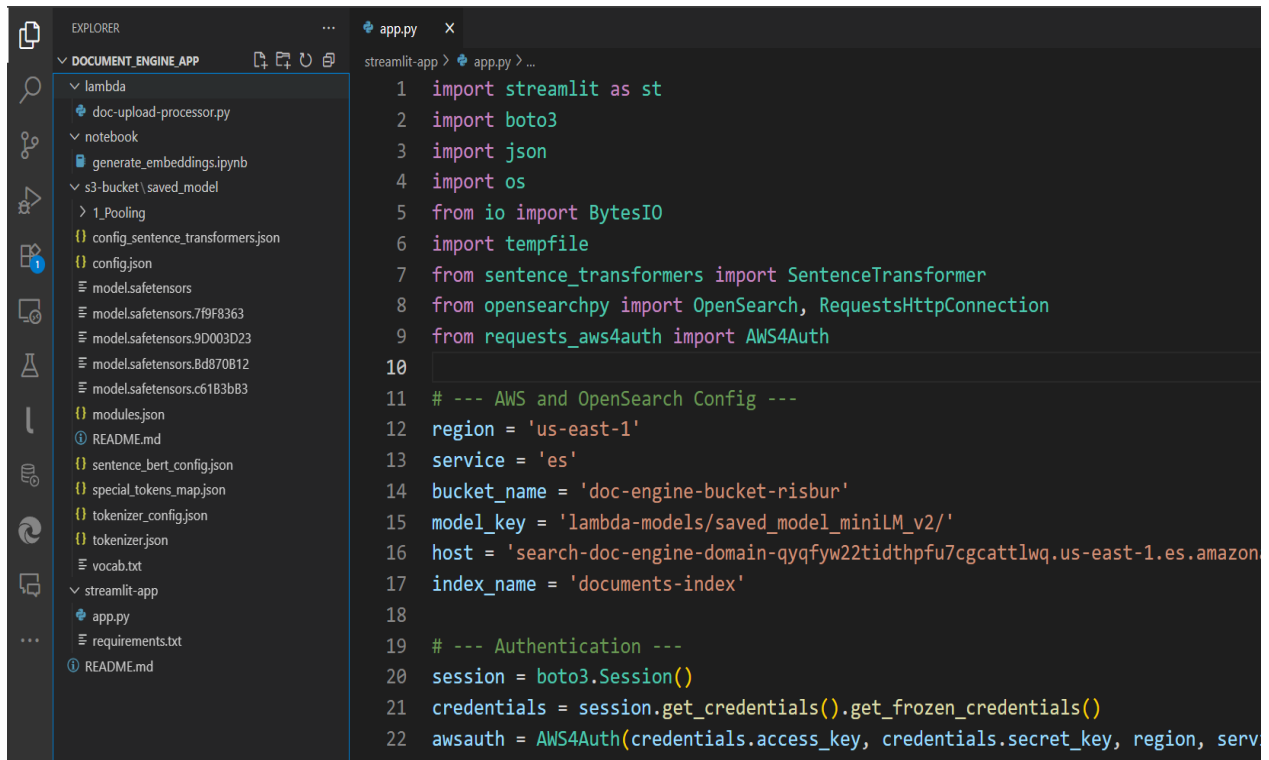


- Output with the most relevant line highlighted.





## Project Structure



The screenshot shows a VS Code editor with a project named 'DOCUMENT\_ENGINE\_APP'. The Explorer sidebar on the left lists the following files and folders:

- lambda
  - doc-upload-processor.py
- notebook
  - generate\_embeddings.ipynb
- s3-bucket \ saved\_model
  - 1\_Pooling
    - config\_sentence\_transformers.json
    - config.json
    - model.safetensors
    - model.safetensors.7f9f8363
    - model.safetensors.9D003D23
    - model.safetensors.Bd870B12
    - model.safetensors.c61B3b83
    - modules.json
    - README.md
    - sentence\_bert\_config.json
    - special\_tokens\_map.json
    - tokenizer\_config.json
    - tokenizer.json
    - vocab.txt
- streamlit-app
  - app.py
  - requirements.txt
  - README.md

The main editor window shows the code for 'app.py' in the 'streamlit-app' directory. The code is as follows:

```
1 import streamlit as st
2 import boto3
3 import json
4 import os
5 from io import BytesIO
6 import tempfile
7 from sentence_transformers import SentenceTransformer
8 from opensearchpy import OpenSearch, RequestsHttpConnection
9 from requests_aws4auth import AWS4Auth
10
11 # --- AWS and OpenSearch Config ---
12 region = 'us-east-1'
13 service = 'es'
14 bucket_name = 'doc-engine-bucket-risbur'
15 model_key = 'lambda-models/saved_model_miniLM_v2/'
16 host = 'search-doc-engine-domain-qyqfyw22tidthpfu7cgcatlqw.us-east-1.es.amazonaws.com'
17 index_name = 'documents-index'
18
19 # --- Authentication ---
20 session = boto3.Session()
21 credentials = session.get_credentials().get_frozen_credentials()
22 awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, serv
```

---

## Technologies Used

Component	Technology
Text Extraction	AWS Textract
Storage	S3, DynamoDB
Embeddings	SentenceTransformer (MiniLM)
Model Hosting	SageMaker
Vector Index	OpenSearch KNN
App Frontend	Streamlit
Deployment	EC2 t2.micro (Free Tier)

Component	Technology
Authentication	Boto3 + SigV4 (requests-aws4auth)

---

### ✓ Benefits Recap

- **Privacy First:** No data goes to third-party LLMs.
  - **Fast Search:** Instant semantic answers from large documents.
  - **Contextual Understanding:** Not just keyword matching.
  - **Industry-Ready:** Applicable in finance, law, healthcare.
- 

## How to Run the Project

### 1. Upload Financial Docs to S3

S3 Bucket: doc-engine-bucket-risbur

Trigger Lambda: doc-upload-processor

Stores in: DocumentTextTable

### 2. Run `generate_embeddings.ipynb`

Generates vector embeddings & stores in OpenSearch index.

### 3. Launch Streamlit UI

`streamlit run app.py`

---

- ✓ S3 document upload interface.

The screenshot shows the AWS S3 document upload interface. At the top, there's a navigation bar with the AWS logo, a search bar, and user information. Below the navigation bar, the main heading is "Upload: status" with a "Close" button. A message box states: "After you navigate away from this page, the following information is no longer available." Below this, there's a "Summary" section with three columns: "Destination" (s3://doc-engine-bucket-risbur), "Succeeded" (1 file, 1.2 KB (100.00%)), and "Failed" (0 files, 0 B (0%)). Below the summary, there are two tabs: "Files and folders" (selected) and "Configuration". The "Files and folders" section shows a table with one file: "sample\_financial\_report.pdf" (application/pdf, 1.2 KB, Succeeded). The table has columns for Name, Folder, Type, Size, Status, and Error.

- ✓ Lambda function deployment

The screenshot shows the AWS Lambda console for the "doc-upload-processor" function. At the top, there's a green notification bar: "Successfully updated the function doc-upload-processor." Below this, the "Code source" section is active, showing the "lambda\_function.py" file. The code is as follows:

```

1 import json
2 import boto3
3 import urllib.parse
4
5 def lambda_handler(event, context):
6     print("Event:", json.dumps(event))
7
8     s3 = boto3.client('s3')
9     textract = boto3.client('textract', region_name='us-east-1') # Specify region explicitly
10
11     for record in event['Records']:
12         bucket = record['s3']['bucket']['name']
13         key = urllib.parse.unquote_plus(record['s3']['object']['key'])
14
15     try:
16         response = textract.detect_document_text(
17             Document={'S3Object': {'Bucket': bucket, 'Name': key}}
18         )

```

On the right side, there's a "Tutorials" section with a "Create a simple web app" tutorial. The tutorial text says: "In this tutorial you will learn how to: Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage. Invoke your function through its function URL." There's a "Learn more" link and a "Start tutorial" button.

- ✓ Lambda execution log showing Textract.

The screenshot shows the AWS CloudWatch console. The breadcrumb navigation indicates the path: CloudWatch > Log groups > /aws/lambda/doc-upload-processor > 2025/06/19/[\$LATEST]30b301ffe2a74de8a97d69f1e3b3711a. The left sidebar shows the 'Log groups' link under the 'Logs' section. The main area is titled 'Log events' and contains a search bar, a 'Display' button, and a table of log events. The events show the initialization of a Lambda function, the start of a request, and the extraction of text from a PDF file named 'sample\_financial\_report.pdf'. The extracted text includes 'Financial Report - Q4 2024', 'Company: GlobalShop Inc.', 'Report Date: December 31, 2024', 'Prepared By: Finance Department', 'Summary: Q4 Revenue reached \$2 million, marking a 12% increase from Q3.', 'Operating expenses stood at \$750,000.', 'Net profit recorded: \$1.25 million.', and 'Top performing product: SmartGadget Pro'.

Timestamp	Message
2025-06-19T14:15:14.773Z	INIT_START Runtime Version: python:3.10.v78 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:b556158cad85934b6c377a5efb9a60...
2025-06-19T14:15:15.064Z	START RequestId: 5c62ef22-f576-4bb1-b5e2-19e2389a043b Version: \$LATEST
2025-06-19T14:15:15.065Z	Event: {"Records": [{"eventVersion": "2.1", "eventSource": "aws:s3", "awsRegion": "us-east-1", "eventTime": "2025-06-19T14:15:13..."}
2025-06-19T14:15:18.594Z	Extracted text: Financial Report - Q4 2024
2025-06-19T14:15:18.594Z	Company: GlobalShop Inc.
2025-06-19T14:15:18.594Z	Report Date: December 31, 2024
2025-06-19T14:15:18.594Z	Prepared By: Finance Department
2025-06-19T14:15:18.594Z	Summary:
2025-06-19T14:15:18.594Z	Q4 Revenue reached \$2 million, marking a 12% increase from Q3.
2025-06-19T14:15:18.594Z	Operating expenses stood at \$750,000.
2025-06-19T14:15:18.594Z	Net profit recorded: \$1.25 million.
2025-06-19T14:15:18.594Z	Top performing product: SmartGadget Pro

- ✓ DynamoDB console showing text records.

The screenshot shows the AWS DynamoDB console. The breadcrumb navigation indicates the path: DynamoDB > Explore items > DocumentTextTable. The left sidebar shows the 'Explore items' link. The main area is titled 'DocumentTextTable' and contains a 'Scan' button, a 'Query' button, and a table of scan results. The scan results show that the scan is completed, with 1 item returned, 1 item scanned, 100% efficiency, and 2 RCUs consumed. The table 'DocumentTextTable' has 1 item returned, and the scan started on June 19, 2025, at 20:49:42. The table has two columns: 'DocumentName (String)' and 'ExtractedText'. The first item is 'sample\_financial\_report.pdf' with the value 'Financial Report - Q4 2024 Company: GlobalShop Inc. Report Date: December 31...'.

DocumentName (String)	ExtractedText
sample_financial_report.pdf	Financial Report - Q4 2024 Company: GlobalShop Inc. Report Date: December 31...

- ✓ SageMaker embedding generation outputs.

The screenshot shows a SageMaker JupyterLab interface. On the left, a file browser displays a directory with files 'saved\_model\_minilm\_v2' (modified yesterday) and 'generate\_embeddings.ipynb' (modified 1 hr ago). The main area shows the 'generate\_embeddings.ipynb' notebook. The code in the notebook is as follows:

```
print(text[:500])

Sample Financial Report - Q1 2025
Company: FinNova Technologies Pvt. Ltd.
Report Period: January 1 - March 31, 2025
Prepared By: Finance & Accounts Department
Date: April 5, 2025
Summary Highlights:
Total Revenue: ₹15.8 Crores
Cost of Goods Sold (COGS): ₹6.2 Crores
Operating Expenses: ₹3.4 Crores
Net Profit: ₹6.2 Crores
EBITDA Margin: 39.2%
Segment Performance:
Retail Banking:
Revenue - ₹9.1 Cr | Net Profit - ₹3.9 Cr
FinTech Services:
Revenue - ₹6.7 Cr | Net Profit - ₹2.3 Cr
Additional Notes:
To

[103]: embedding_vector = model.encode(text)
print("Embedding vector shape:", embedding_vector.shape)
print("Embedding preview:", embedding_vector[:10])

Embedding vector shape: (384,)
Embedding preview: [-0.05999449 -0.02339429 -0.03599763 -0.00625608 -0.00258177 -0.00882858
 0.01203891 0.10468467 -0.00113416 0.02123569]

[104]: model.save('saved_model_minilm_v2')
```

- ✓ OpenSearch dashboard (index + knn).


The screenshot shows the Amazon OpenSearch Service dashboard for the 'documents-index'. The left sidebar contains navigation links for 'Serverless' (Dashboard, Collections, Security) and 'Ingestion' (Dashboard, Pipelines, Integrations, Notifications). The main content area is titled 'documents-index' and contains the following sections:

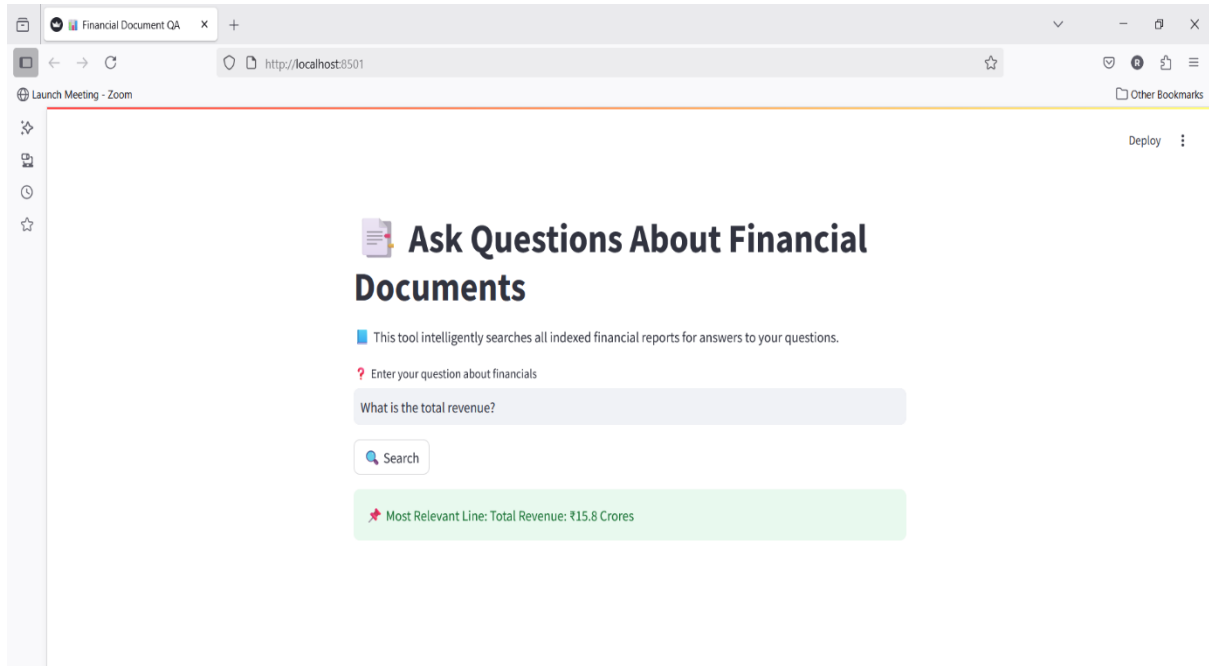
- Index information:**

Name documents-index	Document count 1	Query total 105
	Document size (byte) 17.64 KIB	Mapping type properties
- Mapping structure:**
  - doc
    - embedding
    - fileName
    - text
- Field mappings (3):**

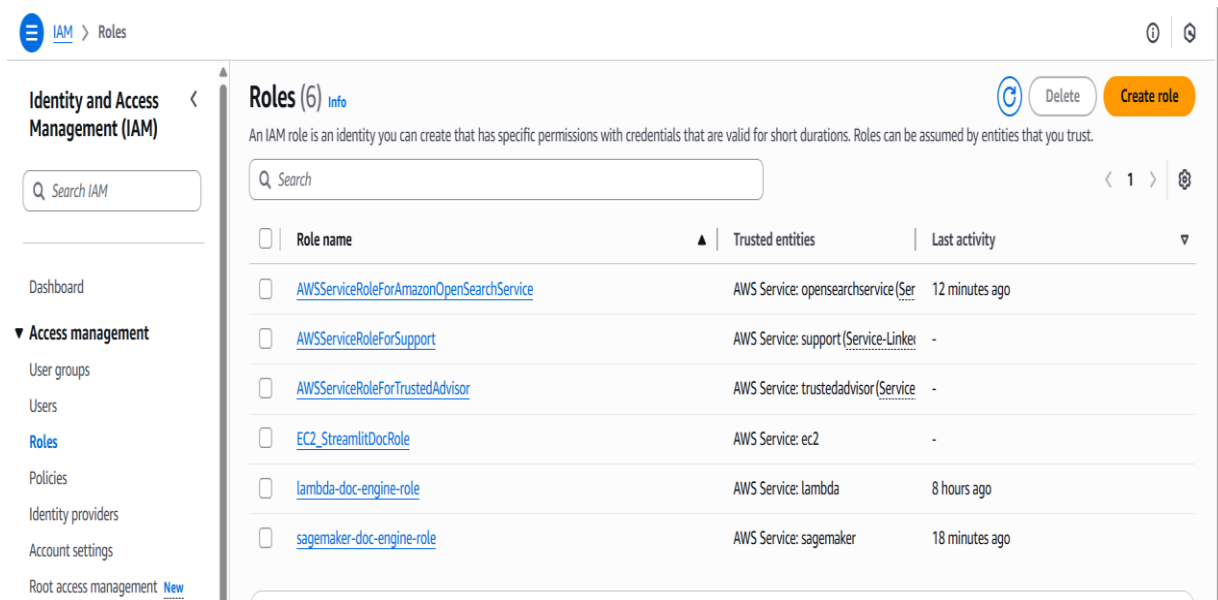
Field type specifies how each field is stored under the index mapping structure. Each level of mapping structure are separated by a dot.

Field	Field type
doc.embedding	knn_vector
doc.fileName	keyword
doc.text	text

-  Streamlit UI answering questions.



-  IAM Roles configuration



## Hosting the Streamlit App on AWS EC2

To make the application accessible publicly and ensure scalability, I deployed the Streamlit-based Q&A interface on an **Amazon EC2 instance** using a **Linux AMI (Amazon Machine Image)** with the **Free Tier t2.micro** configuration.

### EC2 Configuration Details:

Configuration	Value
AMI	Amazon Linux 2
Instance Type	t2.micro (Free Tier eligible)
Storage	8 GB EBS (General Purpose SSD)
Security Group	Allowed inbound traffic on ports <b>8501</b> (Streamlit), <b>22</b> (SSH), and optionally <b>80</b> for HTTP
Key Pair	Generated and used for SSH access securely
Public IP Assignment	Enabled (auto-assigned)

---

### App Deployment Steps on EC2

Here's how I deployed the Streamlit application:

1. **SSH into the EC2 instance:**

```
ssh -i my-key.pem ec2-user@your-ec2-public-ip
```

2. **Installed required system packages:**

```
sudo yum update -y
```

```
sudo yum install python3 git -y
```

**3. Created a virtual environment:**

```
python3 -m venv venv
```

```
source venv/bin/activate
```

**4. Cloned the GitHub repo containing my Streamlit app:**

```
git clone https://github.com/rishit911/document_engine_proj.git
```

**5. Installed Python dependencies:**

```
pip install -r requirements.txt
```

**6. Configured AWS credentials using aws configure to authenticate with SageMaker, OpenSearch, and S3.**

**7. Ran the Streamlit application:**

```
streamlit run app.py --server.port 8501 --server.enableCORS false
```

**8. Kept the Streamlit app running persistently using nohup:**

```
nohup streamlit run app.py &
```

**9. Accessed the app via browser using:**

```
http://3.110.245.87:8501
```

## **Deployment Observations (EC2 Hosting Experience)**

**After developing and testing the complete system locally, I deployed the Streamlit app to an AWS EC2 instance for public access. This phase added a new layer of complexity and learning.**

### **EC2 Instance Configuration**

- **Instance Type: t2.micro (AWS Free Tier)**
- **AMI: Amazon Linux 2023 AMI**



- **Storage Volume: Initially 8 GiB (default)**
  - **Security Group: Opened inbound rules for port 8501 (Streamlit) and 22 (SSH)**
  - **Dependencies Installed:**
    - **Python 3.10**
    - **streamlit, boto3, opensearch-py, sentence-transformers, requests-aws4auth**
    - **Model downloaded from S3 inside EC2 at runtime**
- 

### **Issue #1: Cold Start Latency**

**On each EC2 boot, downloading the SentenceTransformer model from S3, initializing the OpenSearch client with SigV4, and loading dependencies led to a cold start delay (30–50 seconds). This was acceptable for light usage, but for faster production-grade deployments, the following improvements are recommended:**

- **Cache model in EBS volume between reboots**
  - **Containerize and pre-load using Docker on boot**
  - **Use AWS Lambda with provisioned concurrency (if architecture is serverless)**
- 

### **Issue #2: Default Volume Too Small**

**When installing large packages like sentence-transformers, torch, and transformers, I encountered No space left on device errors due to the default 8 GiB volume limit.**

 **Resolution:**

- **Stopped EC2**
- **Went to Volumes > Actions > Modify Volume**
- **Increased size to 20 GiB**
- **Rebooted EC2 and confirmed new size with `df -h`**

## **Personal Learnings & Project Reflections**

### **What I Learned**

Working on this project was a deep dive into real-world **cloud engineering, MLOps, and AI automation**. It helped me:

- ☒ Understand **end-to-end data pipelines** from ingestion (S3) to processing (Lambda & Textract) to storage (DynamoDB) and retrieval (OpenSearch).
  - ☒ Gain hands-on experience with **semantic search** using vector databases and **SageMaker** model serving.
  - ☒ Learn to orchestrate multiple AWS services securely using **IAM roles, boto3, and SigV4 authentication**.
  - ☒ Improve my debugging and deployment skills — from local dev on Streamlit to hosting on **EC2 (free-tier t2.micro)**.
  - ☒ Structure and document a scalable, modular, and production-grade cloud-native application.
-

## Mistakes I Made (and Fixed)

- ✗ **Initial Misconfigurations** in OpenSearch domain endpoint caused timeout errors — I learned how to use correct hostname syntax and adjusted connection retries.
  - ✗ I tried to **load heavy ML models inside Lambda**, which exceeded size limits. I pivoted to **SageMaker inference endpoints** for scalability.
  - ✗ **Model saving/loading errors** (due to meta tensors and PyTorch 2.x issues) taught me the value of version compatibility and lazy loading models from S3 dynamically.
  - ✗ Initially added dropdown document selection in Streamlit — later realized **aggregating search across all documents** was more useful and user-friendly.
- 

## How This Project Prepares Me for a Cloud Engineer Role

- Showcases my **ability to integrate multiple AWS services** into a functional, cloud-native solution.
- Demonstrates my understanding of **event-driven architectures** using Lambda and Textract.
- Proves my skills in **deploying scalable apps on EC2**, including dependency handling, model loading, and real-time querying.
- Highlights **secure authentication practices** using AWS4Auth and boto3.Session.
- Combines **machine learning, data engineering, and cloud operations** in a unified solution.