

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.

The screenshot shows the AWS S3 console with the path: Amazon S3 > Buckets > doc-engine-bucket-risbur. The main content area displays a message: "No data events" and "No data events to display." Below this is a button labeled "Configure in CloudTrail".

Event notifications (1)

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

| Name | Event types | Filters | Destination type | Destination |
|-----------------------|-------------|---------|------------------|----------------------|
| trigger-lambda-upload | Put | - | Lambda function | doc-upload-processor |

Amazon EventBridge

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

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

Transfer acceleration

Disabled

- Upload event from S3 triggering Lambda.

The screenshot shows the AWS CloudWatch Logs console with the path: CloudWatch > Log groups > /aws/lambda/doc-upload-processor > 2025/06/19/[LATEST]30b301ffe2a74de8a97d69f1e3b3711a. The left sidebar shows navigation links for CloudWatch, Favorites and recents, Dashboards, AI Operations, Alarms, Logs (Log groups, Log Anomalies, Live Tail, Logs Insights), and Metrics (All metrics, Explorer).

Log events

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](#)

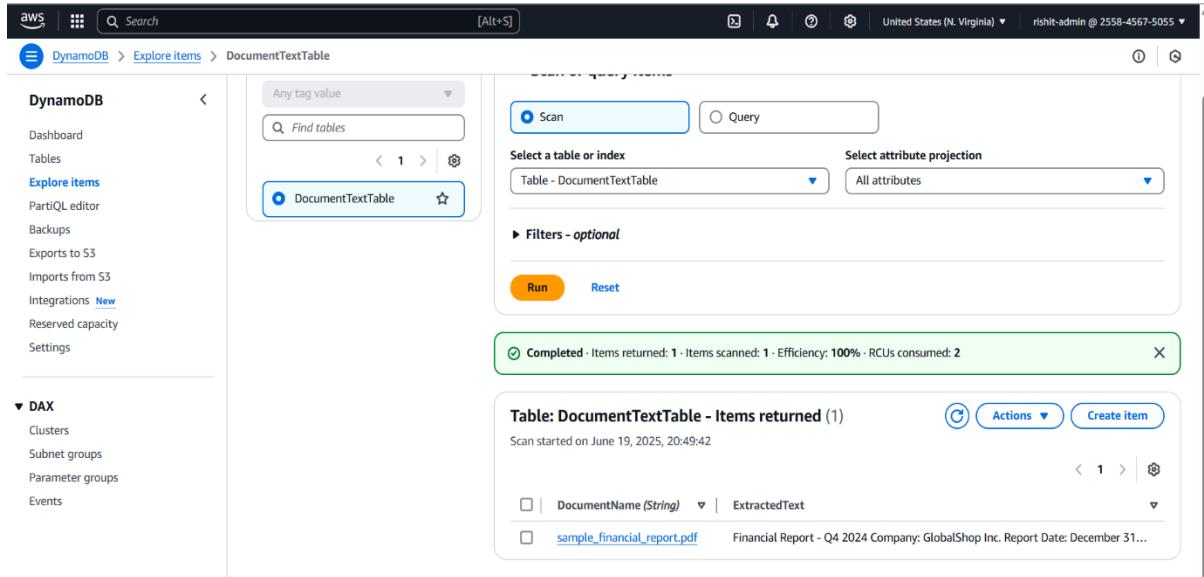
| Timestamp | Message |
|--------------------------|--|
| 2025-06-19T14:15:14.773Z | INIT_START Runtime Version: python:3.10.v78 Runtime Version ARN: arn:aws:lambda:us-east-1:runtime:b556158cad85934b6c377a5efb9a60... |
| 2025-06-19T14:15:15.064Z | START RequestId: 5c62ef22-f576-4bb1-b5e2-19e2389a043b 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 |

- Extracted Text from CloudWatch logs.

Part 2: Embedding Generation & Semantic Indexing

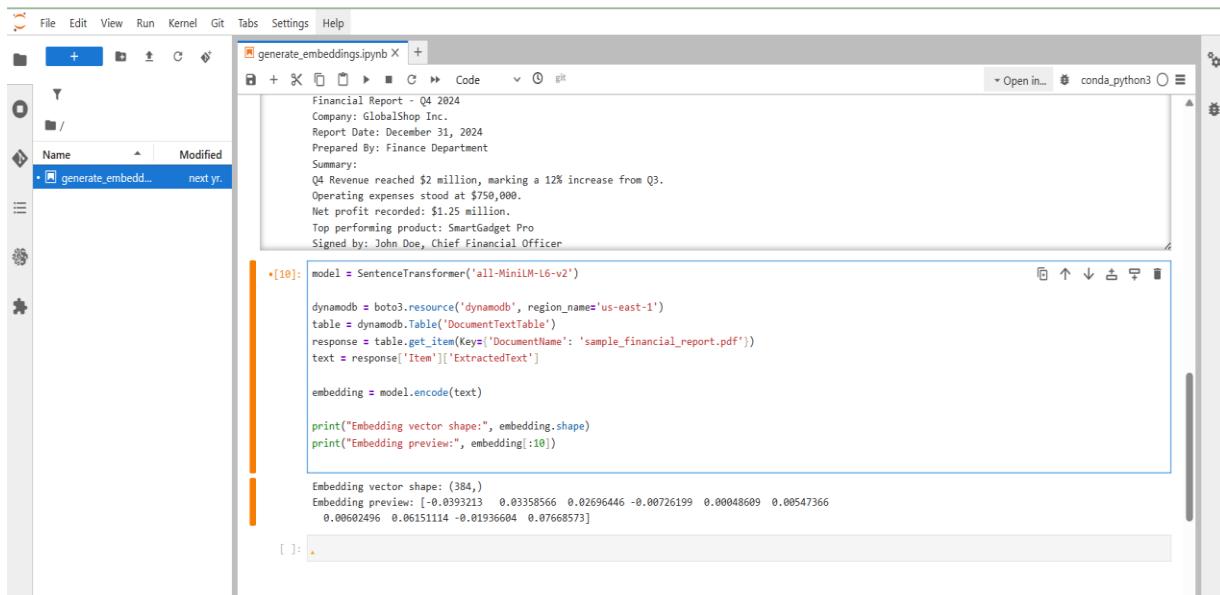
Tool: Jupyter Notebook (generate_embeddings.ipynb)

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



The screenshot shows the AWS DynamoDB console. On the left, there's a sidebar with 'Explore items' selected. The main area shows a table named 'DocumentTextTable'. A scan operation is in progress, returning 1 item. The item details show a document named 'sample_financial_report.pdf' with its extracted text being 'Financial Report - Q4 2024 Company: GlobalShop Inc. Report Date: December 31...'. There are buttons for 'Actions' and 'Create item'.

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



```
model = SentenceTransformer('all-MiniLM-L6-v2')

dynamodb = boto3.resource('dynamodb', region_name='us-east-1')
table = dynamodb.Table('DocumentTextTable')
response = table.get_item(Key={'DocumentName': 'sample_financial_report.pdf'})
text = response['Item']['ExtractedText']

embedding = model.encode(text)

print("Embedding vector shape:", embedding.shape)
print("Embedding preview:", embedding[:10])
```

The screenshot shows a Jupyter Notebook cell with the code provided. The output shows the embedding vector shape as (384,) and a preview of the first 10 elements of the vector.

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 single code cell. The code performs the following steps:

- Imports necessary modules: `os` and `openai`.
- Loads a saved model from a file named `model.pkl`.
- Defines a function `generate_embedding` that takes a query string and returns its embedding using the loaded model.
- Creates a search body for an OpenSearch query. The query uses the KNN search type with a size of 1, a query embedding, and a k value of 1.
- Searches the `documents-index` for the query "What was the net profit in Q4?"
- Prints the retrieved document's source and text.

```
[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'])
```

Output of 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.

The screenshot shows the Amazon OpenSearch Service console with the path: Amazon OpenSearch Service > Domains > doc-engine-domain > indices > documents-index.

The left sidebar shows navigation links for Serverless, Ingestion, and Integrations.

The main area displays the `documents-index` configuration:

- Index information:**
 - Name: documents-index
 - Document count: 1
 - Document size (byte): 17.64 KiB
 - Query total: 105
 - Mapping type: properties
- Mapping structure:** Shows fields: doc, embedding, fileName, text.
- Field mappings (3):**
 - Field: doc.embedding, Field type: knn_vector
 - Field: doc.fileName, Field type: keyword
 - Field: doc.text, Field type: text

🔍 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.

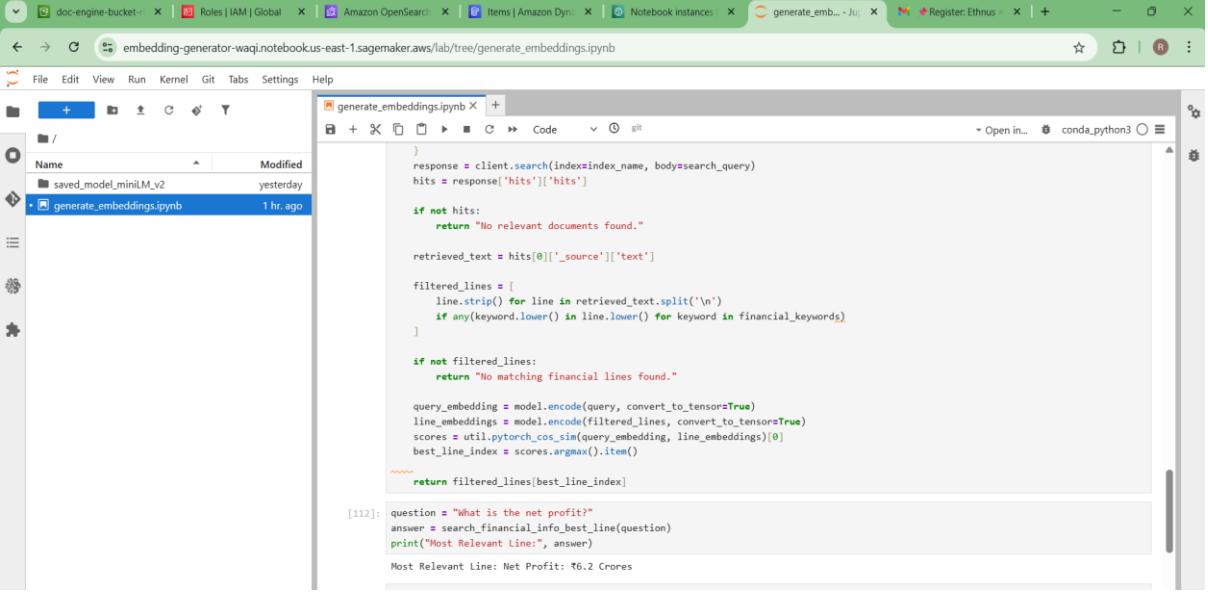
Advantage over keyword search:

- 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 console interface. At the top, there is a navigation bar with tabs: Instance health, Off-peak window, Auto-Tune, Logs, **Indexes**, Tags, Connections - new, VPC endpoints, and Packages. The 'Indexes' tab is currently selected. Below the navigation bar, the title 'Indexes (4)' is displayed, followed by a sub-instruction: 'Indexing is the method by which search engines organize data for fast retrieval. Before you can search data, you must index it.' A 'Learn more' link is provided. A search bar labeled 'Find indexes' is present. To the right of the search bar are navigation icons for back, forward, and refresh. The main content area displays a table with the following data:

| Index | ▲ Document count | ▼ Size (byte) | ▼ Query total | ▼ Mapping type | ▼ Field mappings |
|--|--------------------|-----------------|-----------------|-------------------------|--------------------|
| _kibana_1 | 1 | 5.21 KiB | 52 | dynamic,_meta,proper... | 128 |
| opensearch-observability | 0 | 208.00 B | 0 | dynamic,properties | 11 |
| plugins-ml-config | 1 | 3.94 KiB | 1 | _meta,properties | 6 |
| documents-index | 1 | 17.64 KiB | 105 | properties | 3 |

- Search query payload and matching result.



The screenshot shows a Jupyter Notebook interface with several tabs at the top: 'doc-engine-bucket...', 'Roles | IAM | Global', 'Amazon OpenSearch', 'Items | Amazon Dyn...', 'Notebook instance...', 'generate_embeddings.ipynb', 'Register: Ethnus...', and '+'. The main area displays a Python script named 'generate_embeddings.ipynb'.

```

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]

```

In the code editor, there is a line [112]: followed by:

```

question = "What is the net profit?"
answer = search.financial_info_best_line(question)
print("Most Relevant Line:", answer)

```

The output pane shows the result of the last print statement:

```

Most Relevant Line: Net Profit: ₹6.2 Crores

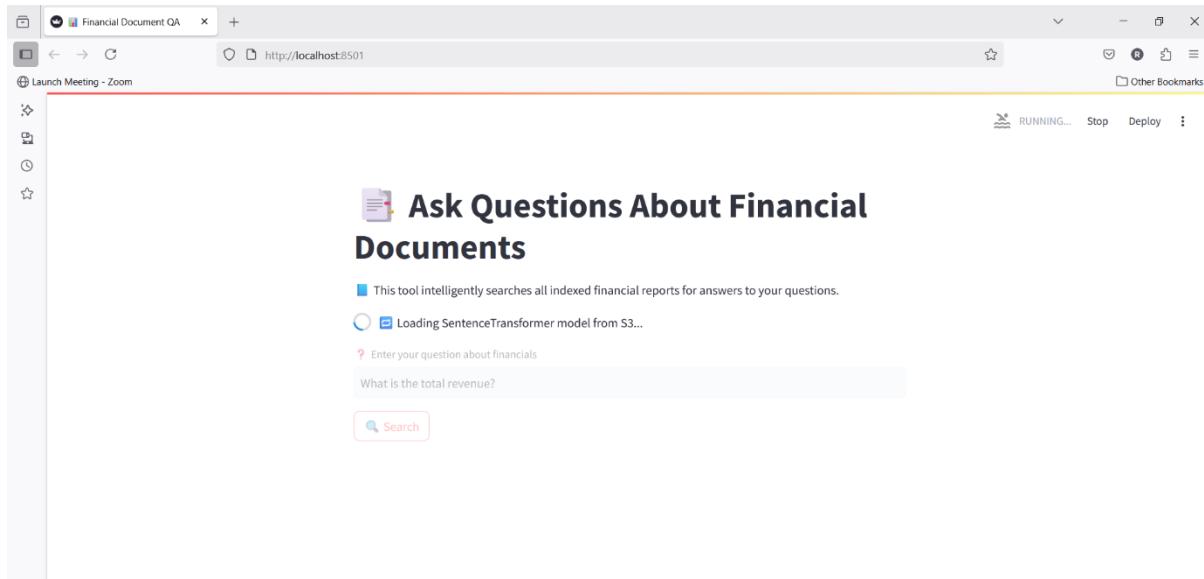
```

Streamlit Frontend App

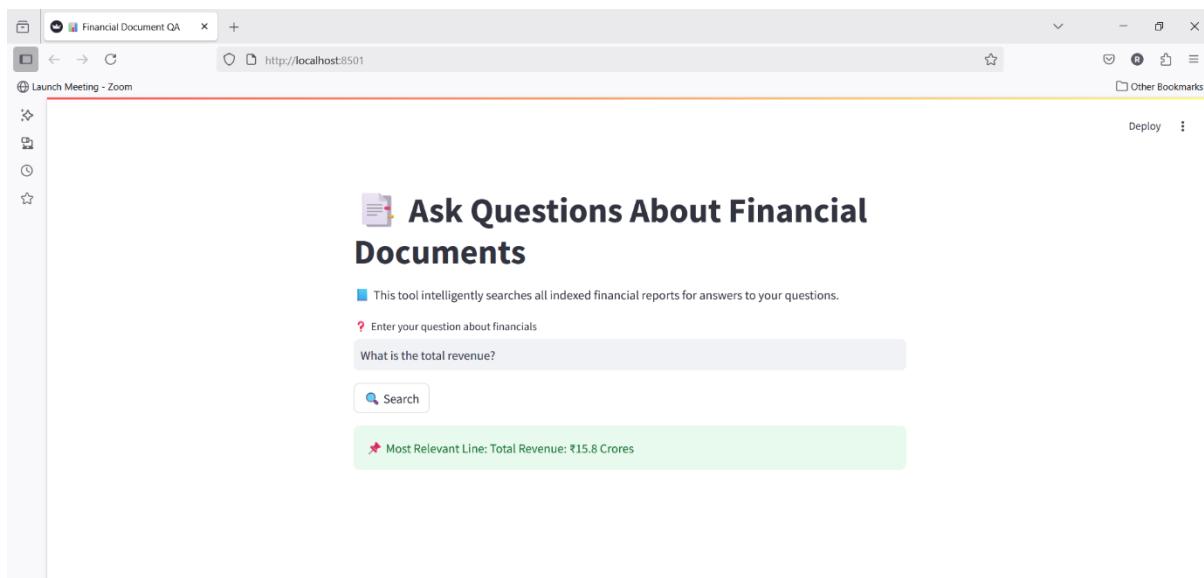
Functionality:

- 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**.

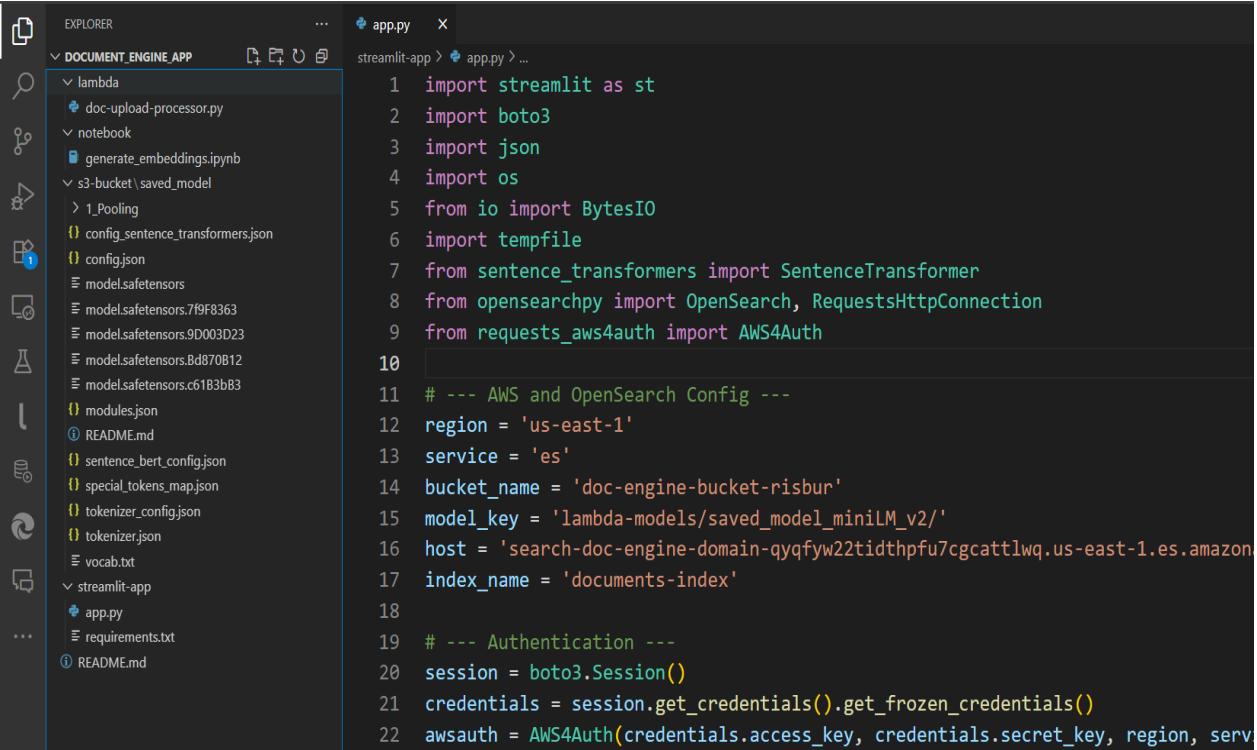
- Streamlit interface with user question.



- Output with the most relevant line highlighted.



Project Structure



The screenshot shows a code editor interface with two main panes. The left pane, titled 'EXPLORER', displays the project structure of 'DOCUMENT_ENGINE_APP'. It includes a 'lambda' folder containing 'doc-upload-processor.py', a 'notebook' folder with 'generate_embeddings.ipynb', an 's3-bucket\saved_model' folder with '1_Pooling', 'config_sentence_transformers.json', 'config.json', 'model.safetensors', 'model.safetensors.7f9F8363', 'model.safetensors.9d003D23', 'model.safetensors.Bd870B12', 'model.safetensors.c61B3bB3', 'modules.json', 'README.md', 'sentence_bert_config.json', 'special_tokens_map.json', 'tokenizer_config.json', 'tokenizer.json', and 'vocab.txt'. There's also a 'streamlit-app' folder with 'app.py' and 'requirements.txt', and a 'README.md' file. The right pane shows the content of 'app.py' with syntax highlighting for Python code.

```
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-qyqfyw22tidthpfu7cgcattlwq.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, service)
```

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

bash

CopyEdit

streamlit run app.py

- ✓ S3 document upload interface.

The screenshot shows the AWS S3 "Upload: status" page. At the top, there's a summary section with a note about navigating away. Below it, the "Summary" section shows a destination of "s3://doc-engine-bucket-risbur". It lists one succeeded file ("sample_financial_report.pdf") at 1.2 KB (100.00%) and zero failed files. Under "Files and folders", there's a table showing the uploaded file. The table has columns for Name, Folder, Type, Size, Status, and Error. The file "sample_financial_report.pdf" is listed with a status of "Succeeded".

| Name | Folder | Type | Size | Status | Error |
|-----------------------------|--------|-----------------|--------|-----------|-------|
| sample_financial_report.pdf | - | application/pdf | 1.2 KB | Succeeded | - |

- ✓ Lambda execution log showing Textract.

The screenshot shows the AWS CloudWatch "Log events" page for a log group under the "/aws/lambda/doc-upload-processor" namespace. The log entries are timestamped and show the execution of a Lambda function, including the initiation of the runtime, the start request ID, and the execution of the Textract API to extract financial report text. The logs also mention Q4 2024 revenue and operating expenses.

| 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 interface. On the left, there's a sidebar with navigation links like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Under the 'Explore items' section, 'DocumentTextTable' is selected. The main area has a search bar at the top with 'Find tables' and a dropdown for 'Any tag value'. Below it, there are two buttons: 'Scan' (which is highlighted) and 'Query'. A dropdown menu for 'Select a table or index' shows 'Table - DocumentTextTable' and another for 'Select attribute projection' showing 'All attributes'. There's also a 'Filters - optional' section with a 'Run' button and a 'Reset' button. At the bottom, a green message box says 'Completed · Items returned: 1 · Items scanned: 1 · Efficiency: 100% · RCU consumed: 2'. The main table view shows a single row with a checkbox, 'DocumentName (String)', and 'ExtractedText'. The 'ExtractedText' column contains the value 'sample_financial_report.pdf' and 'Financial Report - Q4 2024 Company: GlobalShop Inc. Report Date: December 31...'. There are also 'Actions' and 'Create item' buttons above the table.

- ✓ SageMaker embedding generation outputs.

```

File Edit View Run Kernel Git Tabs Settings Help
generate_embeddings.ipynb + + X C git
Name Modified
saved_model_minilm_v2 yesterday
generate_embeddings.ipynb 1 hr ago
print(text[:500])
Sample Financial Report - Q1 2025
Company: FinInova 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
 0.01283891 0.18468467 -0.00113416 0.02123569]
[104]: model.save('saved_model_minilm_v2')

```

- OpenSearch dashboard (index + knn).

The screenshot shows the Amazon OpenSearch Service dashboard. The left sidebar has sections for Serverless (Dashboard, Collections, Security, Authentication, Data access policies, Encryption policies, Network policies, Data lifecycle policies, VPC endpoints) and Ingestion (Dashboard, Pipelines). The main area is titled 'documents-index' and contains 'Index information' (Name: documents-index, Document count: 1, Document size (byte): 17.64 kB, Query total: 105, Mapping type: properties), 'Mapping structure' (doc, embedding, fileName, text), and 'Field mappings (3)' (Field: doc.embedding, Field type: knn_vector; Field: doc.fileName, Field type: keyword; Field: doc.text, Field type: text).

- Streamlit UI answering questions.

The screenshot shows a Streamlit application window titled 'Financial Document QA'. The browser address bar shows 'http://localhost:8501'. The Streamlit app has a sidebar with icons for file, database, clock, and star. The main content area has a title 'Ask Questions About Financial Documents' with a document icon. Below it, a message says 'This tool intelligently searches all indexed financial reports for answers to your questions.' A question input field contains 'What is the total revenue?'. A search button with a magnifying glass icon is below the input field. A green success message at the bottom says '★ Most Relevant Line: Total Revenue: ₹15.8 Crores'.

- IAM Roles configuration

The screenshot shows the AWS IAM Roles configuration page. On the left, there's a sidebar with navigation links for Identity and Access Management (IAM), including Dashboard, Access management (User groups, Users, Roles, Policies, Identity providers, Account settings), and Root access management. The main area is titled "Roles (6) Info" and contains a table with the following data:

| Role name | Trusted entities | Last activity |
|--|---|----------------|
| AWSServiceRoleForAmazonOpenSearchService | AWS Service: opensearchservice (Service-Linker) | 12 minutes ago |
| AWSServiceRoleForSupport | AWS Service: support (Service-Linker) | - |
| AWSServiceRoleForTrustedAdvisor | AWS Service: trustedadvisor (Service-Linker) | - |
| EC2_StreamlitDocRole | AWS Service: ec2 | - |
| lambda-doc-engine-role | AWS Service: lambda | 8 hours ago |
| sagemaker-doc-engine-role | AWS Service: sagemaker | 18 minutes ago |

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 8501 (Streamlit), 22 (SSH), and optionally 80 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 --server.enableXsrfProtection 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
```

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.