

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	doc-upload-processor

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](#) [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

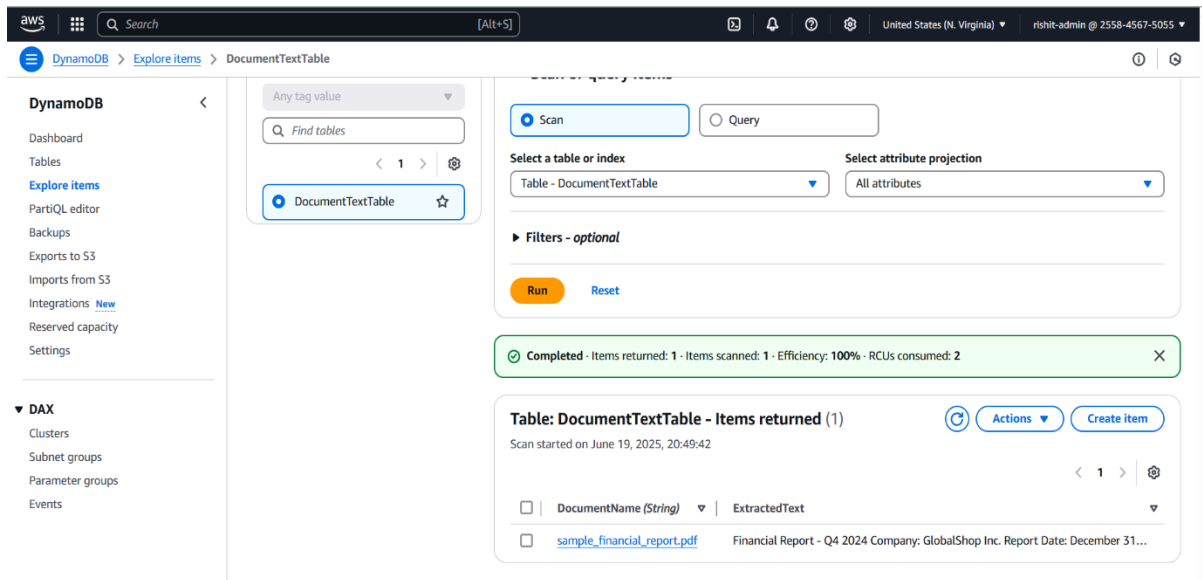
[Back to top](#)

- Extracted Textract output from CloudWatch logs.

Part 2: Embedding Generation & Semantic Indexing

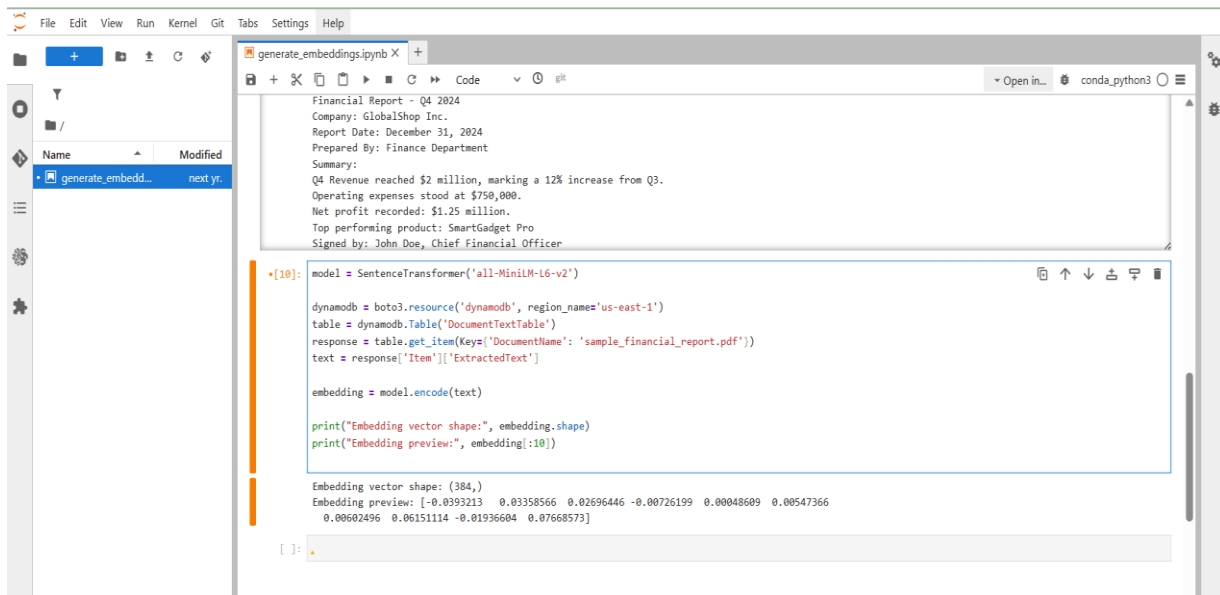
Tool: Jupyter Notebook (generate_embeddings.ipynb)

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



The screenshot shows the AWS DynamoDB console interface. On the left is a navigation menu with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area displays the 'DocumentTextTable' with a 'Scan' button selected. Below the scan controls, a green status bar indicates 'Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCUs consumed: 2'. A table titled 'Table: DocumentTextTable - Items returned (1)' shows the scan results, including a document named 'sample_financial_report.pdf' with its extracted text.

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



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:

```
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

[*]: 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']

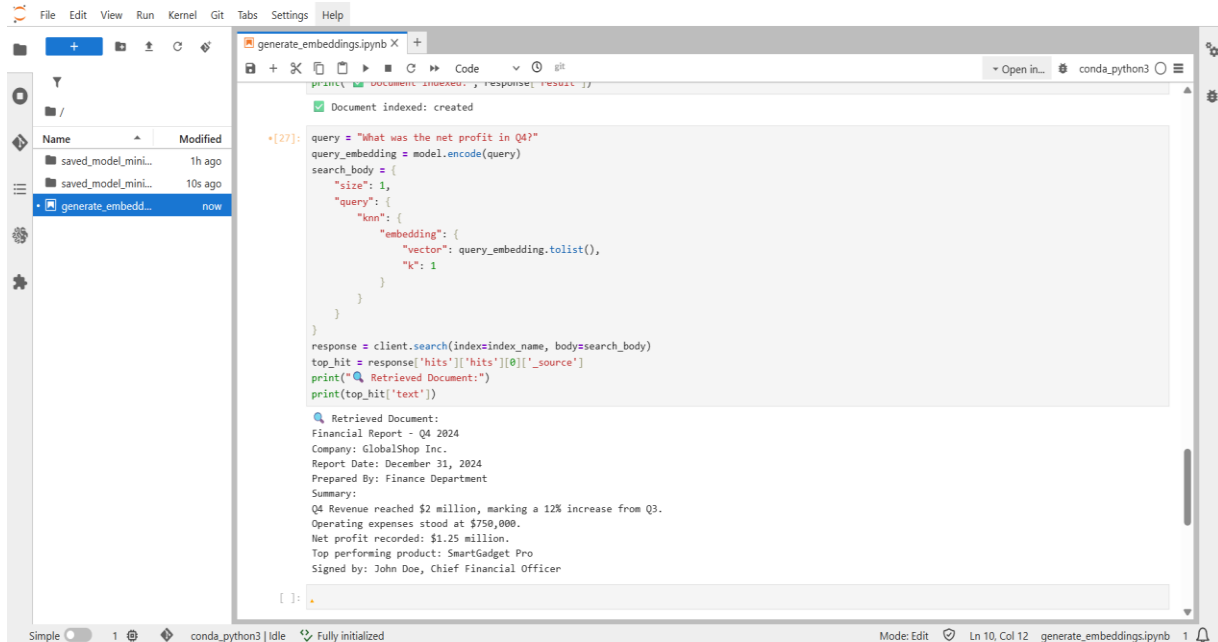
embedding = model.encode(text)

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

Embedding vector shape: (384,)
Embedding preview: [-0.0393213  0.03358566  0.02696446 -0.00726199  0.00048609  0.00547366
  0.00602496  0.06151114 -0.01936604  0.07668573]
```

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:

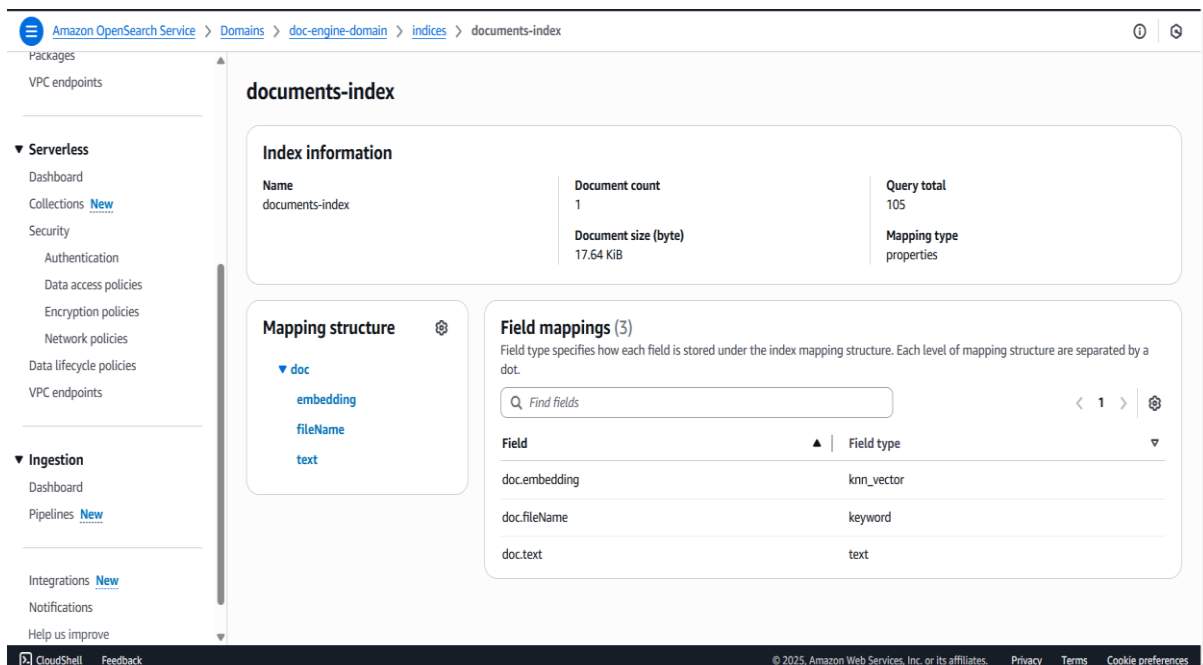
```
print("DOCUMENT INDEXED", response["result"])
# Document indexed: created

* [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.

<

Instance health

Off-peak window

Auto-Tune

Logs

Indexes

Tags

Connections - new

VPC endpoints

Packages

>

Indexes (4)

Indexing is the method by which search engines organize data for fast retrieval. Before you can search data, you must index it. [Learn more](#)

Find indexes

<

1

>

Index	Document count	Size (byte)	Query total	Mapping type	Field mappings
.kibana_1	1	5.21 KiB	52	dynamic,_meta,prope...	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 titled 'generate_embeddings.ipynb' in a web browser. The left sidebar displays a file explorer with 'saved_model_miniLM_v2' and 'generate_embeddings.ipynb'. The main area shows the notebook's code and output. The code defines a function to search for relevant financial lines based on a query. The output shows the result of a search for 'What is the net profit?'.

```

}
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]

[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

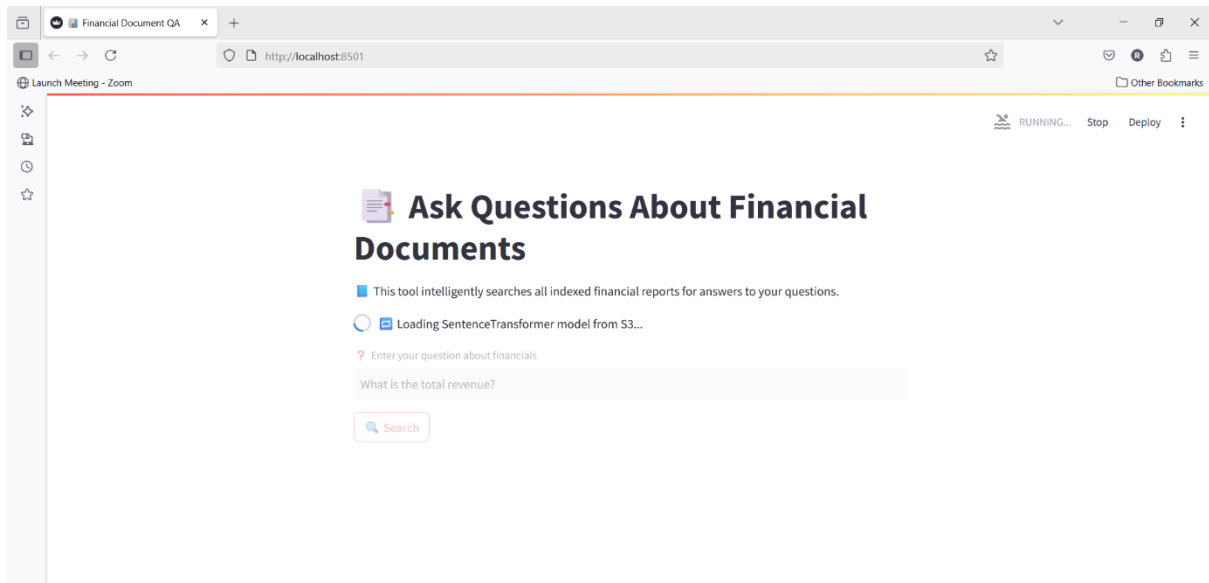
```

Streamlit Frontend App

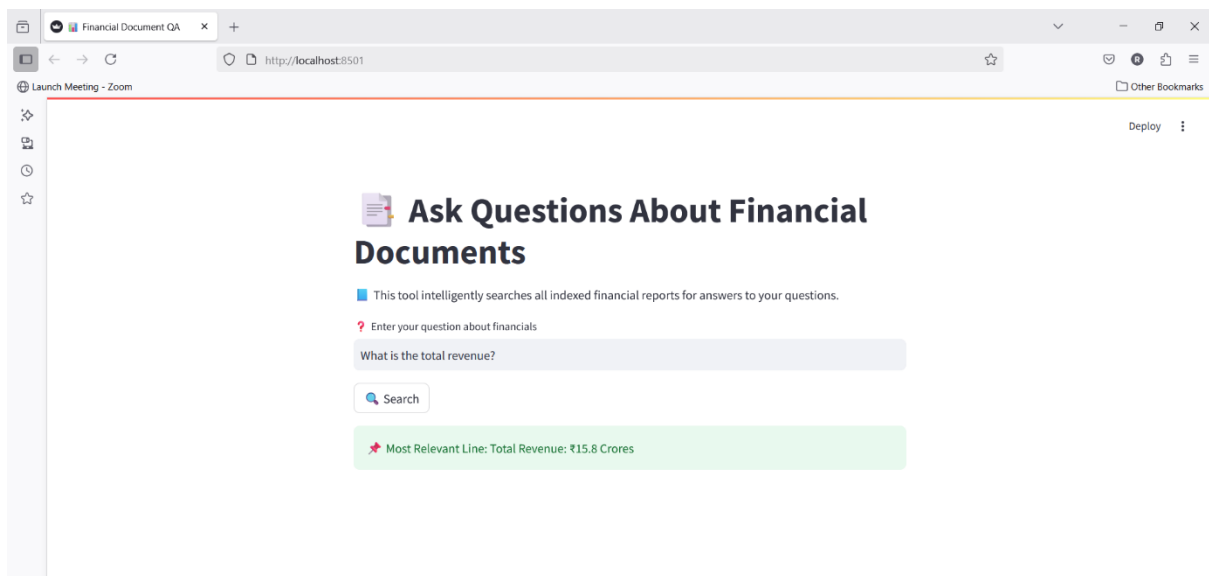
Functionality:

- 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**.

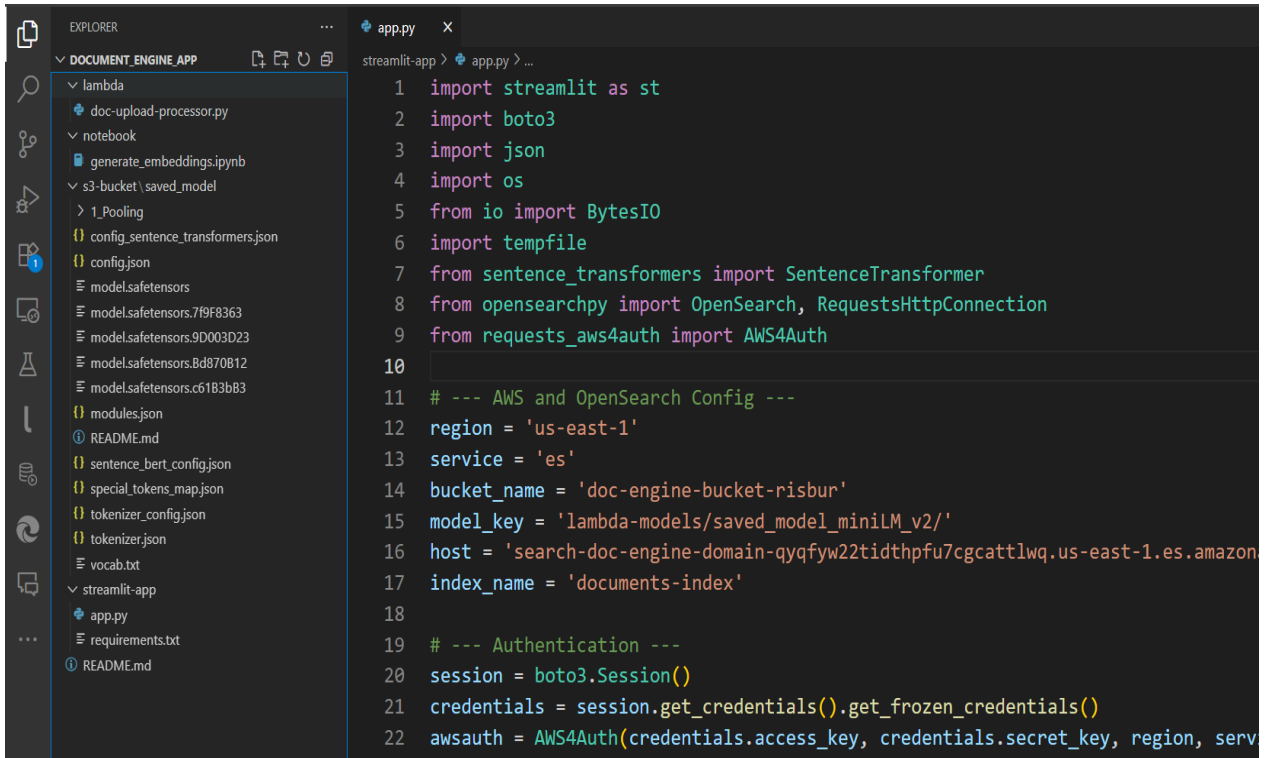
- Streamlit interface with user question.



- Output with the most relevant line highlighted.



Project Structure



The screenshot shows the VS Code interface. The Explorer panel on the left displays the project structure for 'DOCUMENT_ENGINE_APP'. It includes a 'lambda' folder with 'doc-upload-processor.py', a 'notebook' folder with 'generate_embeddings.ipynb', and an 's3-bucket' folder containing a 'saved_model' subfolder. The 'saved_model' folder contains a '1_Pooling' subfolder and several files: 'config_sentence_transformers.json', 'config.json', 'model.safetensors', 'model.safetensors.7f9f8363', 'model.safetensors.9D003D23', 'model.safetensors.Bd870B12', 'model.safetensors.c6183b83', 'modules.json', 'README.md', 'sentence_bert_config.json', 'special_tokens_map.json', 'tokenizer_config.json', 'tokenizer.json', and 'vocab.txt'. Below these are 'streamlit-app', 'app.py', 'requirements.txt', and another 'README.md'. The main editor shows the 'app.py' file with the following 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, 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

bash

CopyEdit

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 page title 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" tab 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 execution log showing Textract.

The screenshot shows the AWS CloudWatch console. The left sidebar contains navigation links for CloudWatch, Log groups, Alarms, Logs, and Metrics. The main content area is titled "Log events" and shows a list of log events for the Lambda function "doc-upload-processor". The events are filtered by the time range "2025/06/19/[\$LATEST]30b301ffe2a74de8a97d69f1e3b3711a". The events are displayed in a table with columns for Timestamp and Message. The messages show the execution details of the Lambda function, including the runtime version, the request ID, the event source, and the extracted text from the document. The events are: INIT_START, START, Event, Extracted text, Company, Report Date, Prepared By, Summary, Q4 Revenue, Operating expenses, Net profit, and Top performing product.

- ✓ DynamoDB console showing text records.

The screenshot shows the AWS DynamoDB console interface. On the left is a navigation menu with options like Dashboard, Tables, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. The main area displays the 'DocumentTextTable' with a 'Scan' operation selected. A status bar indicates 'Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCUs consumed: 2'. Below this, a table titled 'Table: DocumentTextTable - Items returned (1)' shows a single record with 'DocumentName (String)' as 'sample_financial_report.pdf' and 'ExtractedText' as 'Financial Report - Q4 2024 Company: GlobalShop Inc. Report Date: December 31...'.

- ✓ SageMaker embedding generation outputs.

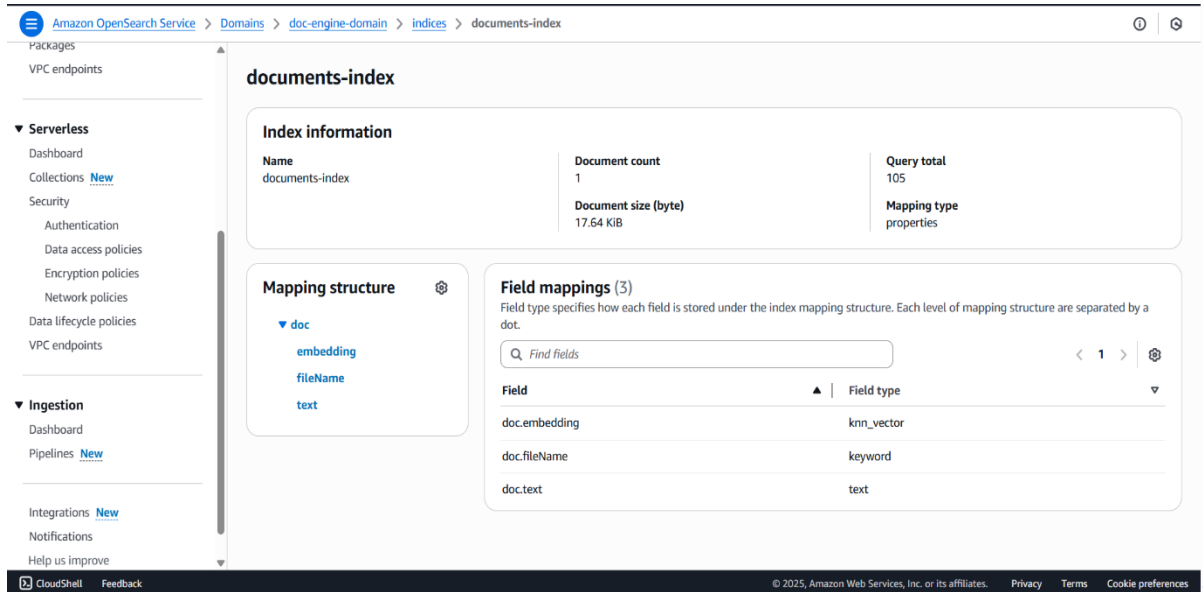
The screenshot shows the SageMaker JupyterLab interface. The left sidebar displays a file explorer with 'saved_model_minilm_v2' and 'generate_embeddings.ipynb'. The main area shows the 'generate_embeddings.ipynb' notebook with the following code and output:

```
[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]
```

The notebook also contains a text block with sample financial report data, including company name, report period, and summary highlights.

-  OpenSearch dashboard (index + knn).




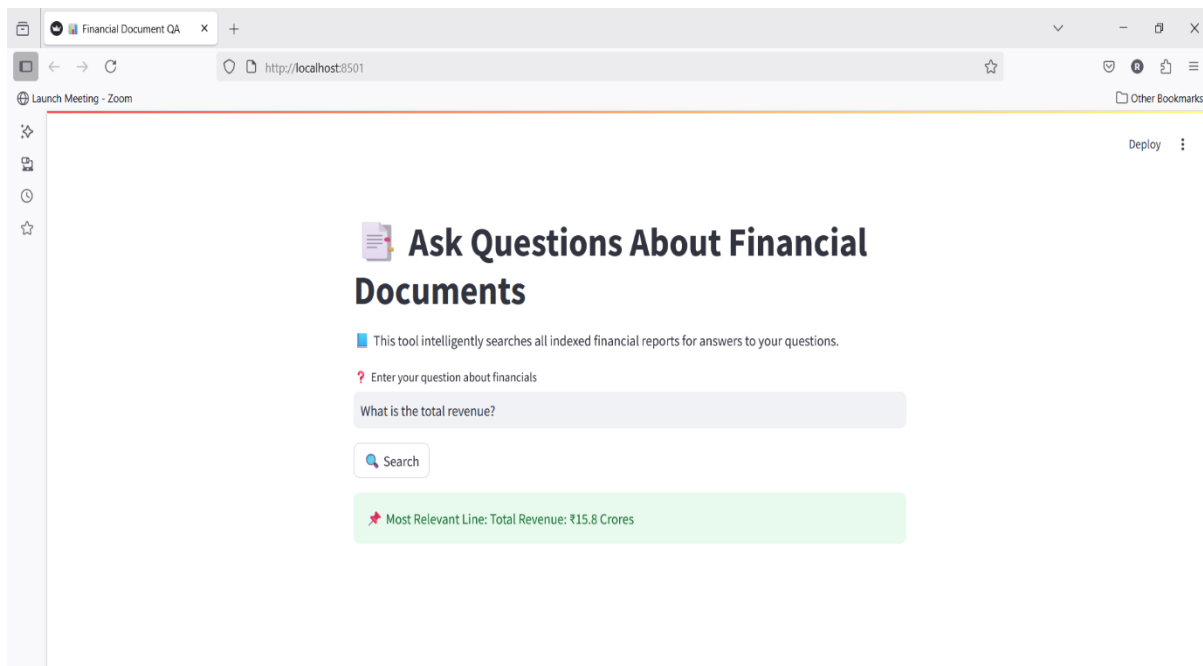
The screenshot shows the Amazon OpenSearch Service console for the 'documents-index'. The left sidebar contains navigation links for Packages, VPC endpoints, Serverless (Dashboard, Collections, Security), Ingestion (Dashboard, Pipelines), and Integrations. The main content area displays the index configuration:

- Index information:**
 - Name: documents-index
 - Document count: 1
 - Document size (byte): 17.64 KiB
 - Query total: 105
 - 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.



The screenshot shows a Streamlit web application titled 'Ask Questions About Financial Documents'. The interface includes a header, a description, a search input field, and a search button. The search results are displayed below the input field.

Ask Questions About Financial Documents

This tool intelligently searches all indexed financial reports for answers to your questions.

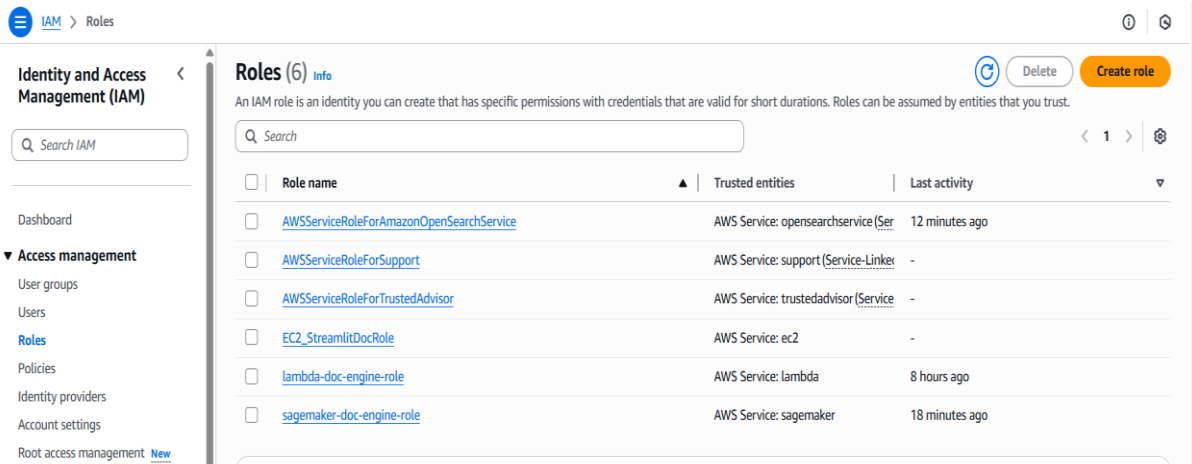
Enter your question about financials

What is the total revenue?

Search

Most Relevant Line: Total Revenue: ₹15.8 Crores

-  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 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.