

Assignment 6. MLOps with AWS

Project Setup

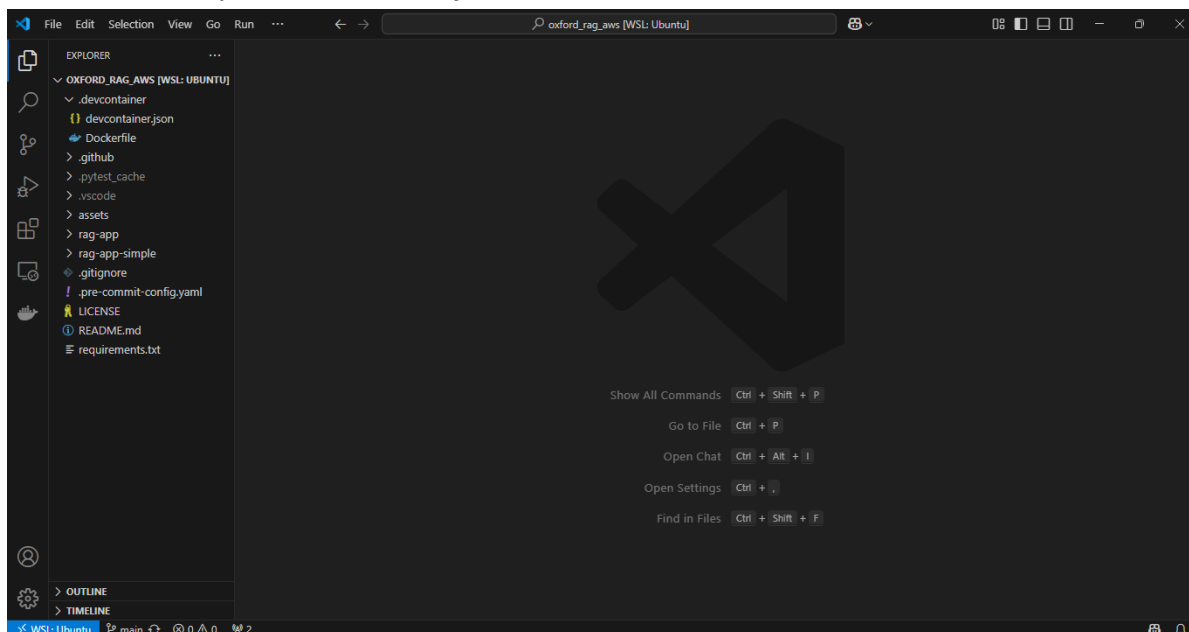
Clone project repo:

I forked the original repo into my GitHub account ([repo link](#))

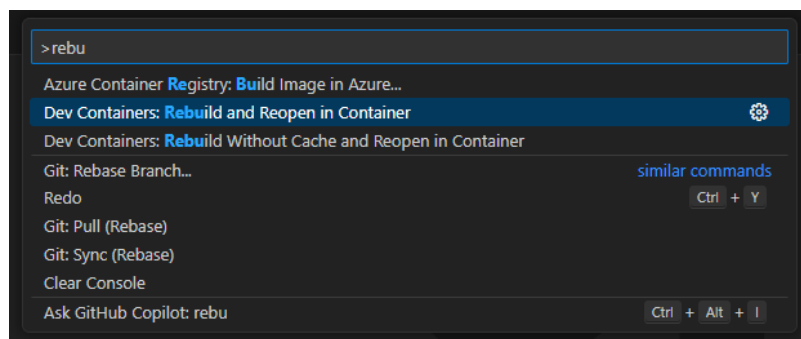
Then cloned it locally

```
git clone https://github.com/MohmedMonsef/oxford-genai-llmops-project.git
```

And now, the repo is cloned to my local machine:

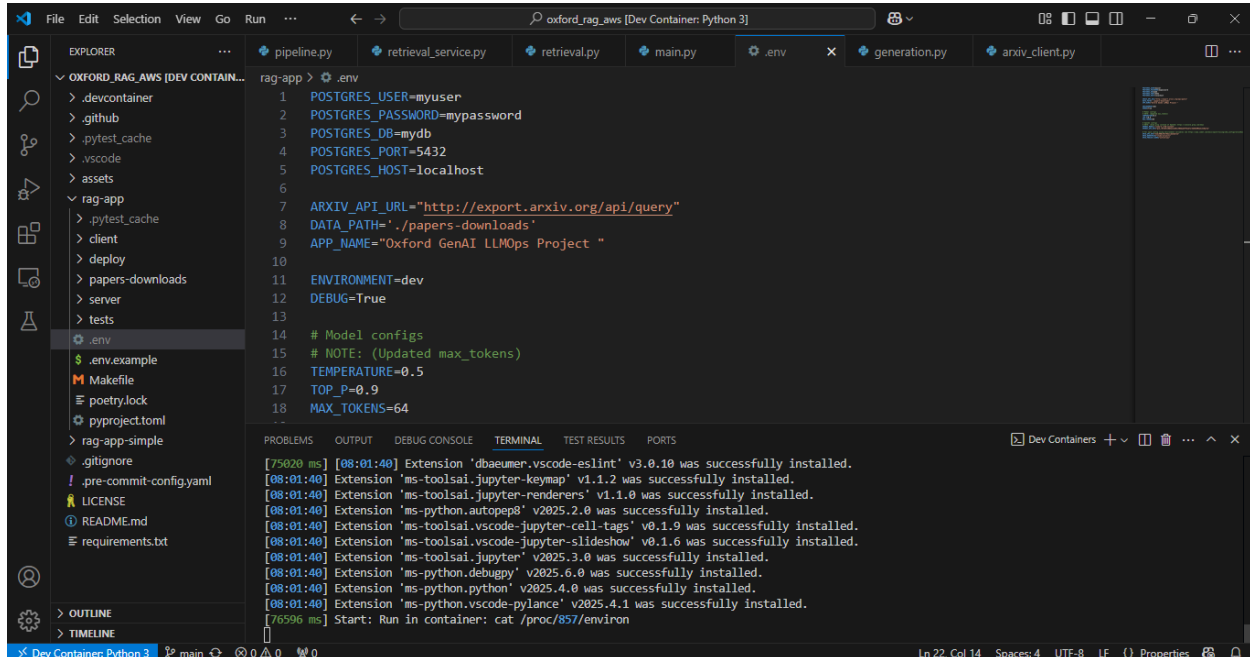


Build the Dev Container

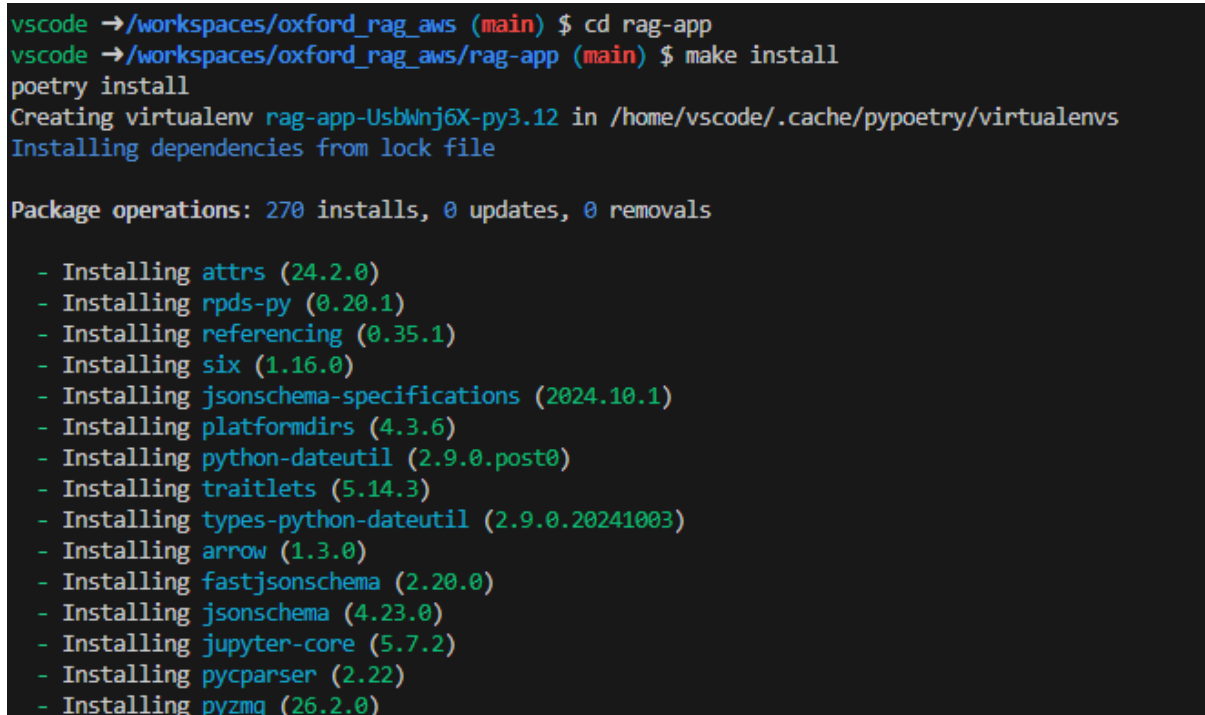


Setup .env file

Create the .env file and complete missing environment variables related to OpenAI and Comet Opik



Install dependencies



Run the application

```
o vscode →/workspaces/oxford_rag_aws/rag-app (main) $ make run-app
poetry run bash -c 'PYTHONPATH=./server/src uvicorn server.src.main:app --reload'
INFO: Will watch for changes in these directories: ['/workspaces/oxford_rag_aws/rag-app']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [20335] using StatReload
/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj6X-py3.12/lib/python3.12/site-packages/transformers/tokenization_utils_base.py:1617: FutureWarning
: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be deprecated in transformers v4.45, and will be th
en set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj6X-py3.12/lib/python3.12/site-packages/pydantic/_internal/_config.py:341: UserWarning: Valid confi
g keys have changed in V2:
* 'orm_mode' has been renamed to 'from_attributes'
  warnings.warn(message, UserWarning)
/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj6X-py3.12/lib/python3.12/site-packages/pydantic/_internal/_config.py:341: UserWarning: Valid confi
g keys have changed in V2:
* 'orm_mode' has been renamed to 'from_attributes'
  warnings.warn(message, UserWarning)
INFO: Started server process [20340]
INFO: Waiting for application startup.
Spinning up lifespan context...
Configure opik...
OPIK: Opik is already configured. You can check the settings by viewing the config file at /home/vscode/.opik.config
Loading embedding model...
/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj6X-py3.12/lib/python3.12/site-packages/transformers/tokenization_utils_base.py:1617: FutureWarning
: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be deprecated in transformers v4.45, and will be th
en set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
INFO: Application startup complete.
```

Inspect the server

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS 1

Port		Forwarded Address	Running Process	Origin
8000		127.0.0.1:8001	/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj...	Auto Forwarded
<div>Add Port</div>				

```
← → ↺ ⓘ 127.0.0.1:8001
Pretty-print ☐
{"message":"Welcome to the RAG app!"}
```

GitHub Actions

We have defined the following GitHub workflow to:

- Print the required message
- Install dependencies via poetry
- Run unit tests

```
name: CI Initialise

on:
  push:
    branches:
      - '**'

jobs:
  initialise:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Print CI message
        run: echo "CI step initialising"

      - name: Setup python
        uses: actions/setup-python@v5
        with:
          python-version: '3.12'

      - name: Install poetry
        run: |
          curl -sSL https://install.python-poetry.org | python3 -
          echo "$HOME/.local/bin" >> $GITHUB_PATH

      - name: Cache poetry dependencies
        uses: actions/cache@v4
        with:
          path: |
            ~/.cache/pypoetry
            ~/.cache/pip
```

```
        key: poetry-${ runner.os }-${
hashFiles('**/poetry.lock') }}
        restore-keys: |
            poetry-${ runner.os }-

- name: Install dependencies
  run: |
    cd rag-app
    poetry install --no-root

- name: Run pytest
  run: |
    cd rag-app
    echo "${ secrets.DUMMY_ENV }}" > .env
    poetry run pytest
```

Note

The tests provided in the original repo were buggy and did not pass. We solved these issues in [this](#) commit.

Now, the tests are passing, and the CI workflow is running successfully.

26 workflow runs				Event ▾	Status ▾	Branch ▾	Actor ▾
✓	Modify: use llm-related papers to be the documents in th...	main	14 hours ago	...			
Makefile CI #13: Commit d27be56 pushed by MohmedMonsef				17s			
✓	Modify: use llm-related papers to be the documents in th...	main	14 hours ago	...			
CI Initialise #13: Commit d27be56 pushed by MohmedMonsef				2m 5s			

Summary

Jobs

initialise

Run details

Usage

Workflow file

initialise

succeeded 14 hours ago in 1m 59s

Search logs

> ✓ Set up job

> ✓ Checkout code

> ✓ Print CI message

> ✓ Setup python

> ✓ Install poetry

> ✓ Cache poetry dependencies

> ✓ Install dependencies

> ✓ Run pytest

> ✓ Post Cache poetry dependencies

> ✓ Post Setup python

> ✓ Post Checkout code

> ✓ Complete job

1s

0s

0s

1s

12s

1m 27s

6s

10s

0s

0s

0s

0s

```
✓ Run pytest 10s
1 ▶ Run cd rag-app
39 /home/runner/.cache/pypoetry/virtualenvs/rag-app-uhWz2AE6-py3.12/lib/python3.12/site-packages/pytest_asyncio/plugin.py:208:
PytestDeprecationWarning: The configuration option "asyncio_default_fixture_loop_scope" is unset.
40 The event loop scope for asynchronous fixtures will default to the fixture caching scope. Future versions of pytest-asyncio will
default the loop scope for asynchronous fixtures to function scope. Set the default fixture loop scope explicitly in order to avoid
unexpected behavior in the future. Valid fixture loop scopes are: "function", "class", "module", "package", "session"
41
42 warnings.warn(PytestDeprecationWarning(_DEFAULT_FIXTURE_LOOP_SCOPE_UNSET))
43 ===== test session starts =====
44 platform linux -- Python 3.12.10, pytest-8.3.3, pluggy-1.5.0
45 rootdir: /home/runner/work/oxford-genai-llmops-project/oxford-genai-llmops-project/rag-app
46 configfile: pyproject.toml
47 plugins: opik-1.0.4, cov-6.0.0, anyio-4.6.0, asyncio-0.24.0, Faker-30.8.2
48 asyncio: mode=Mode.STRICT, default_loop_scope=None
49 collected 6 items
50
51 tests/services/test_generation_service.py ..... [ 83%]
52 tests/services/test_retrieval_service.py . [100%]
53
54 ===== 6 passed in 5.02s =====
```

Database setup

Build the database

Outside the dev container.

```
$ cd rag-app
$ make build-db
docker compose --env-file .env -f deploy/docker/postgres/docker-compose.yaml up --build
WARN[0000] /home/monsef/projects/oxford_rag_aws/rag-app/deploy/docker/postgres/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion
Compose can now delegate builds to bake for better performance.
To do so, set COMPOSE_BAKE=true.
[+] Building 2.0s (11/11) FINISHED                                docker:default
=> [postgres internal] load build definition from pgvector2.Dockerfile      0.0s
=> => transferring dockerfile: 771B                                         0.0s
=> [postgres internal] load metadata for docker.io/library/postgres:alpine 1.7s
=> [postgres internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                              0.0s
=> [postgres builder 1/4] FROM docker.io/library/postgres:alpine@sha256:7062a2109c4b51f3c792c7ea01e83ed12ef9a988886e3b3d380a7d2e5f6ce3f5 0.0s
=> CACHED [postgres builder 2/4] RUN apk add --no-cache build-base git postgresql-dev clang llvm-dev 0.0s
=> CACHED [postgres builder 3/4] WORKDIR /build                             0.0s
=> CACHED [postgres builder 4/4] RUN git clone --branch v0.5.0 https://github.com/pgvector/pgvector.git && cd pgvector && make && mak 0.0s
=> CACHED [postgres stage-1 2/3] COPY --from=builder /usr/local/lib/postgresql/ /usr/local/lib/postgresql/ 0.0s
=> CACHED [postgres stage-1 3/3] COPY --from=builder /usr/local/share/postgresql/ /usr/local/share/postgresql/ 0.0s
=> [postgres] exporting to image                                             0.0s
=> => exporting layers                                                       0.0s
=> => writing image sha256:d507caf0a5207bcd9779daf1a95b123036174ada866d246256aa0088885252768 0.0s
=> => naming to docker.io/library/postgres-postgres                       0.0s
=> [postgres] resolving provenance for metadata file                        0.0s
[+] Running 2/2
✓ postgres Built 0.0s
✓ Container postgres-postgres-1 Created 0.0s
Attaching to postgres-1
postgres-1 |
postgres-1 | PostgreSQL Database directory appears to contain a database; Skipping initialization
```

Test the database is working correctly

You need to install requirements for psql and make sure the .env file is written correctly to be able to source it.

```
$ cd rag-app
$ source .env
$ psql -h localhost -U $POSTGRES_USER -d $POSTGRES_DB -p $POSTGRES_PORT
$ \dt
```

Output in the terminal

```
Password for user myuser:
psql (14.17 (Ubuntu 14.17-0ubuntu0.22.04.1), server 17.4)
WARNING: psql major version 14, server major version 17.
         Some psql features might not work.
Type "help" for help.

mydb=# \dt
          List of relations
 Schema | Name  | Type  | Owner
-----+-----+-----+-----
 public | papers | table | myuser
(1 row)
```

Also, we can test it is running via:

```
$ docker ps
```

Output in the terminal, you will find the postgres container running.

CONTAINER ID	IMAGE	COMMAND	CREATED
10a591dbb745	vsc-oxford_rag_aws-c1b5af4e4eb42d0d417600d7f5e1da46ea59b0943eef9ba7b109c5fc677b7c3a-uid	"/bin/sh -c 'echo Co..."	3 hours
ago	Up 3 hours	clever_engelbart	
138a3a8ec820	postgres-postgres	"docker-entrypoint.s..."	6 days
ago	Up 7 minutes	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp	postgres-postgres-1

Data Ingestion and Embedding

Keyword for search

The keyword for search is defined in (rag-app/server/src/ingestion/arxiv_client.py)

```
arxiv_client.py 1, M X
rag-app > server > src > ingestion > arxiv_client.py > ...
91
92
93 if __name__ == "__main__":
94     # papers = fetch_papers(query="ti:perovskite", max_results=10)
95     papers = fetch_papers_paginated(
96         query="ti:perovskite", max_results=20, results_per_page=5, wait_time=5
97     )
```

I have tried a different query keyword (llm) and downloaded the new data using

```
$ make download-data
```

```
arxiv_client.py 1 X
rag-app > server > src > ingestion > arxiv_client.py > ...
91
92
93 if __name__ == "__main__":
94     # papers = fetch_papers(query="ti:perovskite", max_results=10)
95     papers = fetch_papers_paginated(
96         query="ti:llm", max_results=20, results_per_page=5, wait_time=5
97     )
```

Then ingest the data using

```
$ make run-ingestion
```

```
vscode → /workspaces/oxford_rag_aws/rag-app (main) $ make run-ingestion
poetry run python ./server/src/ingestion/pipeline.py
/home/vscode/.cache/pypoetry/virtualenvs/rag-app-UsbWnj6X-py3.12/lib/python3.12/site-packages/transformers/tokenization_utils_base.py:1617: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be deprecated in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Reading JSON files from ./papers-downloads...
Successfully processed 80 papers.
Successfully inserted 80 rows into the papers table.
Completed ingestion into database mydb
```

Make sure the rows are inserted in the database

```
$ psql -h localhost -U $POSTGRES_USER -d $POSTGRES_DB -p $POSTGRES_PORT  
mydb=# SELECT count(*) FROM public.papers;
```

Output in the terminal:

```
mydb=# SELECT count(*) FROM public.papers;  
count  
-----  
      80  
(1 row)
```

Note, if you ran the ingestion twice, then you have inserted the 80 documents twice in the database. To clear the database and reset the auto-increment id, use the following command

```
$ psql -h localhost -U $POSTGRES_USER -d $POSTGRES_DB -p $POSTGRES_PORT  
mydb=# TRUNCATE TABLE public.papers RESTART IDENTITY;
```

This will reset the database to have 0 rows, and to fill it again, ingest the documents once again.

Output in the terminal:

```
mydb=# TRUNCATE TABLE public.papers RESTART IDENTITY;  
TRUNCATE TABLE  
mydb=# SELECT count(*) FROM public.papers;  
count  
-----  
      0  
(1 row)
```

RAG Workflows

top-k is forced in “LIMIT %s”, and the used limit is passed in “execute()” method

```
try:
    cursor = conn.cursor()

    # SQL query to find the top_k chunks using cosine similarity
    query = """
    SELECT id, title, chunk, embedding <=> %s::vector AS similarity
    FROM papers
    ORDER BY similarity ASC
    LIMIT %s;
    """

    # Execute the query with the query embedding and top_k value
    cursor.execute(query, (query_embedding, top_k))
    rows = cursor.fetchall()
```

Exercise questions

a. What does top_k mean in this context (i.e., of RAG)?

top_k specifies the number of most relevant document chunks to retrieve from the database based on their similarity to the query embedding. It controls how many context passages are fed to the language model.

b. What will happen if I increase or decrease top_k?

- **Increase top_k** → More documents retrieved, potentially more relevant context but slower performance and higher token cost. Also, too large k may add irrelevant context that may confuse the LLM.
- **Decrease top_k** → Fewer documents retrieved; faster and cheaper but might miss important context.

c. What is the algorithm inside the database query and how does it work?

The algorithm is **cosine similarity** (implemented via the `<=>` operator in pgvector for vector types). It computes the angular distance between the query embedding and document embeddings:

- Cosine similarity = dot product of normalized vectors.
- Lower distance implies higher similarity.

d. Why is the query to Postgres asking for results in ascending order?

Because the `<=>` operator returns **cosine distance**, not similarity — smaller values indicate closer (i.e., more similar) vectors. Sorting by ASC ensures the **most similar** chunks come first.

There is an error control flow in the `retrieve_top_k_chunks_endpoint` that deals with a failure to retrieve chunks (i.e context). Under what scenarios could this happen?

Not sure about that, but some possibilities are:

- The papers table has no data
- Embeddings are missing or malformed
- The query embedding can't be compared (dimension mismatch)

The process flow that happens when a user hits the “generate” endpoint with a query:

Steps:

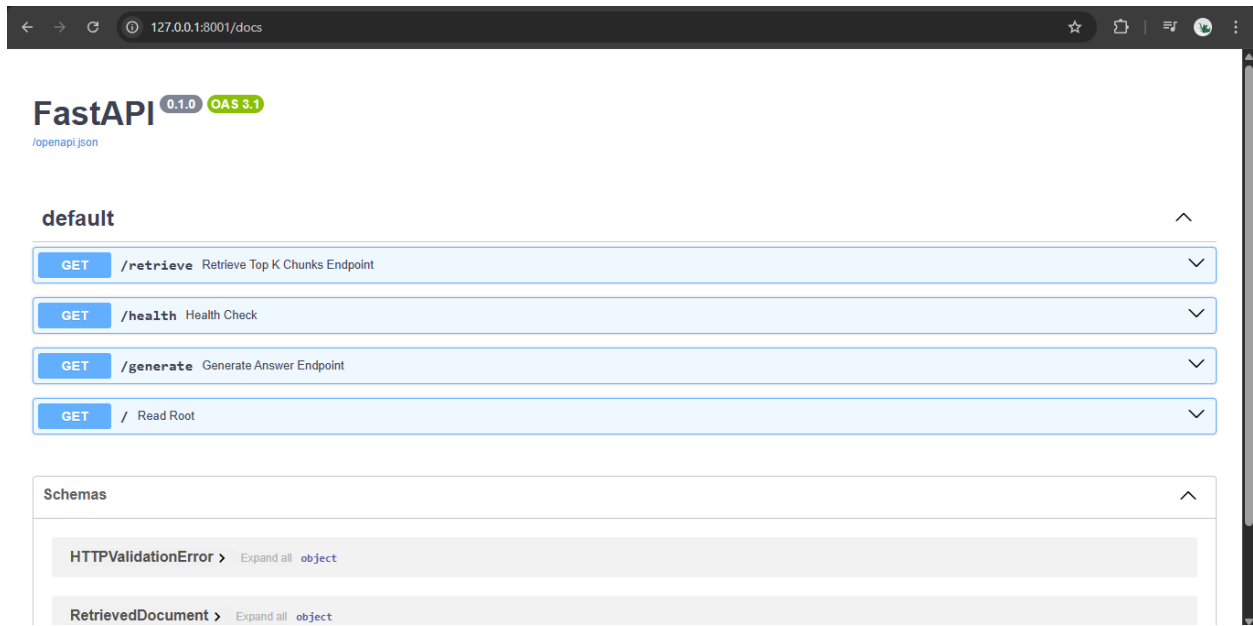
- `rag-app/server/src/controllers/generation.py => generate_answer_endpoint(...)`
 - `chunks = retrieve_top_k_chunks(query)`
 - `generated_response = generate_response(query, chunks)`
- `return generated_response`

Run the app

```
$ make run-app
```

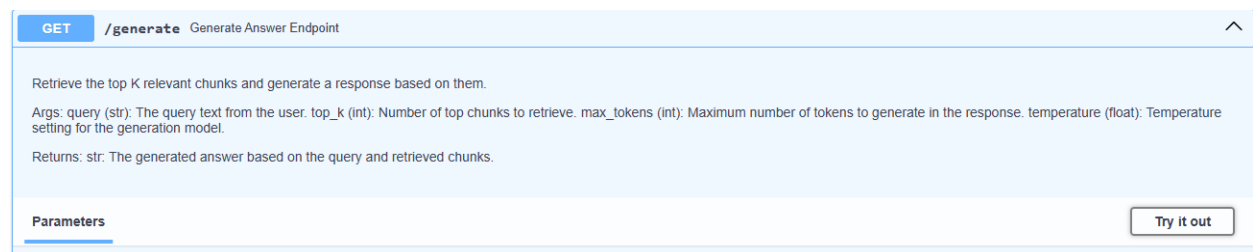
Then navigate to <http://127.0.0.1:8000/docs>

You will find swagger docs



Generate

Try the generation endpoint



Insert your query and other parameters

ParametersCancel

Name	Description
query * required string (query)	The query text from the user <input type="text" value="Instead of compressing existing models, wha"/>
top_k integer (query)	Number of top chunks to retrieve <input type="text" value="5"/>
max_tokens integer (query)	The maximum number of tokens to generate <input type="text" value="200"/>
temperature number (query)	Sampling temperature for the model <input type="text" value="0.7"/>

Execute**Clear**

Execute to get the response

Responses

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8001/generate?query=Instead%20of%20compressing%20existing%20models%20what%20is%20the%20alternative%20approach%20mentioned%20in%20the%20summary%20for%20obtaining%20smaller%20language%20models%3F&top_k=5&max_tokens=200&temperature=0.7' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8001/generate?
query=Instead%20of%20compressing%20existing%20models%20what%20is%20the%20alternative%20approach%20mentioned%20in%20the%20summary%20for%20obtaining%20smaller%20language%20models%3F&top
_k=5&max_tokens=200&temperature=0.7
```

Server response

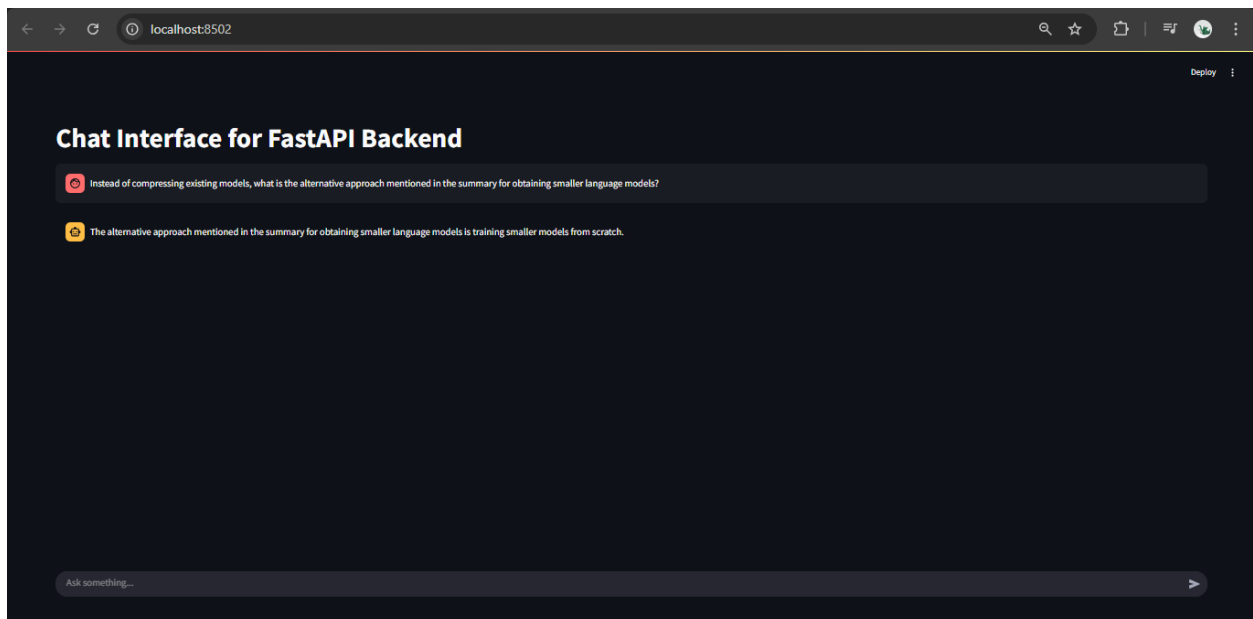
Code	Details
200	<div>Response body<pre>{ "response": "The alternative approach mentioned in the summary for obtaining smaller language models is training smaller models from scratch. This is mentioned as an alternative to data-driven compression of existing pretrained models.", "response_tokens_per_second": 31.485714285714284, "query_expanded": false }</pre>Download</div> <div>Response headers<pre>content-length: 308 content-type: application/json date: Sat, 03 May 2025 20:15:33 GMT server: uvicorn</pre></div>

Frontend

You can do the same but with a front-end client

```
○ vscode →/workspaces/oxford_rag_aws/rag-app (main) $ make run-client  
poetry run streamlit run client/streamlit_app.py  
  
You can now view your Streamlit app in your browser.  
  
Local URL: http://localhost:8501  
Network URL: http://172.30.185.175:8501
```

navigate to <http://127.0.0.1:8502/>, you will find streamlit app running



Tracing and Tracking

It is possible to run some logic upon the initialization of your FastAPI server that will persist specific objects or configurations throughout the lifetime of the session. Can you see where this is done in the `server/src/main.py` file?

What has been initialized, why is this a good place to run these commands?

Yes, this is done inside the `lifespan_context` async function:

- **Where:** In `server/src/main.py`, it's done in the `lifespan_context` function.
- **What is initialized:**
 - `opik.configure()` — a global config setup
 - `embedding_model = SentenceTransformer("all-MiniLM-L6-v2")` — a sentence embedding model loaded once into memory.
- **Why is it a good place:**
 - The lifespan context is the **ideal place for startup logic** because it's run **only once** when the server starts (not on every request), making it efficient for heavy initialization like loading models.
 - It also allows proper **cleanup** at shutdown via the “finally” block.
 - Keeping models in memory here allows **reusing expensive resources** without reloading them per request, improving performance and scalability.

Add some tracking and tracing to other services and functions in the application. Check that these are successfully logged into your Opik instance in the online portal.

We Tracked the `get_db_connection(...)` function

```
@opik.track
def get_db_connection(db_config: dict):
    """
    Establishes a connection to the Postgres database.

    Args:
        db_config (dict): Dictionary containing Postgres connection details

    Returns:
        psycopg2.connection: The connection object.
    """
    return psycopg2.connect(**db_config)
```

It is successfully tracked in opik

The screenshot displays the Opik online portal interface. On the left, a 'Trace spans' sidebar shows a tree of spans: 'retrieve_top_k_...' (0.3s), 'retrieve_top_...' (0.3s), and 'get_db_co...' (0s). The 'get_db_co...' span is selected. The main panel shows the 'Input/Output' tab for this span. The input is a YAML dictionary: `db_config: {dbname: mydb, user: myuser, password: mypassword, host: localhost, port: '5432'}`. The output is a string representation of a PostgreSQL connection object: `<connection object at 0x7f335f6756c0; dsn: 'user=myuser password=xxx dbname=mydb host=localhost port=5432', closed: 1>`. The interface includes a search bar, a 'Collapse all' button, and a 'Delete' button in the top right.

Also, other functions are being tracked

The screenshot shows a trace of LLM calls. On the left, a 'Trace spans' panel displays a tree of spans: 'generate_resp...' (0.8s), 'generate_re...' (0.8s), and 'call_llm' (0.8s). The main panel shows the details for the 'generate_response' span, which took 0.8s. It includes tabs for 'Input/Output', 'Feedback scores', and 'Metadata'. The 'Input/Output' tab shows the input prompt: 'Instead of compressing existing models, what is the alternative approach mentioned in the summary for obtaining smaller language models?' and the output: 'The alternative approach mentioned in the summary for obtaining smaller language models is training smaller models from scratch.'

Online Evaluation

Created a custom evaluation metric

The screenshot shows the 'oxford-aws' online evaluation interface. It has tabs for 'Traces', 'LLM calls', 'Threads', 'Metrics', and 'Online evaluation'. The 'Online evaluation' tab is active, showing a table with columns: Name, Last updated, Created, Created by, and # Sampling rate. The table contains one row for 'Custom Score' with a value of 1. A 'Show logs' link is next to the value. The interface also includes a search bar, a 'Columns' button, and a 'Create new rule' button.

And check it is running

The screenshot shows the Comet ML interface. On the left, a sidebar lists various sections: Home, Observability, Projects, Evaluation, Datasets, Experiment, Prompt engine, Prompt libr, Playground, Production, Online eval, Configuration, and Configurati. The 'Trace spans' panel shows a tree of spans: 'generate...' (1.1s), 'genera...' (1.1s), and 'call_...' (1.1s). The main panel shows the details for the 'generate_response' span, which took 1.1s and has 1 feedback score. It includes tabs for 'Input/Output', 'Feedback scores', and 'Metadata'. The 'Feedback scores' tab shows a table with columns: Key, Score, and Reason. The table contains one row for 'my_custom_metric' with a score of 0.650104928 and a reason of 'Reason for the score'.