Assignment 6. MLOps with AWS

Project Setup

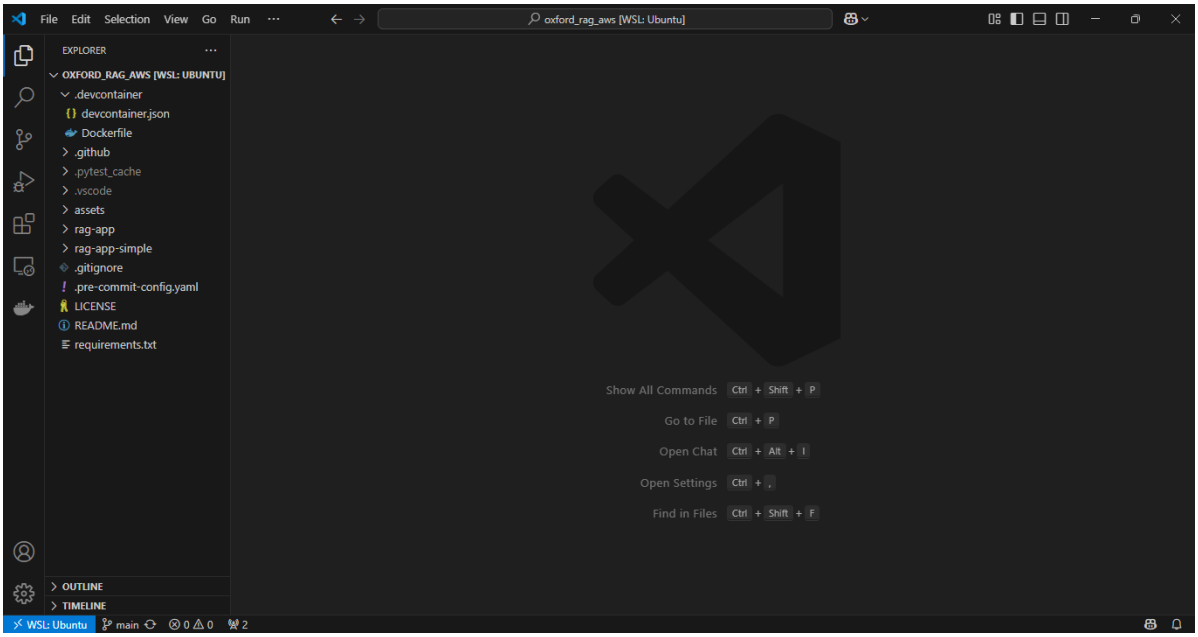
Clone project repo:

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

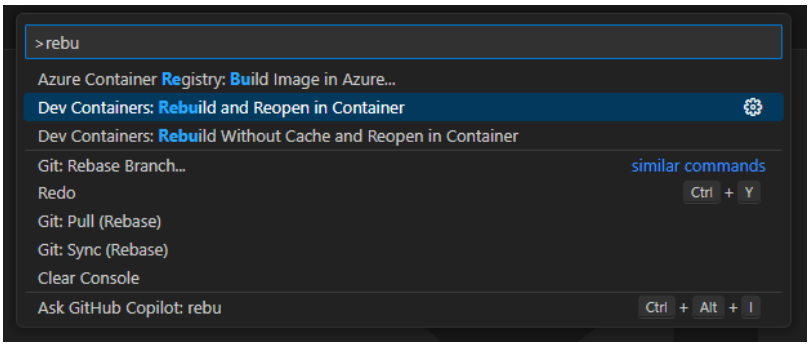
Then cloned it locally

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

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

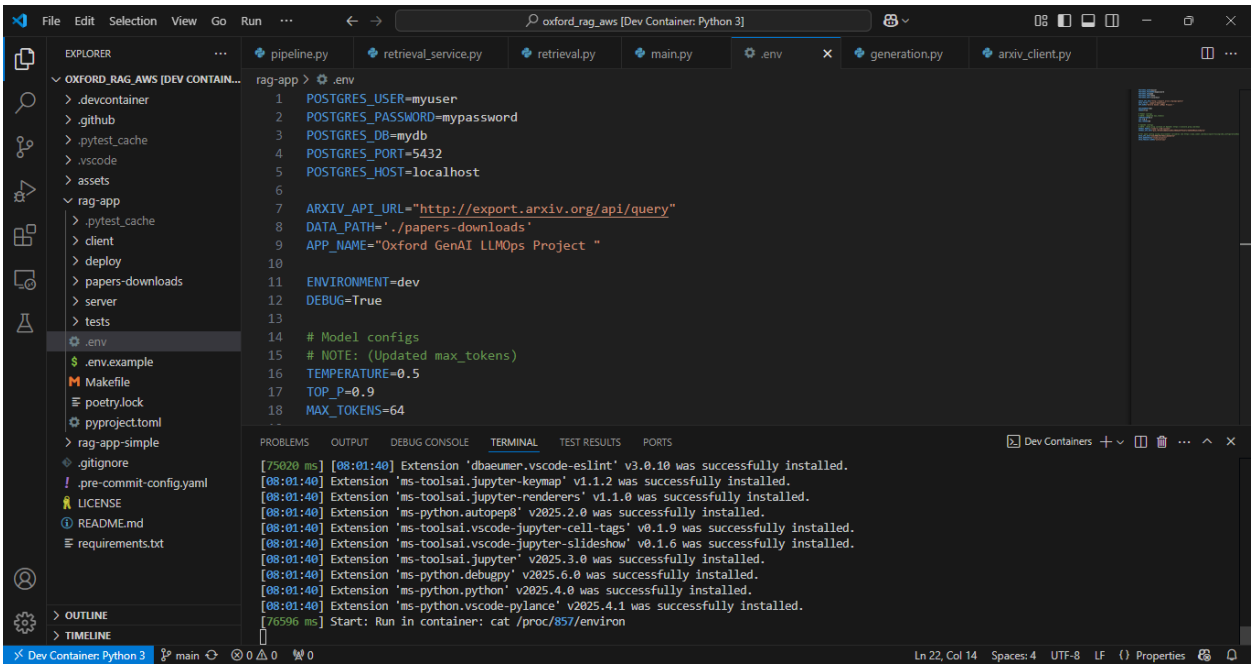


Build the Dev Container



Setup .env file

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



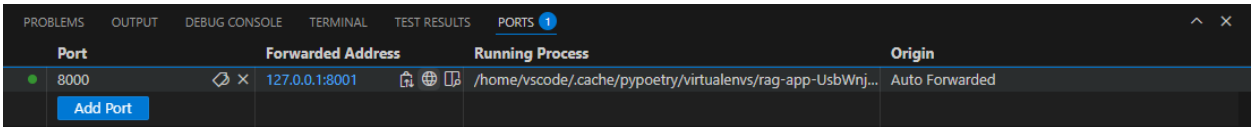
Install dependencies

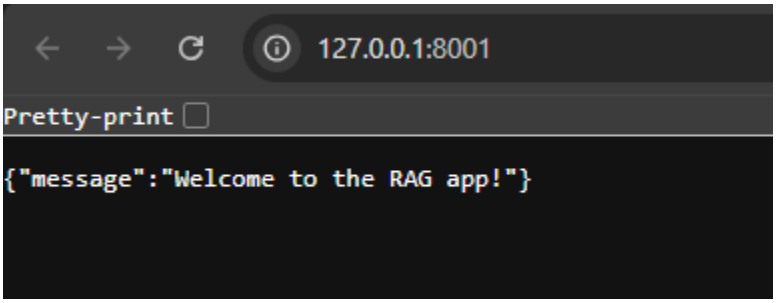
```
vscode ➜/workspaces/oxford_rag_aws (main) $ cd rag-app
vscode ➜/workspaces/oxford_rag_aws/rag-app (main) $ make install
poetry install
Creating virtualenv rag-app-UsbWnj6X-py3.12 in /home/vscode/.cache/pypoetry/virtualenvs
Installing dependencies from lock file

Package operations: 270 installs, 0 updates, 0 removals

  - Installing attrs (24.2.0)
  - Installing rpds-py (0.20.1)
  - Installing referencing (0.35.1)
  - Installing six (1.16.0)
  - Installing jsonschema-specifications (2024.10.1)
  - Installing platformdirs (4.3.6)
  - Installing python-dateutil (2.9.0.post0)
  - Installing traitlets (5.14.3)
  - Installing types-python-dateutil (2.9.0.20241003)
  - Installing arrow (1.3.0)
  - Installing fastjsonschema (2.20.0)
  - Installing jsonschema (4.23.0)
  - Installing jupyter-core (5.7.2)
  - Installing pycparser (2.22)
  - Installing pyzmq (26.2.0)
```

Run the application

```
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





GitHub Actions

We have defined the following GitHub workflow to:

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

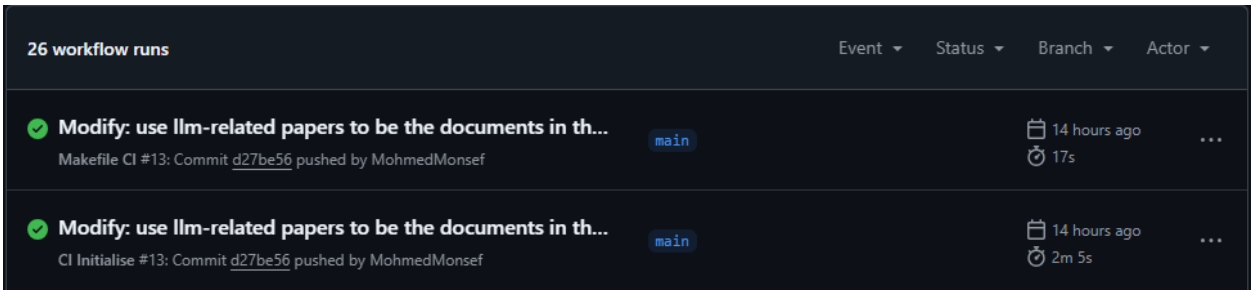
```yaml
name: CI Initialise

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
```

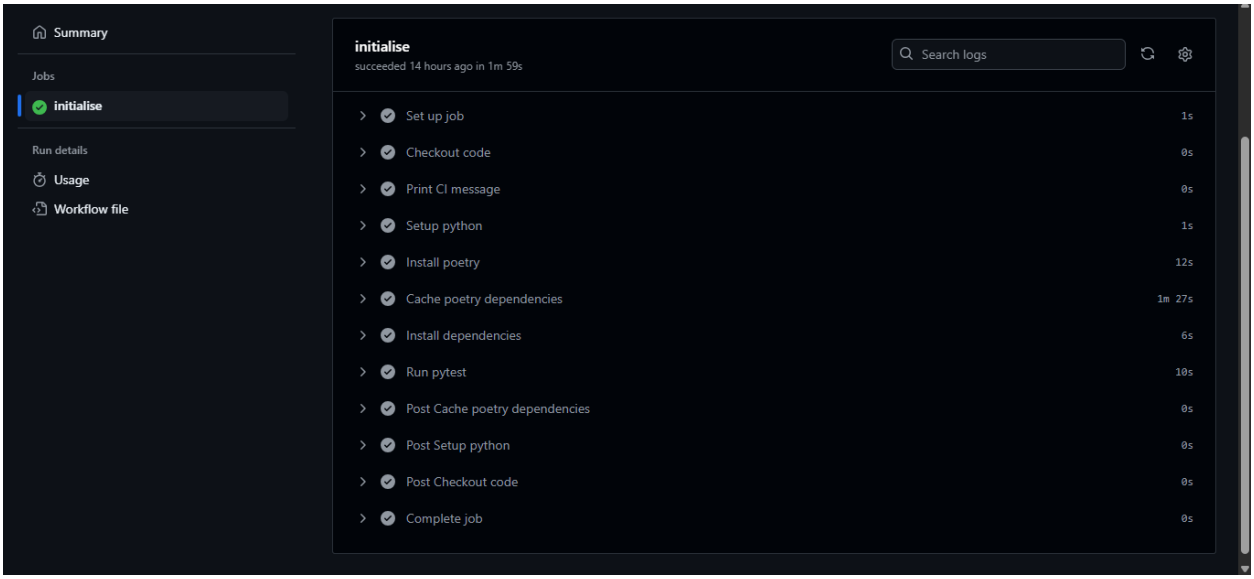
```
                key: poetry-${{ runner.os }}-${{
hashFiles('**/poetry.lock') }}
                restore-keys: |
                    poetry-${{ runner.os }}-

            - 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.





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 ignor
ed, 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@sha256:7062a2109c4b51f3c792c7ea01e83ed12ef9a980886e3b3d380a7d2e5f6ce3f5    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:d507caf0a5207bcd9779daf1a95b123036174ada866d246256aa008885252768                                                 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
```

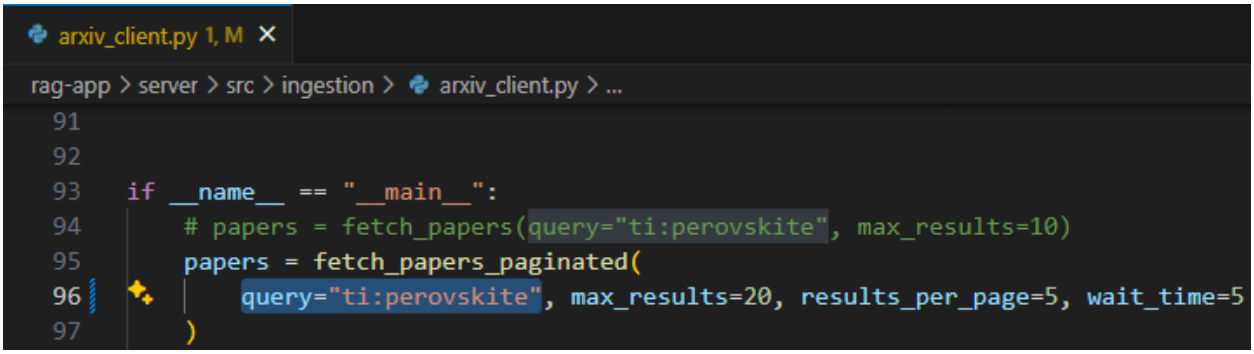
Output in the terminal, you will find the postgres container running.

```
CONTAINER ID   IMAGE                                                                                                COMMAND                  CREATED
       STATUS          PORTS                                       NAMES
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

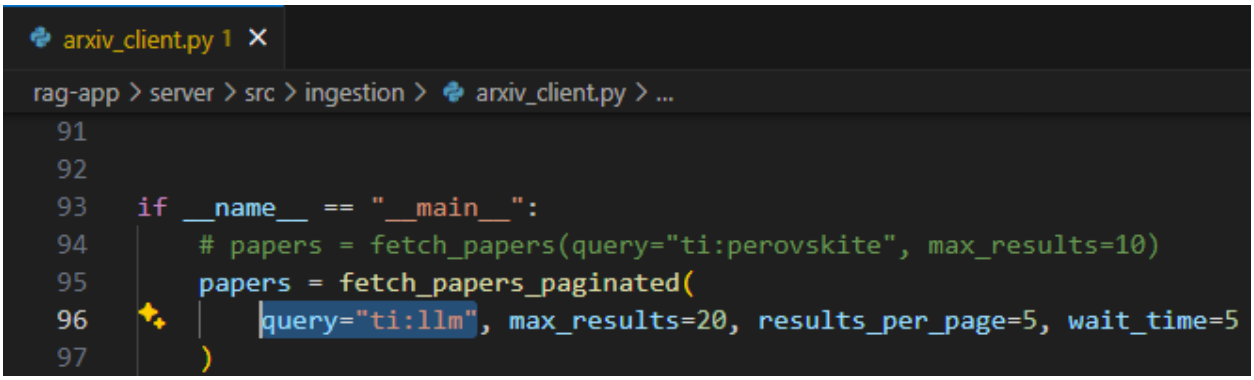
Keyword for search

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



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

```
$ make download-data
```



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:161
7: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be deprecated in tr
ansformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transform
ers/issues/31884
  warnings.warn(
Reading JSON files from ./papers-downloads...
Succesfully 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:

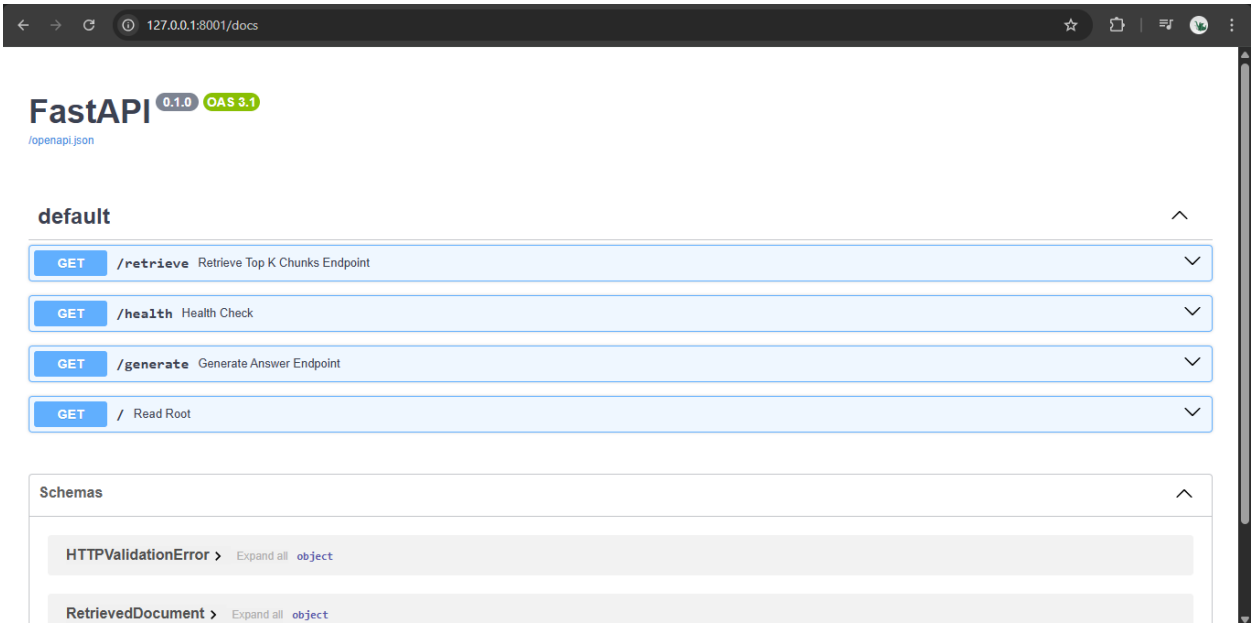
- rag-app/server/src/controllers/generation.py => generate_answer_endpoint(...)
    - chunks = retrieve_top_k_chunks(query)
    - generated_response = generate_response(query, chunks)
- return generated_response

Run the app

```
$ make run-app
```

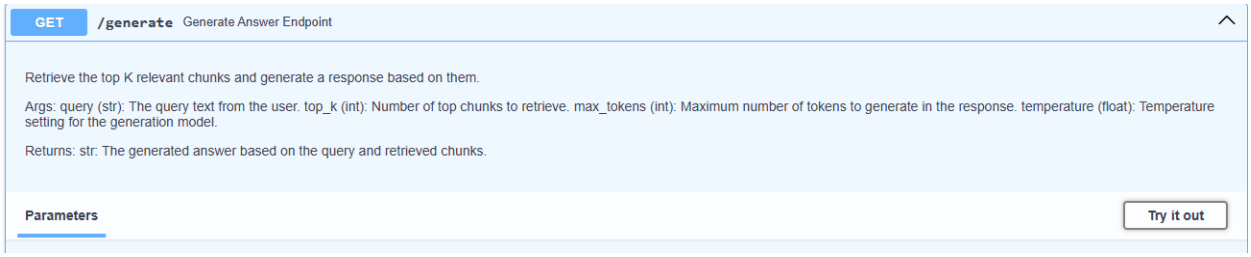
Then navigate to http://127.0.0.1:8000/docs

You will find swagger docs

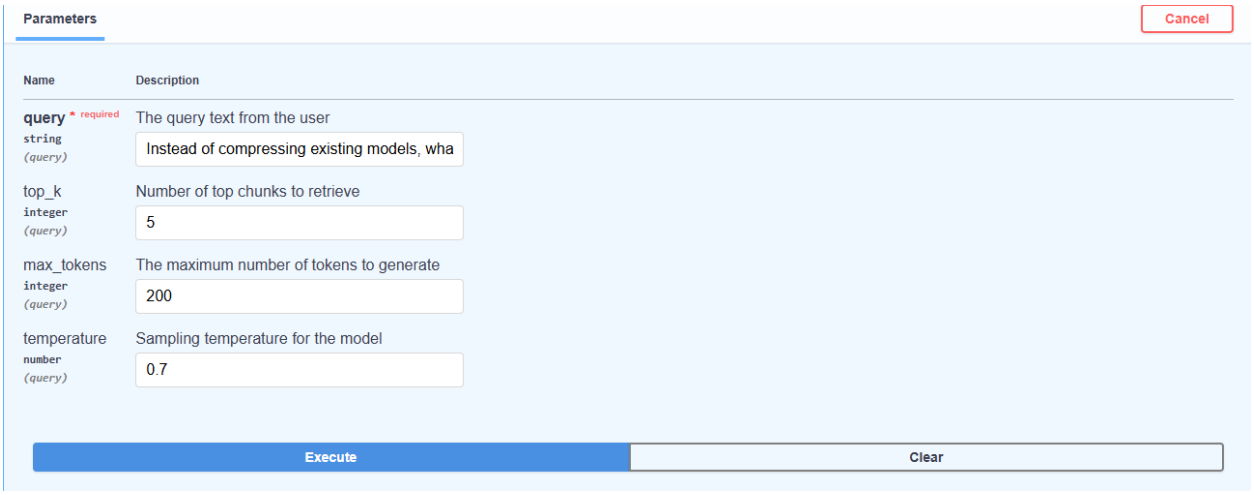


Generate

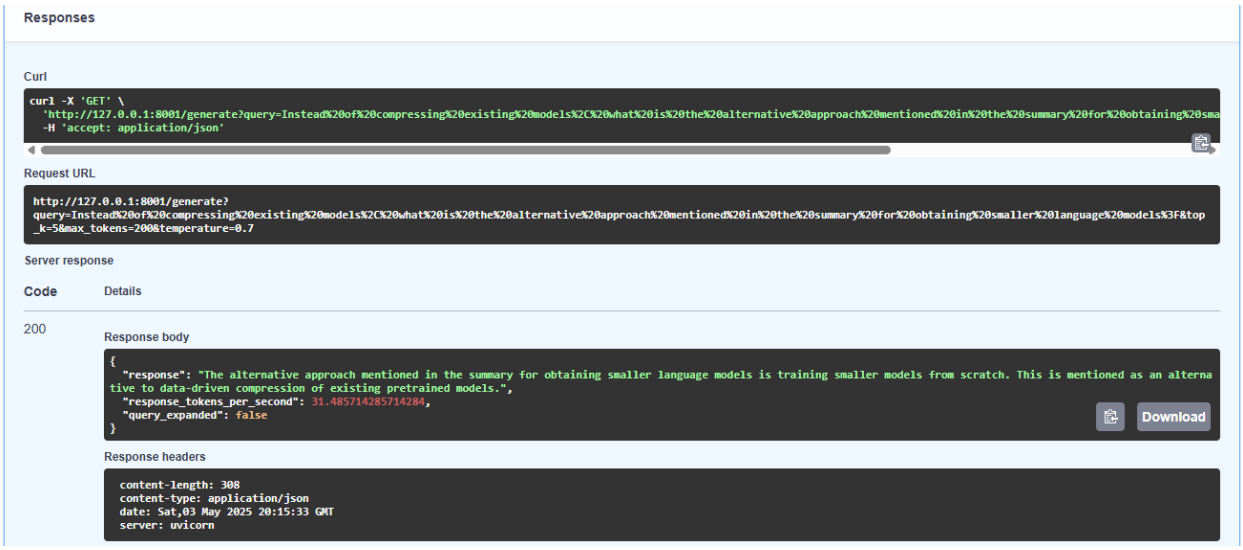
Try the generation endpoint



Insert your query and other parameters



Execute to get the response

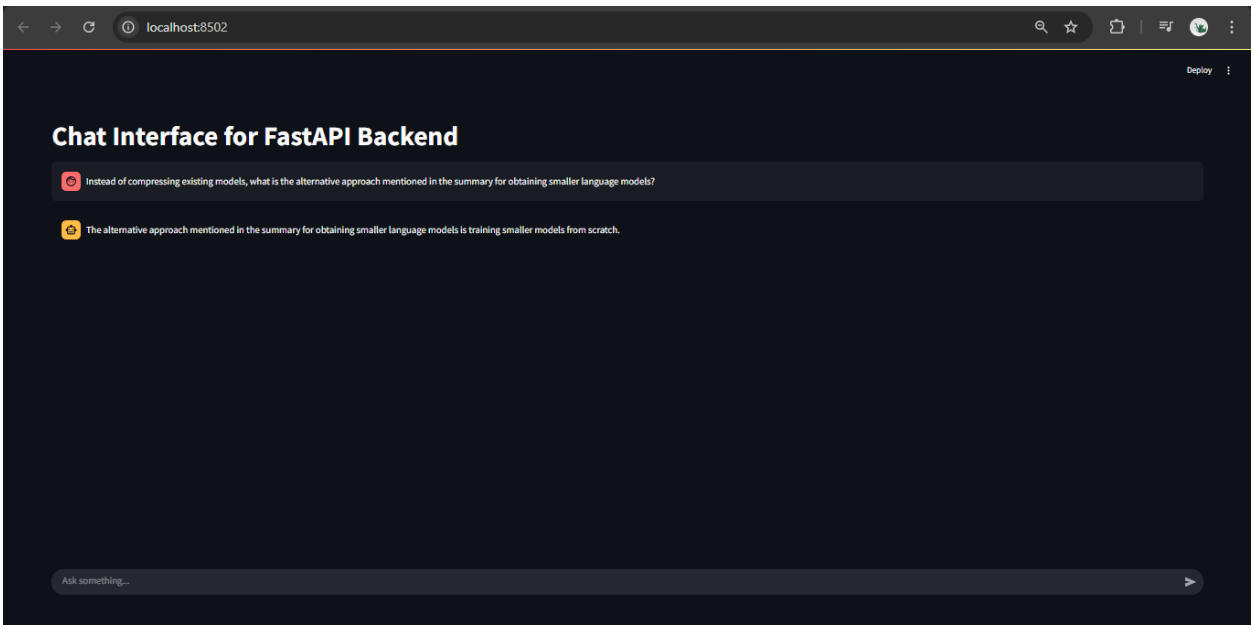


Frontend

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:

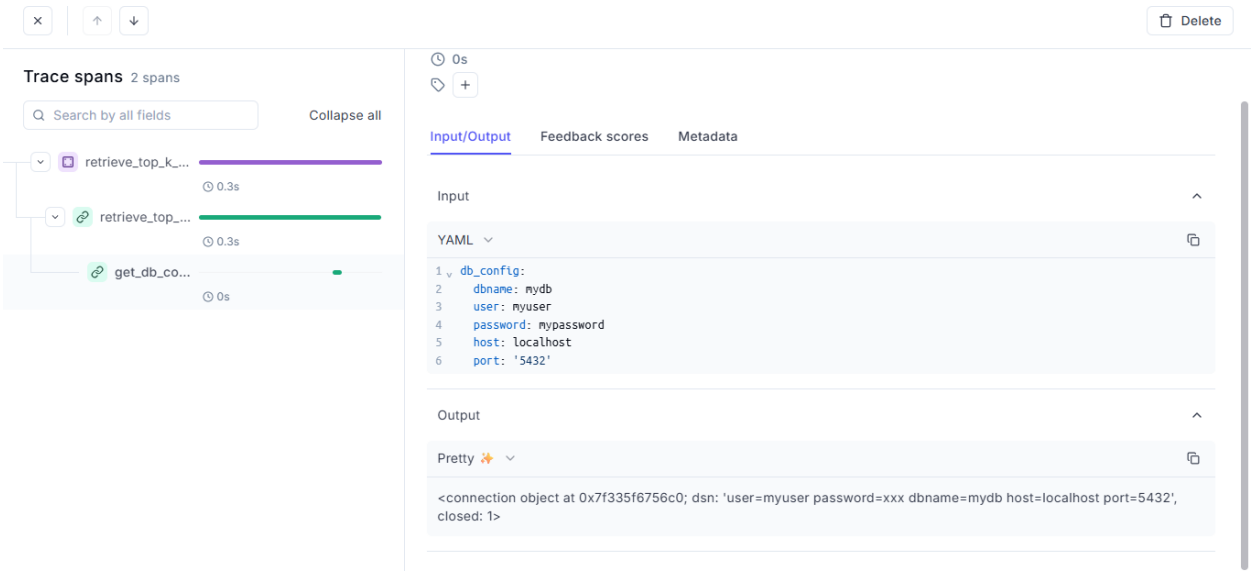
- **Where**: In server/src/main.py, it's done in the lifespan_context function.
- **What is initialized**:
    - opik.configure() — a global config setup
    - embedding_model = SentenceTransformer("all-MiniLM-L6-v2") — a sentence embedding model loaded once into memory.
- **Why is it a good place**:
    - 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.

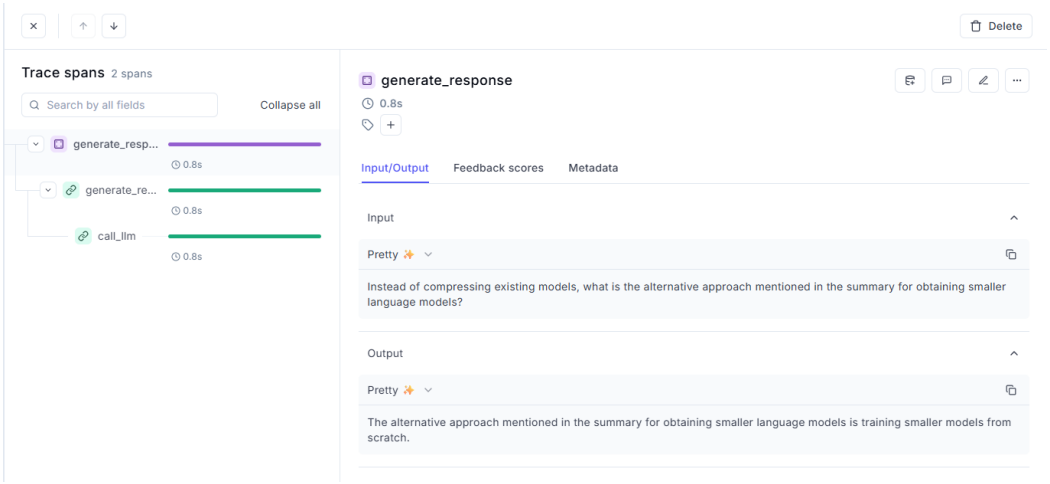
We Tracked the get_db_connection(...) function

```python
@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

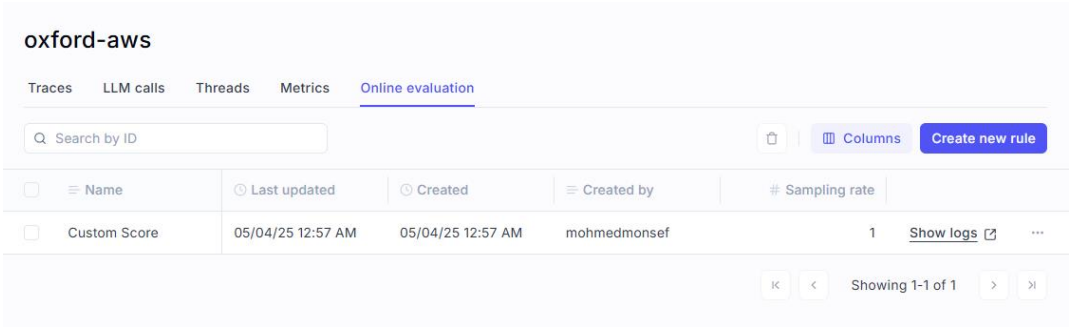


Also, other functions are being tracked



Online Evaluation

Created a custom evaluation metric



And check it is running