National College of Ireland

Configuration Manual

MSc Research Practicum
MSc Data Analytics

Priya Sethi Khurana
Student ID: x23292547

School of Computing
National College of Ireland

Supervisor: Aaloka Anant

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Priya Sethi Khurana
Student ID:	X23292547
Programme:	MSc Data Analytics **Year:** 2024-2025
Module:	Research Practicum 2 (Configuration Manual)
Lecturer:	Aaloka Anant
Submission Due Date:	August 11, 2025
Project Title:	"A Lightweight Semantic Patent Search system for the Electric Vehicle Domain"

Word Count: 1393 **Page Count:** 9

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Priya

Date: 11/08/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Priya Sethi Khurana
X23292547

Introduction

This configuration manual provides the necessary information to reproduce the EV Patent Semantic Search system. The project was implemented primarily in **Google Collab**, a cloud-based execution environment, which eliminates the need for complex local setup. All required code is provided in the form of an. ipynb notebook, which can be opened directly in Collab. For users wishing to run the system locally, equivalent hardware, software, and library requirements are also detailed

- Execution environment and prerequisites
- Dataset acquisition and preparation
- Model setup and embedding generation
- Indexing and retrieval configuration
- Evaluation and result reproduction

1. Execution Environment

1.1 **Primary Platform**: Google Collab (Python 3.11.13 runtime)

Runtime type: GPU (recommended) or CPU

Pre-installed libraries in Collab: pandas, NumPy, matplotlib, torch

1.2 Local Environment

Device name	DESKTOP-3A7QM0I
Processor	13th Gen Intel(R) Core(TM) i5-1335U (1.30 GHz)
Installed RAM	16.0 GB (15.7 GB usable)
Device ID	458A1197-3B12-49DC-BD93-C1FB1248A93C
Product ID	00342-42676-59428-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Local Environment

2. Connecting to a Collab Session and Drive setup

2.1 Collab session setup

- Go to https://colab.research.google.com/
- If prompted, sign in with your Google account.
- Select the supplied. ipynb file (x23292547_Research Practicum 2_Patent_search_MSc_Data Analytics. ipynb) from your system. And the steps are

Go to Fi⟹ Upload Notebook(.ipynb file mentio⟹ve) Click Browse(or just drag)

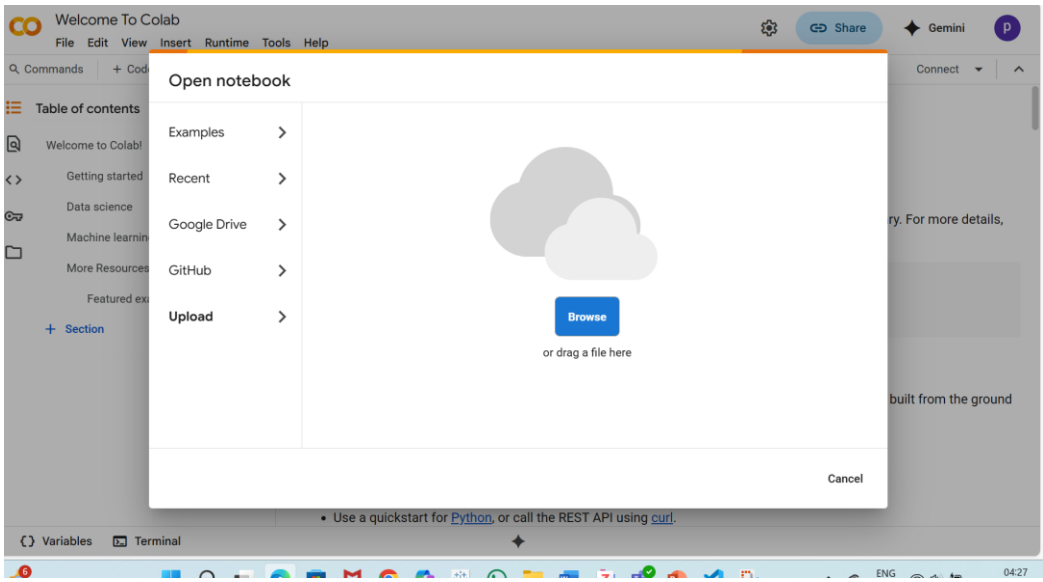

Figure 2: Open .ipynb(code file)

- To speed up code execution, use a GPU runtime (CPU can also be used if GPU is not available, but processing will be slower):

1. Go to Runtime → Change runtime type
2. Under Hardware accelerator**, select** GPU (T4 GPU recommended)
3. Click Save

(See below Figure for the runtime settings interface.)

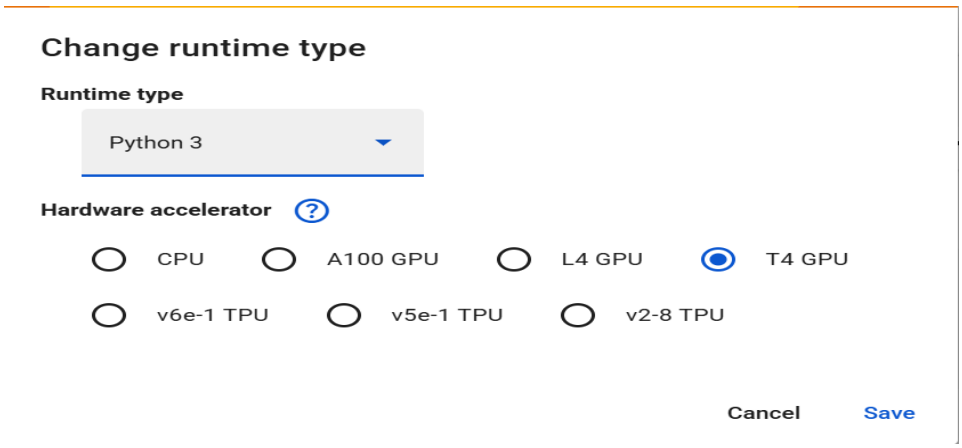

Figure 3 : Runtime Setting Interface

2.2 Drive Setup

Google Drive is used in this project to store the dataset and generated files (e.g., embeddings, FAISS index) in a persistent location. When you open a new Colab notebook, you must

connect it to your Google account so the notebook can access your Drive. This connection is done by running the mount command:

- Run the Mount Command by running this in code cell

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

When you run this, a **permission pop-up** will appear in the notebook asking:

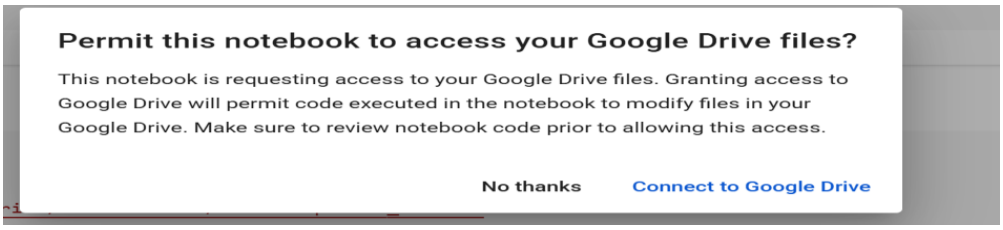

Figure 4: Drive Setup

Click Connect to Google Drive" to proceed. Make sure browser pop-ups are not blocked.

You may be prompted to select your Google account and confirm access.

2.3 Uploading Files to Google Drive

Before you start working with Google Collab, you need to upload the required files to the same Google Drive account that will be connected to Collab in Section 2.2 (Drive Setup). Using the same account avoids file path and permission issues during execution.

- Go to https://drive.google.com/ and sign in with your Google account.
- In your Google Drive, create a new folder called **PatentSearch**
- Upload "Modifiedpatent_12k.csv" to the PatentSearch folder.

3. Essential Libraries

To ensure the code runs without errors and all features work as intended, install the following Python libraries in your Google Collab environment before proceeding with data preprocessing or model building. Some of these libraries are pre-installed in Collab, but running the installation cell ensures that compatible versions are available

```
# Check Python version (Expected: Python 3.11.13)
!python --version
# Install required libraries
!pip install numpy pandas matplotlib seaborn plotly wordcloud faiss-cpu tqdm
!pip install -q sentence-transformers
```

Note: If you run the notebook in a fresh Collab session, install these libraries before importing them in your code to avoid ModuleNotFoundError issues.

4. Dataset preparation

4.1 Loading the Dataset in Collab

Once the dataset Modifiedpatent_12k.csv has been uploaded to the PatentSearch folder in your Google Drive (as described in Section 2.3), it can be loaded into the Collab environment using the following code:

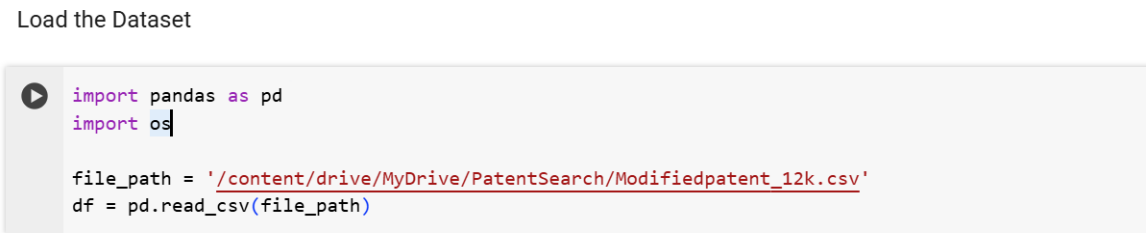

Figure 5 : Loading Dataset

Run this cell in code block manually and make sure file_path should exactly match the location in your Google Drive.

5. Preprocessing and Claim Weighting

This step processes the dataset to prepare it for semantic embedding:

- Claim Splitting: Separates each patent into its constituent claims.
- Claim Weighting: Assigns higher importance to independent claims and proportionally lower weights to dependent claims. This ensures that search prioritises the most informative sections.

Instructions:

1. Ensure the dataset is loaded (Section 4).
2. Run all the cells after step 4.1 which are preprocessing (including cleaning) and EDA and weighting cells sequentially.
3. No modifications are required unless adjusting weights for experimentation.

Purpose:
Weighting claims before embedding improves retrieval accuracy by focusing the model on the most relevant text segments.

Applying weights to claim_list

```python
import pandas as pd
import re

# Assuming you already have `df` with 'claim_list' column
# from the splitting step (no CSV loading here)

#  Define weighting function
def weight_claim(claim, index):
    length_score = min(len(claim) / 500, 1.0)
    position_score = 1.0 - (index * 0.05)
    position_score = max(position_score, 0.3)  # don't go below 0.3
    final_weight = round(0.6 * length_score + 0.4 * position_score, 3)
    return final_weight

#  Apply weighting
df['claim_weights'] = df['claim_list'].apply(
    lambda claims: [weight_claim(c, i) for i, c in enumerate(claims)]
)

#  Preview result
print(" Claim weights calculated and stored in memory.")
df[['publication_number', 'claim_list', 'claim_weights']].head()
```

```
Claim weights calculated and stored in memory.
```

	publication_number	claim_list	claim_weights
0	US-7722498-B2	[A control device for a hybrid electric vehicl...	[1.0, 0.98, 0.96, 0.88, 0.762, 0.605, 0.88]
1	US-7657351-B2	[A method for efficiently controlling an engin...	[1.0, 0.632, 0.492, 0.479, 0.451, 0.467, 0.438...
2	US-7679884-B2	[An electrolyte comprising a cationic moiety c...	[1.0, 0.465, 0.496, 0.494, 0.494, 0.451, 0.43,...
3	US-9656599-B2	[The embodiments of the invention in which an ...	[0.533, 0.98, 0.538, 0.644, 0.508, 0.565, 0.74...
4	US-9487099-B2	[An apparatus for connecting an electric vehic...	[1.0, 0.615, 0.522, 0.57, 0.652, 0.548, 0.474,...

Figure 6: Preprocessing and Claim weighting

6. Embedding Generation (Model)

This step converts each processed and weighted claim into a semantic vector using the MiniLM-based Sentence Transformer model.

Instructions:

1. Complete preprocessing and claim weighting (Section 5).
2. Ensure sentence-transformers is installed (Section 3).
3. Run the embedding generation cells to create the embeddings.

```python
from sentence_transformers import SentenceTransformer
import numpy as np
from tqdm import tqdm

from sentence_transformers import SentenceTransformer
model = SentenceTransformer("sentence-transformers/msmarco-MiniLM-L-6-v3")

#  Define function to embed and weight claims
def embed_patent(claims, weights):
    embeddings = model.encode(claims)  # shape: (n_claims, 512)
    weights = np.array(weights).reshape(-1, 1)  # shape: (n_claims, 1)
    weighted = embeddings * weights
    return np.mean(weighted, axis=0)  # shape: (512,)

#  Apply SBERT to all patents
tqdm.pandas()
df['sbert_embedding'] = df.progress_apply(
    lambda row: embed_patent(row['claim_list'], row['claim_weights']),
    axis=1
)

print(" SBERT embeddings added to DataFrame.")
```

Figure 7: Embedding Generation

7. Indexing & Retrieval

This step builds a **FAISS (AI based retrieval model by Meta)** index from the generated claim embeddings, enabling fast nearest-neighbour search in the semantic vector space.

Instructions:

1. Ensure the embeddings. parquet file is available — either generated in Section 6 or provided in the supplied project folder.
2. Load the. parquet file into memory.
3. Run the FAISS indexing cells to create the vector index.
4. Execute the retrieval cells to test queries and return ranked results.
5. Ensure to run complete code cell given in code file named as Patents_search_code. ipynb

```python
# Step 3: Stack and normalize for FAISS
embedding_matrix = np.stack(valid_embeddings)
faiss.normalize_L2(embedding_matrix)

# Step 4: Build FAISS index (Inner Product)
dimension = 384
index = faiss.IndexFlatIP(dimension)
index.add(embedding_matrix)

# Step 5: Filter original dataframe to valid rows only
df_valid = df.iloc[valid_indices].reset_index(drop=True)

# Step 6: Check structure of embeddings
sample = df_valid['sbert_embedding'].iloc[0]
print("🔍 Sample embedding check:")
print("Type:", type(sample))
print("Length:", len(sample))
print("First 5 values:", sample[:5])

# Step 7: Prepare a query for test
query_idx = 0
query_text = df_valid.loc[query_idx, 'claim_list']
```

Figure 8: FAISS Indexing & Retrieval

8. Evaluation & UI

The supplied Stream lit interface allows users to test the semantic search system interactively. It supports:

- Single Query Search: Enter a patent claim/key feature and retrieve top similar results.
- Batch Query Evaluation: Upload a CSV of multiple queries for similarity scoring and visualisation.

Query here includes information that's needed to be search meaning claim element or set of text which might be key feature of the invention & Batch query includes list of different patents to get know about aggregate similarity score for patent record mentioned in that query file

Make a folder "User Interface" or anywhere on your system and keep all of the below files

Required Files:

1. app.py – The main Stream lit application.
2. patent_embeddings. parquet – The claim embeddings file.
3. requirements.txt – List of dependencies for installation.

Note: All files required to run the UI, including the patent embeddings. parquet vector database, are already supplied with this submission (and present in folder name User Interface). ***To be clear, regenerating them using code cell named as "For Exporting file for User Interface (UI)" in attached. ipynb file is optional and only necessary if you modify the dataset or claim weighting***

This project's Stream lit-based UI can be run locally using **Visual Studio Code**. Follow these steps:

8.1 Open the For Visualization Folder in VS Code

1. Launch Visual Studio Code.
2. Go to File → Open Folder.
3. Select the folder ("User Interface") containing:
    - app.py (UI code)
    - patent_embeddings. parquet (vector database)
    - requirements.txt (dependencies)
    - Click on app.py to open file in vs code

8.2 Open the integrated terminal

Windows: Press `Ctrl + `` (backtick) or go to **View → Terminal**.

Mac: Press `Cmd + `` (backtick) or go to **View → Terminal**.

8.3 Install Dependencies (First Time Only)

Windows: *pip install -r requirements.txt*

Mac: *pip3 install -r requirements.txt*

8.4 Launch the Stream lit App

Windows**:** *streamlit run app.py*

Mac**:** *streamlit run app.py*

8.5 Open in Browser

By default, Streamlit opens your **default browser** automatically.

- On **Windows**, this will be **Edge** unless you've changed it.
- On **macOS**, this will be **Safari** unless you've set Chrome or another browser as default.

If the browser does not open automatically:

1. Copy the **local URL** from the terminal (e.g., http://localhost:8501).
2. Paste it into your preferred browser.

8.6 Troubleshooting

a. Port Already in Use**:** Change the port number when launching by typing below command in same terminal of VS code

streamlit run app.py --server. Port 8505

b. Browser Not Opening Automatically

- Copy the Local URL shown in the terminal (e.g., http://localhost:8501 note: the port number may vary).
- Paste it into your preferred browser (Edge, Chrome, Safari, etc.).

c. Page Not Loading

- Ensure no firewall or VPN is blocking the connection.
- Verify Stream lit is still running in the terminal.

9. Final UI

1. Single Search Screen to Enter query (Claims or text from invention to search similar patents)

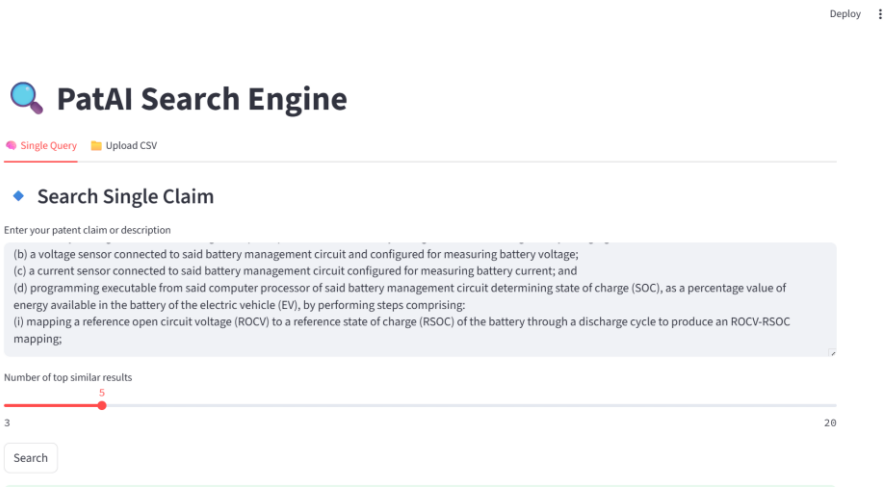

Figure 9: Final UI

2. UI that shows results

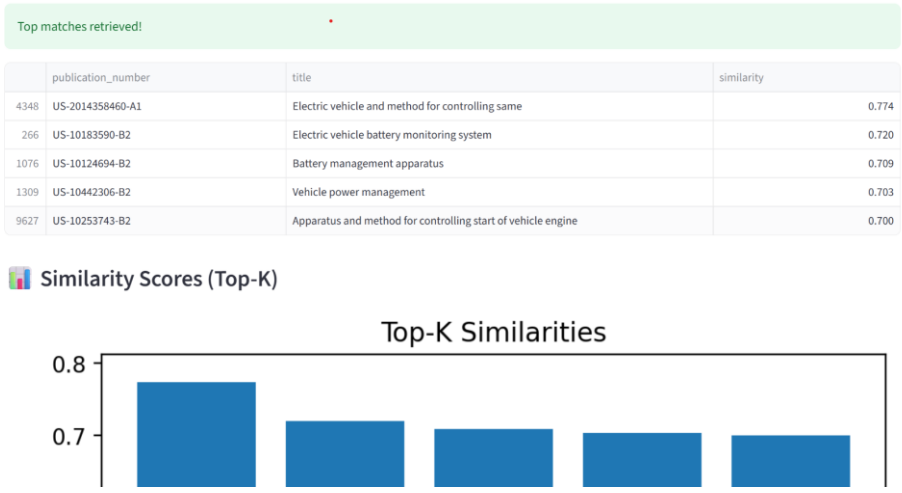

Figure 10: Results

3. For Batch Query (To upload list of patents to know details of Batch of query where uploaded query set should have query_claim_text column to search)

To check query set used in Experiment 1

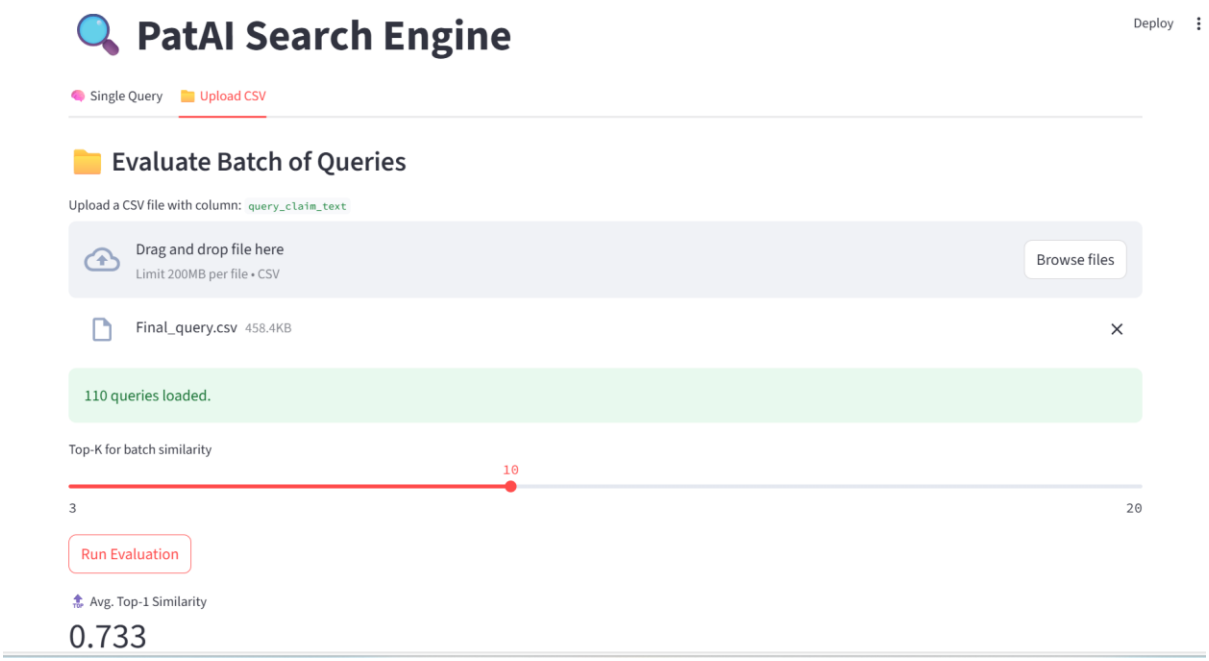

Figure 11: Batch Query Screen

Following this manual will allow you to fully reproduce and test the EV Patent Semantic Search system as demonstrated, using the provided code, datasets, and additional files for UI.