National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Nithin Reddy Yelala
Student ID: 23416858

School of Computing
National College of Ireland

Supervisor: Barry Haycock

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Nithin Reddy Yelala
Student ID:	23416858
Programme:	MSc in Data Analytics **Year:** 2025
Module:	Research Practicum
Lecturer:	Barry Haycock
Submission Due Date:	11-12-2025
Project Title:	Embedding Model Evaluation in Legal RAG systems for Automobile Law Question Answering
Word Count:	1043 **Page Count:** 10

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:	Nithin Reddy Yelala
Date:	11-12-2025

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

Embedding Model Evaluation in Legal RAG Systems for Automobile Law Question Answering

Nithin Reddy Yelala
23416858

1. Introduction

Retrieval-Augmented Generation (RAG) has become a crucial technique for building domain-specific question-answering systems, especially in fields where legal, regulatory, and procedural knowledge must be retrieved with precision. The complexity of Indian automobile laws, combined with the need for accurate interpretation, creates challenges for traditional NLP models that rely solely on prompt-based LLM responses. These models lack grounding in authoritative legal sources and cannot guarantee factual accuracy.

To address these issues, this project develops a comprehensive RAG pipeline capable of extracting, embedding, retrieving, and reasoning over legal texts derived from **The Motor Vehicles Act, 1988 (Government of India)**. The system converts law documents into structured text chunks, encodes them into semantic embeddings, stores them in a vector database, retrieves relevant chunks based on user queries, and finally generates context-aware, legally grounded answers using an LLM.

A major focus of this project is evaluating multiple embedding models—including All-MiniLM-L6-v2, bge-large-zh, UAE-Large-V1, Qwen3-Embedding-0.6B, InLegalBERT, KoSimCSE-roberta-multitask, all-mpnet-base-v2, and multilingual-e5-large—to determine which model provides the highest semantic alignment and retrieval quality for Indian automobile law queries.

This system is designed for:

- Legal assistants and law researchers
- Traffic enforcement analysis
- Automobile law consultancy
- Intelligent legal chat systems
- AI-assisted legal document search

Through structured text preprocessing, embedding comparison, RAG assembly, and performance evaluation, this project aims to establish an efficient and reliable legal QA pipeline grounded in authoritative legislative sources.

2. Research Question

Primary Research Question:

How effectively do different embedding models support retrieval-augmented legal question answering for Indian automobile laws, and which model provides the most accurate and semantically aligned responses within a RAG pipeline?

Detailed Explanation

Indian automobile laws are highly specific, context-driven, and text-dependent. The quality of retrieval depends strongly on the embedding model's ability to:

(A) Legal Semantic Understanding

Evaluate whether embeddings accurately capture:

• Legal terminology and definitions
• Penalties, clauses, sections, and subsections
• Contextual interpretation of legal obligations
• Domain-specific phrasing found in government legislation

The research question investigates whether some models—particularly multilingual or legal-domain models—encode legal semantics more effectively.

(B) Retrieval Precision and Ranking Quality

Determine how embedding choice affects:

• Retrieval of correct law sections
• Ranking of relevant chunks at higher positions
• Reduction of noise in retrieved legal text
• Ability to match a user query to its specific legal counterpart

Because correct retrieval is critical for proper answer generation, the study compares models on recall@k, precision@k, and nDCG to show how embedding models influence retrieval quality.

3. Environment Setup

Proper environment setup ensures consistent processing, embedding creation, retrieval operations, and LLM-based answer generation.

3.1 Recommended Hardware

Component	Recommended
GPU	NVIDIA T4 / RTX 3060 / A100 (optional but beneficial for large embedding models)
RAM	Minimum 8 GB, recommended 16–32 GB
CPU	Quad-core or better
Storage	5–10 GB for datasets + embeddings + vector DB

Most steps run smoothly on CPU; GPU helps for faster embedding generation for large models.

3.2 Operating System

Compatible OS:

• Windows 10/11
• macOS
• Linux (Ubuntu recommended for scalability)

3.3 Python Version

Python **3.10 or 3.11** is recommended.

The codebase and dependencies have been tested successfully on these versions.

3.4 Required Libraries

Core Libraries

• numpy
• pandas
• matplotlib
• seaborn

NLP + Embedding

• sentence-transformers
• HuggingFace transformers
• langchain
• langchain-community
• chromadb

Machine Learning / Evaluation

• scikit-learn

Utility

• os
• re
• json
• tqdm

3.5 Installation Commands

pip install numpy pandas matplotlib seaborn scikit-learn
pip install sentence-transformers transformers
pip install langchain langchain-community
pip install chromadb
pip install pypdf python-dotenv

If GPU is available:

pip install torch --index-url https://download.pytorch.org/whl/cu118

4. Dataset Description

The dataset is the Motor Vehicles Act, 1988 from indiacode.nic.in. It includes definitions, licensing rules, offenses, traffic regulations, and legal procedures. The PDF is converted to structured text for embedding creation.

5. Data Preprocessing

Text is extracted, cleaned, and chunked (512–1000 tokens) while preserving sections. Metadata like section numbers and chunk IDs are added. This ensures uniformity for vector database storage.

[7]:

```
def preprocess_text(text):
    text = re.sub(r'Page \d+|arXiv preprint.*', '', text)
    text = re.sub(r'\.{2,}', '.', text)
    text = re.sub(r'\n\s*\n', '\n\n', text)
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'Section\s*\d+[A-Za-z]*[:.]?', '', text)
    return text.strip()

cleaned_docs = [
    Document(page_content=preprocess_text(doc.page_content), metadata=doc.metadata)
    for doc in docs
]

splitter = RecursiveCharacterTextSplitter(
    chunk_size=350,
    chunk_overlap=100,
    separators=["\n\n", ".", ";", ":"]
)


chunks = splitter.split_documents(cleaned_docs)
print(f"Split into {len(chunks)} chunks")
```

```
Split into 1336 chunks
```

+ Code + Markdown

6. Exploratory Data Analysis

EDA examines text distribution and legal complexity using word clouds, frequency plots, section-wise volumes, and n-gram patterns. Insights guide chunking and preprocessing decisions.

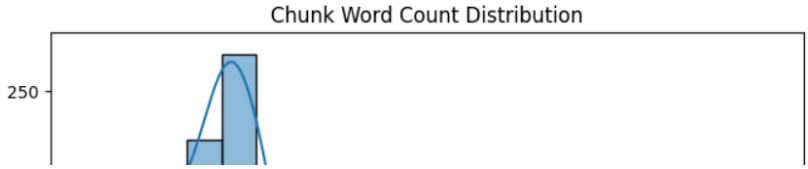
[11]:

```
# Convert chunks to DataFrame for easy EDA
df = pd.DataFrame({
    "content": [doc.page_content for doc in chunks],
    "length": [len(doc.page_content.split()) for doc in chunks]
})

print(" Total Chunks:", len(df))
print(df.head())

#  1. Basic statistics
print("\n Text Length Summary:")
print(df["length"].describe())
```

```
 Total Chunks: 1336
                                             content  length
0  1 THE MOTOR VEHICLES ACT, 1988 _____ AR...      51
1  . Necessity for driving licence. 4. Age limit ...      54
```

[8]:
```
# 2. Visualize chunk length distribution

plt.figure(figsize=(8,5))
sns.histplot(df["length"], bins=20, kde=True)
plt.title("Chunk Word Count Distribution")
plt.xlabel("Number of Words per Chunk")
plt.ylabel("Frequency")
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/seaborn/_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecate
d and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

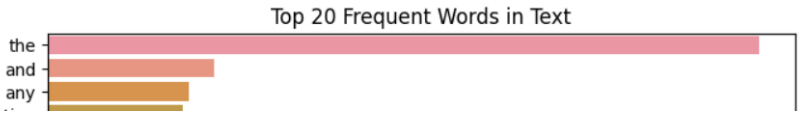


[9]:
```
# 3. Word Cloud for full document

all_text = " ".join(df["content"])
wordcloud = WordCloud(width=1000, height=500, background_color='white', colormap='viridis').generate(all
plt.figure(figsize=(12,6))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title("Most Frequent Words in Document")
plt.show()
```

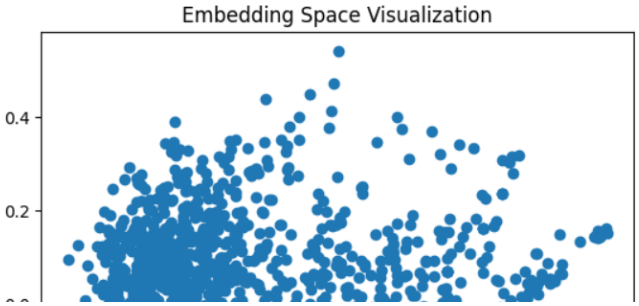
[10]:
```
#  4. Check duplicate or near-duplicate chunks
unique_chunks = df["content"].nunique()
duplicates = len(df) - unique_chunks
print(f" Duplicate Chunks Found: {duplicates}")
```

```
Duplicate Chunks Found: 12
```

[11]:
```python
#  5. Find top frequent words

from collections import Counter
import re

def get_top_words(texts, n=20):
    words = re.findall(r'\b[a-zA-Z]{3,}\b', " ".join(texts).lower())
    return Counter(words).most_common(n)

top_words = get_top_words(df["content"])
top_df = pd.DataFrame(top_words, columns=["word", "frequency"])

plt.figure(figsize=(8,5))
sns.barplot(data=top_df, x="frequency", y="word")
plt.title("Top 20 Frequent Words in Text")
plt.show()
```



[12]:
```python
#  6. Semantic Embedding Visualization

from sklearn.decomposition import PCA
embeddings = [embedding_model.embed_query(doc.page_content) for doc in chunks]
embeddings = np.array(embeddings)

reduced = PCA(n_components=2).fit_transform(embeddings)
plt.scatter(reduced[:,0], reduced[:,1])
plt.title("Embedding Space Visualization")
plt.show()
```



7. Embedding Model Architectures

Eight embedding models are evaluated, including multilingual-e5-large (best) and InLegalBERT. Each model uses identical pipelines to ensure fair comparison.

EMBEDDING MODEL

[14]:
```python
embedding_model = HuggingFaceEmbeddings(
    model_name="law-ai/InLegalBERT",
    model_kwargs={'device': 'cuda'}
)
embedding_dim = len(embedding_model.embed_query("test"))
print(f'[info] Embedding dimention = {embedding_dim} ')
```

8. Training / RAG Pipeline Execution

Text chunks are embedded and stored in ChromaDB. User queries are matched to relevant chunks, which are fed to an LLM for context-aware answer generation.

VECTOR DATABASE

[21]:
```
persist_dir = "/kaggle/working/database/chroma_db"

# Check if directory exists

if not os.path.exists(persist_dir):
    os.makedirs(persist_dir)

# Initialize Chroma vector store (empty)
vector_store = Chroma(persist_directory=persist_dir, embedding_function=embedding_model)

# Add documents with progress bar
print("[info] Building Chroma vector store with progress bar...")
for chunk in tqdm(chunks, desc="Embedding & adding docs"):
    vector_store.add_documents([chunk])

# Persist to disk
vector_store.persist()
print("[info] Chroma vector store built and saved at:", persist_dir)

# Assign to db for retrieval
```

[22]:
```
# Load Chroma database from disk
persist_dir = "/kaggle/working/database/chroma_db"
db = Chroma(
    persist_directory=persist_dir,
    embedding_function=embedding_model  # same embedding model used before
)

print("[info] Chroma database loaded sucessfully!")
```

```
[info] Chroma database loaded sucessfully!
```

[23]:
```
import json
questions=[]
answers=[]
# Load full JSON files
with open("/kaggle/input/automobile-rag/Dataset/motor_vehicles_act_qa(50_questions).json") as f:
    data = json.load(f)
# for i in data:
#     questions.append(i.get('Question'))
#     answers.append(i.get('Answer'))
for i in data:
    questions.append(i.get('question'))
    answers.append(i.get('answer'))


print("Number of Questions:", len(questions))
print("Number of answers:", len(answers))
```

```
Number of Questions: 50
Number of answers: 50
```

LLM MODEL ¶

[16]:
```python
from langchain.prompts import PromptTemplate
from langchain.chat_models import ChatOpenAI
from langchain.chains import LLMChain
from langsmith.run_helpers import tracing_context

with tracing_context(project_name="MotorVehicleAct-RAG"):
    llm = init_chat_model("llama-3.1-8b-instant", model_provider="groq",temperature=0.1)

    #  Prompt with gold answer
    prompt_template = """
    You are an expert legal assistant trained on the Motor Vehicles Act, 1988.

    First, read the context carefully.
    Then, identify the key clauses or sections relevant to the question.
    Finally, answer using only those facts in a precise and legal tone.
    Limit your response to 1–2 sentences in such a way that It should be excatly similar to gold answer,
    Do not add extra explanation or commentary.
    If the answer cannot be found, respond exactly: "Not found in the documents."

    Context:
    {context}

    Question:
```

9. Evaluation Metrics

Retrieval quality is measured via Recall@k, Precision@k, nDCG, and cosine similarity. QA outputs are evaluated using EM, F1, ROUGE-L, BLEU, and embedding similarity.

[19]:
```python
import evaluate
from langsmith import traceable
from sklearn.metrics.pairwise import cosine_similarity

# rouge = evaluate.load("rouge")
# bleu_metric = evaluate.load("bleu")

rouge = evaluate.load("rouge")
bleu_metric = evaluate.load("bleu")

# ----------------------------------------
# Helper Functions
# ----------------------------------------

@traceable(name="qa_f1_score")
def qa_f1_score(pred, gold):
    """Compute token-level F1 between predicted and gold answers."""
    pred_tokens = pred.lower().split()
    gold_tokens = gold.lower().split()
    common = set(pred_tokens) & set(gold_tokens)
    if not common:
```

```python
avg_retrieval = np.mean(retrieval_scores, axis=0)
avg_qa = np.mean(qa_scores, axis=0)

print("\n========== Evaluation Summary ==========")
print("\n-----  Retrieval Metrics -----")
print(f"Recall:    {avg_retrieval[0]:.4f}")
print(f"Precision: {avg_retrieval[1]:.4f}")
print(f"nDCG:      {avg_retrieval[2]:.4f}")

print("\n-----  QA System Evaluation -----")
print(f"Exact Match:       {avg_qa[0]:.4f}")
print(f"Token-level F1:    {avg_qa[1]:.4f}")
print(f"ROUGE-L (F1):      {avg_qa[2]:.4f}")
print(f"BLEU:              {avg_qa[3]:.4f}")
print(f"Cosine Similarity: {avg_qa[4]:.4f}")
print("========================================\n")
```

Loading widget...