

LegalRAG: an LLM based RAG system to provide legal literacy and support

DISSERTATION

Submitted in partial fulfillment of the requirements of the
Degree : MTech in Artificial Intelligence and Machine Learning.

By

Sachin Shankar Hebbar
2022AC05016

Under the supervision of

Naveen Rathani, Quantitative Analytics Manager

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE
Pilani (Rajasthan) INDIA

January, 2025

Abstract

The complex world of India's legal system, with its long history and vast literature, can intimidate many non-professionals. However, everyone needs to have a reasonable level of legal literacy to protect themselves from harm of different kinds. The advent of Large Language Models (LLM), in conjunction with the availability of vast amounts of public data regarding laws, acts, and cases, provides an opportunity to build a system that can act as the bridge between the legal system and the common people.

This work aims to build LegalRAG, a RAG-based system that uses an LLM to provide relevant responses and resources to users' legal queries related to Legal acts of inheritance, divorce, copyright, and consumer protection.

LLMs without any fine-tuning can be used to accomplish the task mentioned above. LLMs might have included legal information in their training datasets; they also carry tremendous knowledge through their parameters. However, this is general knowledge and is not explicitly tuned to contain Indian legal understanding. Other methods, like vanilla fine-tuning and Low-Rank Adapters (LoRA), would fine-tune the LLMs and equip them to answer legal queries, but these require more memory and resources to fine-tune the additional layers.

The proposed method of LegalRAG includes intent recognition followed by Retrieval Augmented Generation (RAG) architecture. User query passes through an intent recognition model to identify user intent: inheritance, divorce, consumer protection, or copyright-related intent. Depending on the intent, the user query is routed to different RAG systems or knowledge bases.

RAG is a method that combines the LLM's generation capabilities with information retrieval capabilities. Each of the four RAG systems takes in the user query, embeds it, and searches a vector store or knowledge base of Indian Legal data corresponding to one of the four acts. Most relevant context from the search and the user query are sent to the LLM to generate a response.

The hindrances from the previous approaches are resolved through RAG-based architecture as they provide the domain knowledge the LLM would need to generate relevant responses and not hallucinate. This approach requires fine-tuning the intent recognition model but doesn't mandatorily require any fine-tuning of the LLM itself. Therefore, it consumes less time and resources. RAG has the additional benefit of source

verification by providing information on the sources, such as the Act name and section number containing the selected context.

The novelty in this idea is with the intent recognition model, as more legal acts get included in the future; only the intent recognition model, which is much smaller in size than the LLM, needs to be re-trained without many updates to the LLM, therefore this approach requires less memory and resources..

LegalRAG would also include a Named Entity Recognition model to identify Personally Identifiable Information of any individuals and mask them before storing the data in vector stores or knowledge bases to protect the individual's data.

LegalRAG will benefit the general public by providing essential legal literacy and offering procedural steps and templates for filing cases under the abovementioned acts.

Key Words:

Large Language Model (LLM), Named Entity Recognition (NER), Intent Recognition, fine-tuning, Low-Rank Adapters, Retrieval Augmented Generation (RAG), Embedding model, Natural Language Processing (NLP), Legal Literacy

List of Symbols and Abbreviations used

Abbreviation	Full Form
RAG	Retrieval Augmented Generation
PII	Personally Identifiable Information
LLM	Large Language Model
LoRA	Low Rank Adapter
Q-LoRA	Quantized Low Rank Adapter
BAAI	Beijing Academy of Artificial Intelligence
BGE	BAAI General Embedding
HyDE	Hypothetical Document Embeddings
NER	Named Entity Recognition
UI	User Interface

List of Tables

Table 1: Example records for each of the five intents from the training dataset.....	18
--	----

List of Figures

Figure 1: Architecture of LegalRAG.....	11
Figure 2: Architecture of Data masking, chunking, embedding and storing.....	13
Figure 3: Text from a divorce case.....	15
Figure 4: Text from the divorce case after masking.....	15
Figure 5: Code snippet showing chunking.....	15
Figure 6: Embedding model.....	16
Figure 7: Chroma db vector store.....	17
Figure 8: DistilBERT.....	19
Figure 9: Training distilBERT.....	20
Figure 10: Other intent.....	20
Figure 11: Divorce intent.....	21
Figure 12: Hypothetical Document Embeddings.....	23
Figure 13: HyDE output.....	23
Figure 14: Retrieved Context.....	24
Figure 15: LLM and Evaluating LLM.....	25
Figure 16: LLM response.....	25
Figure 17: Relevancy Evaluator.....	26
Figure 18: Faithfulness Evaluator.....	27
Figure 19: Evaluator response and feedback.....	27

Table of Contents

Abstract	2
List of Symbols and Abbreviations used	4
List of Tables.....	5
List of Figures	6
Table of Contents.....	7
Chapter 1	8
Chapter 2	13
Chapter 3	18
Chapter 4	22
Chapter 5	26
Future work	28
Bibliography/References	29

Chapter 1

Purpose of the project:

The Indian Legal system is very elaborate and can be very difficult for common folks to understand. This project named 'LegalRAG' aims at building a system that would make it easier for the general public to access legal literacy and basic legal support using a RAG [1][2] system.

For the purpose of this project, legal acts and data related to inheritance, divorce, consumer protection and copyright laws in India will be used. No other legal acts and areas will be used.

Expected Outcome:

A system called LegalRAG that would take in textual user query related to legal systems of India and get a relevant response along with any additional resources and sources cited to ensure the response is trustworthy.

Objectives:

The following are the objectives listed in the abstract submission document of the project LegalRAG:

- To provide legal literacy to the general public of India.
- To provide basic legal support for filing cases to the common folks of India.
- To provide source verification and resources for all the responses the system provides and enhance user trust.
- To ensure consumer data is protected and masked.

The first objective regarding legal literacy and the second objective of providing filing case procedures and documents required have been partially met as the RAG (Retrieval Augmented Generation) systems are built to answer the user queries related to the acts of inheritance, divorce, consumer protection and copyright as listed in the abstract.

The third objective of source verification and resources have been partially met as the RAG systems built provide relevant context to every response it provides.

The fourth objective is related to identifying and protecting consumer's PII (Personally Identifiable Information) through masking has been met as all the data collected has been passed through a Named Entity Recognition model that identifies all PII data and replaces them with a mask, such as ****.

Literature Review:

Literature review related to RAG methodologies and other LLM fine-tuning methods like LoRA are shared in the Literature References section.

Based on the review of these papers, it is noted that LLM fine-tuning methods like LoRA require more memory and resources to tune the additional layers. However, RAG doesn't mandatorily require fine-tuning an LLM, therefore it is not as resource intensive as fine-tuning an LLM. The availability of several ways of information retrieval from vector stores in RAG also makes it an optimal choice.

The references and blog articles that provide more information on the legal systems of India are also shared in the same section.

Existing Processes and its limitations:

A vanilla LLM or an LLM without fine-tuning can be used to answer Indian legal system related user queries. This is because many LLMs may have Indian legal system data in their training data. LLMs have a huge number of parameters and they contain tremendous knowledge through these parameters.

Some proprietary LLMs may have access to the internet and surf the internet to provide the relevant responses.

LLM fine-tuning methods like LoRA [3] and Q-LoRA [4] exist that can add additional layers to the LLM and fine-tuning these layers on Indian legal system data can also be used to provide relevant response to user queries.

All these methods will provide good results but they have some limitations, such as:

1. A vanilla LLM may hallucinate and provide responses that are not correct.
2. A vanilla LLM's response may not always reflect the latest changes that have been done to the Indian legal system as the training data is older.
3. Proprietary LLMs have a cost and still may hallucinate or provide responses without the links for sources.
4. In some cases, the data may be proprietary and may not be part of the training data and may not be present in the internet (which is not the case for Indian legal data, but it is true for different scenarios)
5. LLM fine-tuning will require additional memory and computational resources for the fine-tuning of these newly added layers to the LLM.

Justification for the chosen methodology:

LegalRAG consists of three components:

1. Masking Personally Identifiable data using Named Entity Recognition - This step is necessary as the legal acts data and the data related to cases could contain names of the individuals, their age and many other personally identifiable Information (PII) that needs to be masked to protect the individual. A named entity recognition model

would identify all the PII data and replace them with '***' or [MASK] before storing the data in the vector store. This is an important step to protect the information of individuals involved in cases.

2. Intent Classification - This step is needed because the goal of the project is to provide relevant responses for four different legal acts. It is possible to have all these legal acts stored in the same location, but storing them in different locations and using an intent classification model to route the user query to the relevant legal act storage location will improve the response accuracy of the LLM.
3. RAG system
 - a. RAG is an effective method to include non-parametric domain knowledge to an LLM's response without fine-tuning the LLM. Therefore it saves the memory and computational cost that comes from fine-tuning.
 - b. RAG system also provides sources from where it got the context for a particular user query, this is a simpler way to ensure the LLM is not hallucinating.
 - c. The RAG architecture makes it easy to update the domain knowledge frequently to reflect the updated Indian legal data as it requires a simple vector store update instead of re-fine tuning the LLM.

Project Components:

Below architecture shows the main project workflow from the user query to the relevant response:

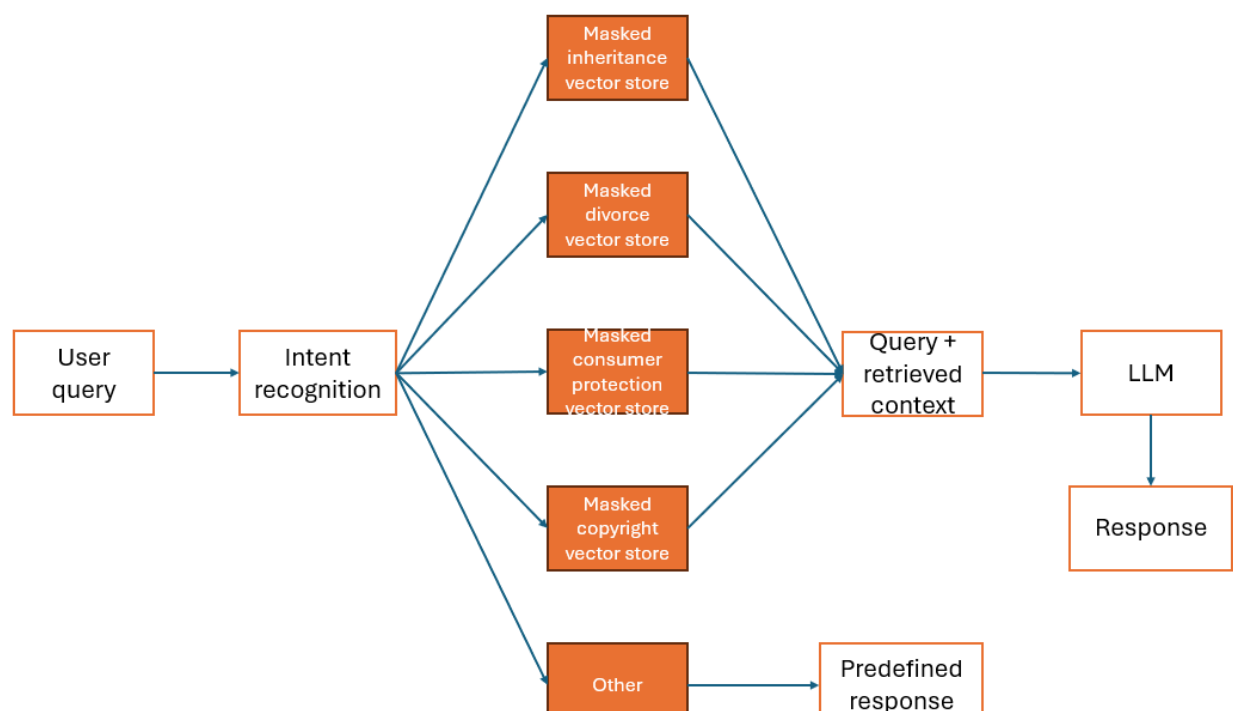


Figure 1: Architecture of LegalRAG

User query is taken into an intent classification or intent recognition model. The intent will be one of the five: inheritance, consumer protection, divorce, copyright or Other. Based on the user intent, the query is routed to the relevant vector store. If the intent is 'Other' then there is no data for the LLM to answer that question, so a predefined response such as 'Please ask questions related to inheritance, divorce, copyright or consumer protection' will be shared with the user.

The data related to the above mentioned four acts will pass through NER to identify PII data and mask them. The masked data would be chunked, embedded and indexed in the relevant vector store.

The user query that is routed to the relevant vector store based on the user intent will be embedded (using the same embedding model as that of the masked text) and searches the masked data vector store to get the relevant context.

The relevant context along with the user query is sent to the LLM to generate the response to the user query along with the source information to ensure trust with the provided response.

Platform and softwares used for the project:

1. Spacy [5] for Named Entity Recognition
2. LlamaIndex [6] for RAG
3. HuggingFace [7] for the LLMs and the embedding models
4. Python [8] for coding

Novelty in the method:

The use of an intent recognition model before the RAG systems is the novelty in this method. Intent recognition is very helpful because as more legal acts get added, only the intent recognition model will need to be re-trained or fine-tuned to involve more classes or intents without making any updates to the LLM.

This is useful as the intent recognition model is much smaller than LLM and therefore will consume less memory and resources to re-train.

This approach makes it easier to add more legal acts and make the LegalRAG system even more comprehensive as time passes without much updates to the LLM.

Components of LegalRAG:

LegalRAG has three main components as listed in the abstract:

1. Legal Data collection and Named Entity Recognition
2. Intent Classification (Novelty)

3. RAG systems using LLMs (Large Language Models)

In the following chapters, this document will further explain each of these components and how they put together accomplish the objectives mentioned above.

Chapter 2

Data Collection:

Data was collected from the following locations regarding the four acts in consideration (copyright, inheritance, consumer protection and divorce):

1. Department of Justice of India [9]
2. Legislative Department of India [10]
3. Indiafilings blog website [11]
4. LegalServiceIndia blog website [12]
5. iPLEaders blog articles (from LawSikho) [13]

Data for each of the four acts collected from these multiple locations were combined to form a single document. Ensuring there is one document per act. This is not a mandatory step as the data reading softwares and platforms are capable of handling multiple documents per act, but this was done as a step to ensure there is manual care, trust and oversight on the data being collected.

Once the data has been collected, the following architecture captures the next steps:

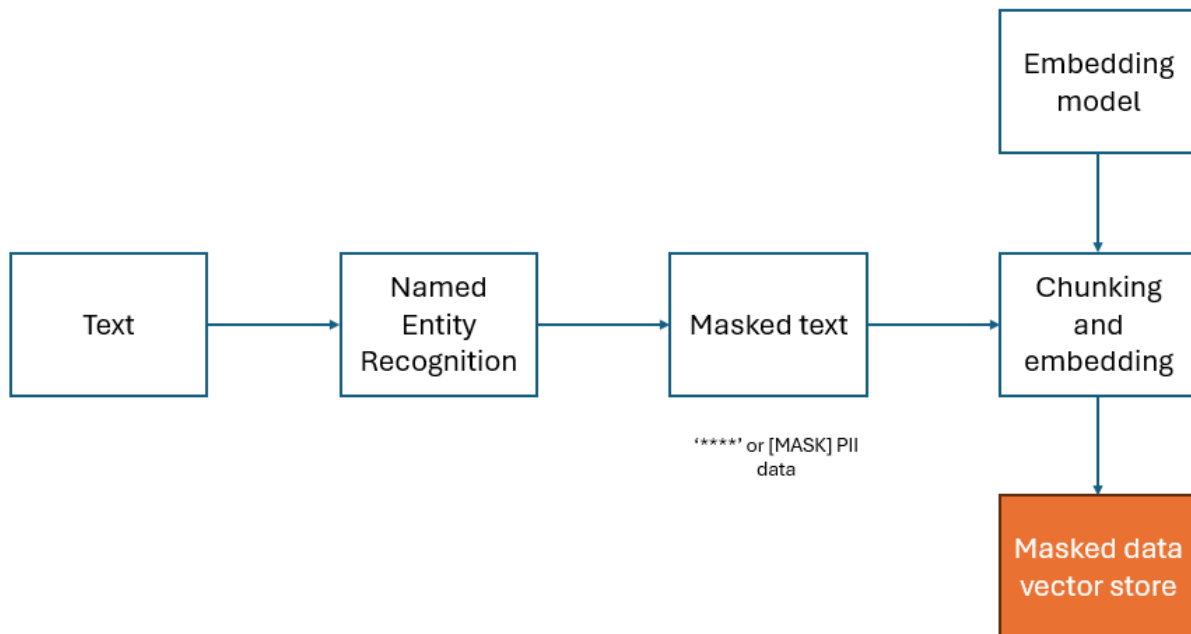


Figure 2: Architecture of Data masking, chunking, embedding and storing

Note that the above architecture will repeat four times, for each of the four acts. So that, each of the four act's PII data gets masked, chunked, embedded and stored in a separate vector store. All the subsections below take about each of these steps shown in the above architecture.

Named Entity Recognition (NER) model:

After all the data was collected, it was found that this data had consumer PII data in some cases. The following are the type of PII data that can be found under each act:

PII data present in inheritance act:

- Names of the deceased person and heirs
- Date of birth and age of individuals
- Family relationship and names (example: spouse, children, siblings)
- Aadhaar number, PAN
- contact numbers
- valuation of the inheritance
- address of the inheritance location and will information

PII data present in divorce act:

- Names of both spouses
- date of birth and age of individuals
- date and place of marriage, marriage certificate information
- names and ages of children, dependents, guardians involved
- residential address and phone number
- aadhaar number and PAN

PII data present in consumer protection act:

- Name of the complainant (consumer) and the respondent (business)
- Age and date of birth
- address and phone number, email addresses
- credit card or debit card information
- dates of transaction

PII data present in copyright act:

- Name of the copyright owner
- date of birth and age
- aadhaar, PAN and phone number
- Copyrighted work case and docket numbers
- Contracts, agreements and license numbers

These PII data have to be protected to ensure no leakage of consumer data. Therefore a Named Entity Recognition model provided by SpaCy has been used to mask this data. All the PII data has been replaced with ****.

Here is an example of the original data and the data after masking:

Original data about a divorce case that contains the name of the divorcee:

```
text = """
The Shah Bano case became a landmark in Indian legal history, setting a precedent for maintenance rights for Muslim women.
Shah Bano, a Muslim woman, was divorced by her husband and sought maintenance under Section 125 of the Criminal Procedure Code.
The Supreme Court initially ruled in her favor, but due to political pressure, the government passed the Muslim Women (Protection of Rights on Divorce) Act in 1986,
limiting the maintenance duration for Muslim women. This case highlighted the need for a uniform civil code to ensure equal rights for women across all religions.
"""
```

Figure 3: Text from a divorce case

The data after NER identified the person's name and masked it:

```
'\nthe **** case became a landmark in indian legal history, setting a precedent for maintenance rights for muslim women.\n****, a muslim woman, was divorced by her husband and sought maintenance under section 125 of the criminal procedure code. \nthe supreme court initially ruled in her favor, but due to political pressure, the government passed the muslim women (protection of rights on divorce) act in 1986, \nlimiting the maintenance duration for muslim women. this case highlighted the need for a uniform civil code to ensure equal rights for women across all religions.\n\n'
```

Figure 4: Text from the divorce case after masking

Data Ingestion and storage (Chunking, Embedding, and Indexing):

The masked data will be chunked, embedded and indexed and stored in the vector store. There will be four vector stores, one for each of the four legal acts.

Chunking:

The masked data or text for each act will be divided into chunks. Each chunk can contain multiple sentences. There are many ways of chunking. As of writing this document, the following approach has been used for chunking:

1. Each chunk will have 100 tokens (words and special characters) in it.
2. Chunk overlap of 50 will be used. This implies each chunk will contain the last 50 tokens of the previous chunk, and the first 50 tokens of the next chunk

Here is a code snippet that shows chunking:

```
text_splitter = SentencesSplitter(
    separator=" ",
    chunk_size=100,
    chunk_overlap=50
)
```

Figure 5: Code snippet showing chunking

Chunking is done to make it easy to store the data as vector embeddings in the vector store. Chunking also helps localize the context. So whenever a user query is used to search the vector store, relevant context can be easily obtained through these chunked embeddings.

Chunk overlap helps provide additional context to each chunk and enriches the information within each chunk.

Embedding:

After chunking the masked data, an embedding model is used to convert these chunks into vector embeddings. There is a need to convert the chunks to numerical vector embeddings as the vector store only stores numerical vector data.

The Embedding model used is bge-small-en-v1.5 [14]

Here bge refers to BAAI General Embedding.

A code snippet showing the embedding model being downloaded

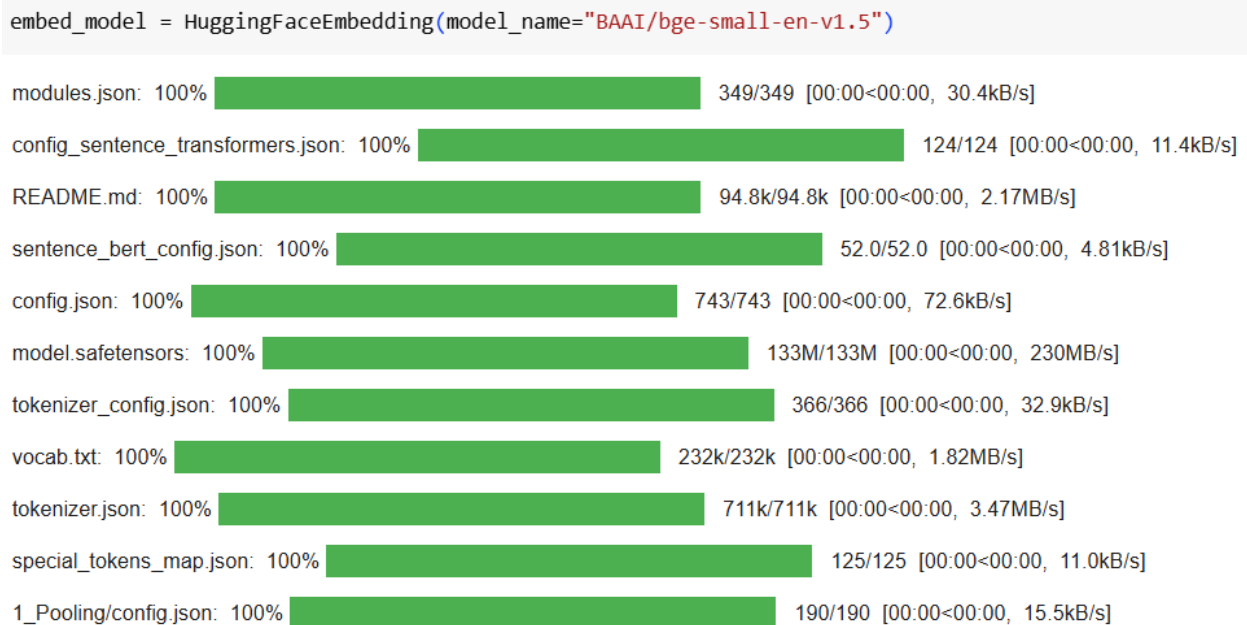


Figure 6: Embedding model

This is an open-source embedding model available in HuggingFace. This has been created by BAAI (Beijing Academy of Artificial Intelligence).

The choice for this embedding model was based on the following factors:

1. It is open source and freely available
2. It is a smaller embedding model which produces vector embeddings of size 1024.
Therefore consumes lesser resources and memory

Indexing and storing in Vector store:

After embedding the data, it is stored in a vector store. Vector store used here is Chromadb [15] which is a freely available, open-source vector store.

A code snippet showing data getting stored in Chromadb, and later getting retrieved

```
chroma_client = chromadb.EphemeralClient()
chroma_collection = chroma_client.create_collection("quickstart")

vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
storage_context = StorageContext.from_defaults(vector_store=vector_store)
index = VectorStoreIndex(nodes, storage_context=storage_context, embed_model=embed_model)

# save the data in a disk
db2 = chromadb.PersistentClient(path="./chroma_db")
# saves the data as chroma.sqlite3

# loading the data from disk
chroma_collection = db2.get_or_create_collection("quickstart")
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
index = VectorStoreIndex.from_vector_store(
    vector_store,
    embed_model=embed_model,
)
```

Figure 7: Chroma db vector store

Benefits of storing data in vector store:

1. Vector stores are most suitable for storing numerical vectors
2. Makes it easy to search the vector store through indexing
 - a. Saves the data in a tree-like structure where each branch is a subspace
 - b. When a user query searches through this tree-like structure it's easier to pass through the structure to reach the subspace which has the highest similarity to the user query.

This completes the first component of LegalRAG, where data is collected and after various processing steps such as Named Entity Recognition, Chunking, Embedding and Indexing is safely stored in a vector store. This step makes it easy for the RAG systems to retrieve relevant context for the user query.

Chapter 3

The second component of the LegalRAG is Intent Recognition or Intent Classification. This intent recognition model was trained as a multi-class classification task. There were 5 intents. One intent for each of the four legal acts: divorce, consumer protection, copyright, and inheritance. There was a fifth intent called 'Other' which was used to provide a template response to the user that the LegalRAG system can only respond to the queries related to the four acts as of writing this document.

Training and test Dataset creation:

A train dataset was created manually ensuring 40 examples for each intent. Care was taken to ensure the examples for each intent was as diverse as possible to ensure the model is better able to predict the correct intent.

The following table shows 3 records from the train dataset for each of the intent:

Query	Intent
What are the grounds for divorce?	Divorce
How long does the divorce process take?	Divorce
Can I file for divorce online?	Divorce
What is the time limit for filing a consumer complaint?	Consumer protection
How to report misleading advertisements?	Consumer protection
What is the process for getting a refund for a faulty product?	Consumer protection
Who inherits my father's wealth?	Inheritance
How to resolve inheritance disputes?	Inheritance
What is the process of writing a will?	Inheritance
What are my rights as a copyright owner?	Copyright
What is copyright infringement?	Copyright
How to register a copyright?	Copyright

Where is the closest hospital?	Other
How to get rich?	Other
What is the best way to learn a new language?	Other

Table 1: Example records for each of the five intents from the training dataset

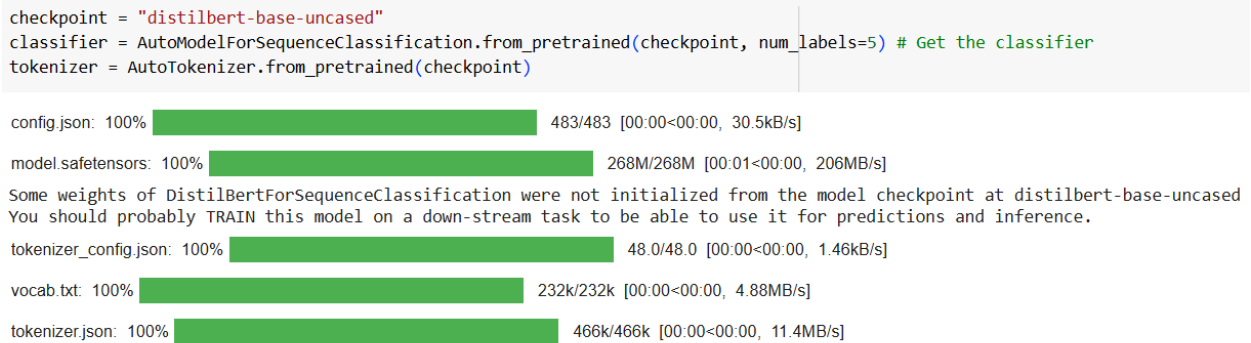
A test set was created with 10 records for each of the intent, ensuring that these records are not the same as the ones in the training dataset

Intent Recognition model:

DistilBERT [16] was used to create the intent recognition model. The pre-trained version of distilbert was taken from HuggingFace: distilbert-base-uncased

A code snippet on getting distilBERT

```
checkpoint = "distilbert-base-uncased"
classifier = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=5) # Get the classifier
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
```



```
config.json: 100% 483/483 [00:00<00:00, 30.5kB/s]
model.safetensors: 100% 268M/268M [00:01<00:00, 206MB/s]
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.46kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 4.88MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 11.4MB/s]
```

Figure 8: DistilBERT

Using the training dataset created, fine-tuning was done on the DistilBERT so that it works as a five-class classification task.

Reasons for choosing Distilbert:

1. DistilBERT is smaller in size compared to other BERT or transformer models.
 - a. Size: 44M parameters (40% of BERT base [17])
2. DistilBERT has been trained on multiple tasks and saved in HuggingFace, and only fine-tuning is required

The intent classification model was fine-tuned for 5 epochs and reached convergence with the following values on the conventional classification metrics:

1. Validation loss (cross entropy loss): 0.08
2. Accuracy: 0.96
3. Precision: 0.966
4. Recall: 0.96

5. F1 score: 0.96

[125/125 04:25, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	No log	0.710698	1.000000	1.000000	1.000000	1.000000
2	No log	0.175783	0.980000	0.981818	0.980000	0.979950
3	No log	0.095075	0.980000	0.981818	0.980000	0.979950
4	No log	0.088341	0.980000	0.981818	0.980000	0.979950
5	No log	0.081049	0.960000	0.966667	0.960000	0.960766

Figure 9: Training distilBERT

The metric values are quite good and add some trust to the intent classification model. Here are some results for unseen queries:

For a user query unrelated to any of the four legal acts:

```
# Example input text
new_text = "Hope I get rich someday"

# Tokenize the new text (same as during training)
inputs = tokenizer(new_text, return_tensors="pt", padding='max_length', truncation=True, max_length=128)

# Convert tokenized input into a Dataset
predict_dataset = Dataset.from_dict({
    'input_ids': inputs['input_ids'],
    'attention_mask': inputs['attention_mask']
})

# Get predictions from the trainer
predictions = trainer.predict(predict_dataset)

# Extract the logits
logits = predictions.predictions

# Get the index of the highest logit (the predicted class)
predicted_label = torch.argmax(torch.tensor(logits), dim=1).item()

print(f"Predicted intent: {intent_label_dict[predicted_label]}")

Predicted intent: Other
```

Figure 10: Other intent

For a user query related to divorce:

```
# Example input text
new_text = "I just want divorce!"

# Tokenize the new text (same as during training)
inputs = tokenizer(new_text, return_tensors="pt", padding='max_length', truncation=True, max_length=128)

# Convert tokenized input into a Dataset
predict_dataset = Dataset.from_dict({
    'input_ids': inputs['input_ids'],
    'attention_mask': inputs['attention_mask']
})

# Get predictions from the trainer
predictions = trainer.predict(predict_dataset)

# Extract the logits
logits = predictions.predictions

# Get the index of the highest logit (the predicted class)
predicted_label = torch.argmax(torch.tensor(logits), dim=1).item()

print(f"Predicted intent: {intent_label_dict[predicted_label]}")

Predicted intent: Divorce
```

Figure 11: Divorce intent

Chapter 4

The third component of LegalRAG are the RAG systems, There will be four RAG systems for each of the four legal acts. Each of the RAG system has two parts:

1. Information Retrieval
2. Response Generation

Information Retrieval:

When a user enters a query then it passes through the intent recognition model and the user's intent is identified. Based on the user's intent, the query is routed to a vector store that contains the relevant legal act. For example: If the user's intent is 'Inheritance', then the query is routed to a vector store that contains the inheritance act data.

Chapter 1 described in detail how the relevant data will be stored in the vector store.

Before the user query is converted to a vector embedding and searches through the vector store, an additional step has been added to enrich the user query. This additional step is using HyDE [18] (Hypothetical Document Embeddings)

HyDE:

HyDE is a method where the user query passes through an LLM and the LLM generates additional information about the user query. This additional information called a hypothetical document can be accurate information or hallucinated information. It is not a problem even if it is hallucinated as the purpose of HyDE is to provide additional context about the user query or a way of enriching the user query.

Some user queries may not be detailed and if just the user query is sent to search the vector store it may not get the most relevant context. So enriching it with hypothetical document will be useful.

The hypothetical document and the user query together are embedded using the same embedding model that was used to embed the data in chapter 1: bge-small-en-v1.5

The embedding model has to be the same one as the one used in chapter 1 otherwise it will not be possible to search and retrieve the relevant context.

Following architecture shows HyDE:

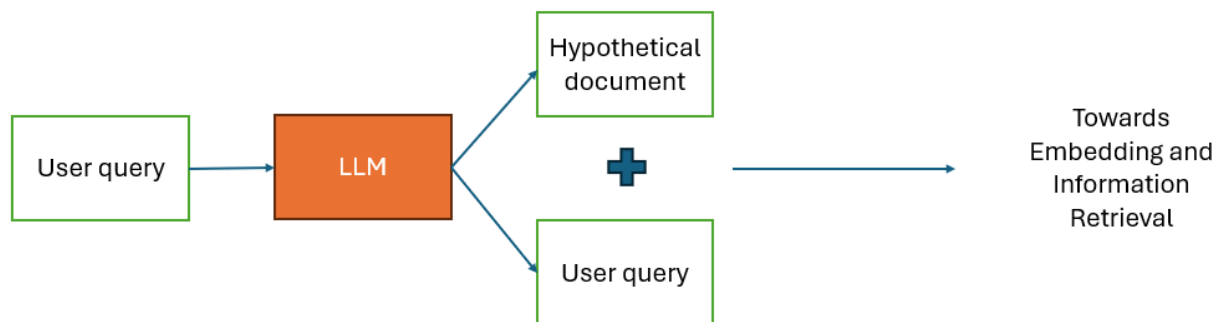


Figure 12: Hypothetical Document Embeddings

The LLM used to get the hypothetical document is the same as the LLM that is used to generate the response in the next step, but it will not be a problem even if it is a different one.

LLM used is Zephyr alpha [19], a 7 billion parameter model that is available in HuggingFace.

Reason for choosing Zephyr alpha:

1. It is a 7 billion parameter model, a smaller size considered to other LLMs, therefore requires less memory.
2. It is an open source model and requires no additional cost.

Here is a hypothetical document generated by HyDE for a question related to inheritance:

```

question3 = "List the documents required for legal heir certificate."
query_bundle = hyde(question3)
hyde_doc = query_bundle.embedding_strs[0]
print(hyde_doc)
response1 = query_engine.query(question3)
response2 = hyde_query_engine.query(question3)
  
```

To obtain a legal heir certificate, the following documents are required:

1. Death certificate of the deceased person.
2. Proof of identity and address of the applicant.
3. Proof of relationship with the deceased person. This can be in the form of a birth certificate, marriage certificate, or any other legal document that establishes the relationship.
4. Affidavit from the applicant stating that they are the legal heir of the deceased person.
5. Copy of the will, if any.
6. Copy of the succession certificate, if any.
7. Copy of the income tax returns of the deceased person for the last three years.
8. Copy of the bank passbook or statement for the last six months.
9. Copy of the land and property documents, if any.
10. Copy of the insurance policy, if any.

The documents required may vary depending on the state and the circumstances of the case. It is always advisable to check with the concerned authorities for the specific requirements.

Figure 13: HyDE output

Search and retrieval:

The hypothetical document and the query are converted to vector embedding using the same embedding model as before. This embedding model searches through the vector

store to find the chunks that are very similar to it. Cosine similarity is used to calculate the similarity between the query and the embeddings in the vector store.

The top 5 similar context are retrieved and used as context in the response generation

Here are the top 3 retrieved context taken from the document for a question related to divorce:

The question is: List the circumstances in which a wife can petition for divorce in India?

```
for node in response.source_nodes:
    print(node.text)
    print("-----")
```

Divorce Rules in India: Everything You Need to Know
The Indian Divorce Act governs divorce among the Christian couples in India. Divorce is the legal dissolution of the marital union between a man and a woman. According to this act, -----
The husband or wife can obtain a decree of judicial separation; on the ground of adultery or cruelty or desertion without reasonable excuse for two years, or more and such decree will be a ground for divorce.
Application for Juridical separation
The petition for juridical separation can be applied on the ground of adultery or cruelty or desertion without reasonable excuse for two years or more.

Divorce is followed by granting alimony, child custody, and child visitation, distribution of property and distribution of debts. Before opting for a divorce, the Christian couple should consult a lawyer.
The Indian Divorce Act – An Outline
The Indian Divorce Act was drafted into the Indian legal system in the year 1869. In India divorce rules and procedure varies according to the community of the couple.

Regarding the bank accounts, everything must be divided.
Duration of Divorce
The duration of divorce by mutual consent varies from six to 18 months, depending on the decision of the court.
Dissolution of Marriage (Divorce without Mutual Consent)
According to Indian Divorce act, either the husband or wife can file the petition for dissolution of Marriage. The conditions when they can file the petition are explained in detail below.
Petition by Husband
Any husband can present a petition to the District Court or the High Court, praying that his marriage needs to be dissolved on the ground that his wife has been guilty of adultery or cruelty or desertion without reasonable excuse for two years or more.

Figure 14: Retrieved Context

The above retrieved context will act as source verification for the user to ensure that the response they got is from the retrieved context which in turn is from legal data sources.

Response Generation:

The user query and the top 5 retrieved contexts are sent to the LLM to generate responses that will answer the user query. Note that the hypothetical document will not be sent as a context to the LLM.

The LLM used here is Zephyr alpha, a 7 billion parameter, open-source model available in HuggingFace. The reason for the choice of this LLM is same as that was explained in HyDE sub-section.

A code snippet showing the use of Zephyr alpha and Zephyr Beta [20] (more information in the next chapter of Zephyr Beta)

```
llm = HuggingFaceInferenceAPI(model_name="HuggingFaceH4/zephyr-7b-alpha", token=HF_TOKEN)
evaluating_llm = HuggingFaceInferenceAPI(model_name="HuggingFaceH4/zephyr-7b-beta", token=HF_TOKEN)
```


Figure 15: LLM and Evaluating LLM

Here is an LLM's response for the question related to inheritance:

Question: "Who can be the legal heir under class I of Hindu Law?"

Under the Hindu Succession (Amendment) Act, 2005, the following can be the legal heir of a person under Class I:

1. Wife (widow)
2. Mother
3. Son
4. Daughter
5. Deceased son's daughter
6. Deceased daughter's daughter
7. Daughter of a pre-deceased son of a pre-deceased son
8. Deceased son's wife (widow)
9. Wife (widow) of a pre-deceased son.

Figure 16: LLM response

The response matches with the one provided in the data source.

Chapter 5

The last chapter is on evaluating the results of the LegalRAG system. It is very important to evaluate the results as manual evaluation will not be possible every time. There are many ways to evaluate the LegalRAG, the methods considered here are two model based evaluation methods:

1. Relevancy Evaluator [21]
2. Faithfulness Evaluator [22]

Relevancy Evaluator

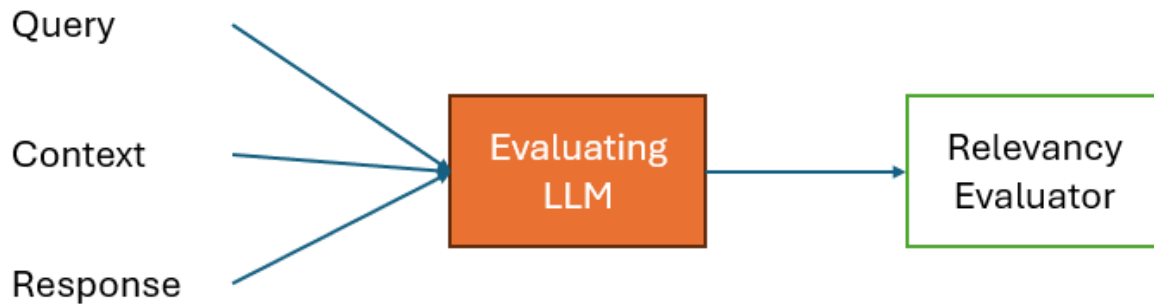


Figure 17: Relevancy Evaluator

The above architecture takes user query, context and the generated response as its inputs. It uses an Evaluating LLM to check whether the context retrieved is related to the user query or not. It also checks whether the user query seems to be answered in the generated response.

Evaluating LLM has to be a different one from the one used to generate responses in the RAG systems. It cannot be the same one as it may always indicate that the queries and contexts are associated and the user query was answered by the response generated.

The Evaluating LLM used here is another 7 billion parameter model, open source model available in HuggingFace: Zephyr Beta

Zephyr Beta is also used as the Evaluating LLM for Faithfulness Evaluator as well.

The output of Relevancy Evaluator will be True if the user query was answered in the response and if the retrieved context seemed to be associated with the user query. It will be False if that is not the case. It also provides textual feedback

Faithfulness Evaluator:

Faithfulness Evaluator is a hallucination check, where the evaluating LLM will take the retrieved context and the generated response and confirm if the response answers from the context or not. Faithfulness Evaluator is True if there is no hallucination, False if there is hallucination. It also provides textual feedback.



Figure 18: Faithfulness Evaluator

Here is a code snippet that shows both Relevancy Evaluator and Faithfulness Evaluator's responses and feedback for the question related to inheritance.

Question: "Who can be the legal heir under class I of Hindu Law?"

```
faithful_eval = faithfulness_evaluator.evaluate_response(response=response2)
relevancy_eval = relevancy_evaluator.evaluate_response(query=question, response=response2)

print(f"Faithfulness: {faithful_eval.passing}")
print("-----")
print(f"Relevancy: {relevancy_eval.passing}")
print(relevancy_eval.feedback)
```

Faithfulness: True

Relevancy: True

Based on the context provided, the response is in line with the context information as it lists out the individuals who can be legal heirs under Class I of Hindu Law, which is in accor

Figure 19: Evaluator response and feedback

For the above question, both Relevancy Evaluator and Faithfulness Evaluator print out True and the feedback makes sense too, therefore the response given by the LLM is correct.

Future work

The next steps of LegalRAG are the following:

1. Combine the different components of LegalRAG into a pipeline, so the user can get the full benefit of a smooth experience.
2. Create a UI (User Interface) to make the user experience better
3. Try different chunking techniques to improve retrieval and data storage
4. Continue to add more relevant data related to the four acts
5. Try more complex retrieval techniques to improve the retrieval mechanisms
6. Try prompt engineering to see if that improves the response from the LLM
7. Try more evaluation metrics like precision and recall by creating a dataset of known questions and answers. Have evaluation metrics for both information retrieval and response generation segments of RAG

Bibliography/References

- [1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks", arXiv:2005.11401, URL <https://arxiv.org/abs/2005.11401>
- [2] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan et al., "Retrieval-Augmented Generation for Large Language Models: A Survey", arXiv:2312.10997, URL <https://arxiv.org/abs/2312.10997>
- [3] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, "LoRA: Low-Rank Adaptation of Large Language Models", arXiv:2106.09685, URL <https://arxiv.org/abs/2106.09685>
- [4] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, Luke Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs", arXiv:2305.14314, URL <https://arxiv.org/abs/2305.14314>
- [5] URL <https://spacy.io/>
- [6] URL <https://www.llamaindex.ai/>
- [7] URL <https://huggingface.co/>
- [8] URL <https://www.python.org/>
- [9] Department of Justice, URL <https://doj.gov.in/>
- [10] Legislative Department of India, URL <https://legislative.gov.in/>
- [11] IndiaFilings, URL <https://www.indiafilings.com/>
- [12] LegalserviceIndia, URL <https://www.legalserviceindia.com/>
- [13] iPLEaders Blog, URL <https://blog.ipleaders.in/>
- [14] URL <https://huggingface.co/BAAI/bge-small-en-v1.5>
- [15] URL <https://www.trychroma.com/>
- [16] URL https://huggingface.co/docs/transformers/en/model_doc/distilbert
- [17] URL <https://huggingface.co/google-bert/bert-base-uncased>
- [18] URL <https://docs.haystack.deepset.ai/docs/hypothetical-document->

[embeddings-hyde](#)

[19] URL <https://huggingface.co/HuggingFaceH4/zephyr-7b-alpha>

[20] URL <https://huggingface.co/HuggingFaceH4/zephyr-7b-beta>

[21] URL

[https://docs.llamaindex.ai/en/stable/examples/evaluation/relevancy eval/](https://docs.llamaindex.ai/en/stable/examples/evaluation/relevancy_eval/)

[22] URL

[https://docs.llamaindex.ai/en/stable/examples/evaluation/faithfulness eval/](https://docs.llamaindex.ai/en/stable/examples/evaluation/faithfulness_eval/)

Check list of items for the Final report

- | | | |
|------|--|-------|
| a) | Is the Cover page in proper format? | Y |
| b) | Is the Title page in proper format? | Y |
| c) | Is the Certificate from the Supervisor in proper format? Has it been signed?
N (this will be added in the final report) | |
| d) | Is Abstract included in the Report? Is it properly written? | Y |
| e) | Does the Table of Contents page include chapter page numbers? | Y |
| f) | Does the Report contain a summary of the literature survey? | Y |
| i. | Are the Pages numbered properly? | Y |
| ii. | Are the Figures numbered properly? | Y |
| iii. | Are the Tables numbered properly? | Y |
| iv. | Are the Captions for the Figures and Tables proper? | Y |
| v. | Are the Appendices numbered?
appendices in the midsem report) | N (no |
| g) | Does the Report have Conclusion / Recommendations of the work?
N (this will be added for the final report) | |
| h) | Are References/Bibliography given in the Report? | Y |
| i) | Have the References been cited in the Report? | Y |
| j) | Is the citation of References / Bibliography in proper format? | Y |

Date 18/01/2025



(Digital Signature of Supervisor)

Name of the supervisor: Naveen Rathani

Email Id of Supervisor: naveen.rathani@gmail.com

Mob # of supervisor: +91 80950 00395



Date 18/01/2025

(Digital Signature of Student)