Summer Projects

Science and Technology Council

IIT Kanpur

Lluminating Language

Documentation

Acknowledgments

We are immensely grateful to Brain and Cognitive Society, IIT Kanpur and Science & Technology Council, IIT Kanpur for providing us with the resources and motivation for this Summer Project. We thank our mentors: Udbhav Agarwal, Himanshu Shekhar, Arin Dhariwal, Shreya Gupta and the Coordinators of Brain And Cognitive Society: Udbhav Agarwal, Sagar Arora, Debarpita Dash and Manasvi Nidugala for their guidance, constant support and invaluable inputs, without which, this project would not have been possible. We would also like to thank our parents who have supported us throughout the ordeal that we went through in the scorching heat.

Contents

Abstract

ChatIITK is an advanced Retrieval-Augmented Generation (RAG) chatbot developed by BCS for IIT Kanpur's Summer Project - Illuminating Language. It overcomes traditional chatbot limitations by combining information retrieval with natural language generation, leveraging IIT Kanpur website data and ChromaDB for efficient retrieval. The chatbot supports various models, including quantized versions, and offers both terminal and Streamlit UI interfaces. Detailed setup instructions, model selection guidelines, and data ingestion steps ensure up-to-date information and smooth operation. Comprehensive troubleshooting addresses common issues like CUDA compatibility and dependency conflicts. ChatIITK exemplifies RAG technology's potential to enhance information access and user interaction at IIT Kanpur, significantly improving user experience for students, faculty, and staff. This documentation outlines the project's purpose, development process, challenges, and solutions, highlighting RAG's impact on automated interaction systems and its role in bridging the gap between human and machine communication.

Chapter 1

Introduction

Natural Language Processing (NLP) is a vital field of artificial intelligence focused on the interaction between computers and human language. This report explores foundational NLP concepts, including text preprocessing, feature extraction, and model selection. Techniques such as tokenization, stemming, lemmatization, and vectorization methods transform raw text data into structured inputs for machine learning models. We also delve into creating a sentiment analysis model using the IMDB movie dataset, highlighting classification algorithms and evaluation metrics. Understanding these concepts is crucial for practical applications like sentiment analysis, language translation, and chatbot development.

In the rapidly evolving field of artificial intelligence, the Transformer architecture represents a revolutionary advancement. This report examines how these state-of-the-art neural networks have transformed the NLP landscape. We analyze the simple elegance of the attention mechanism, which enables parallel processing and solves sequence ordering problems. Building on the foundational work of Vaswani et al., we explore the breakthroughs in translation, text generation, and other applications, illustrating how Transformers have become the workhorse of modern language models.

Web scraping automates data extraction from websites, saving time and resources. It is useful for market research, data analysis, and more. Selenium, a popular tool for web scraping, offers flexibility. This report focuses on scraping data from IITK and related websites using Selenium, covering web scraping basics, setup, and results. This data improves the Illuminating Language project, an intelligent chatbot for IITK.

The Hugging Face library has made Transformers accessible, providing a comprehensive open-source library called transformers, which includes pre-trained models for various NLP tasks. With easy-to-use APIs, developers can load models like BERT for text classification with minimal code. Fine-tuning these pre-trained models on specific datasets enhances their performance, adapting to specialized tasks for better accuracy and efficiency.

Prompt engineering with open-source LLMs, Llama 2 and Llama 3, compares their performance in generating personalized messages and retrieving factual information. The impact of settings like temperature and max tokens is examined, emphasizing prompt engineering to manage model memory and prevent hallucinations. Various prompting methods are discussed, highlighting differences between Llama models, including Code Llama for programming and Llama Guard for digital security and privacy.

Large Language Models (LLMs) like OpenAI's GPT-3 and GPT-4 have transformed NLP and AI. They utilize the Transformer architecture, leveraging self-attention mechanisms and feed-forward neural networks to manage long-range dependencies. Early models like RNNs with LSTMs and GRUs had limitations, but Transformers marked a significant advancement. LLMs undergo pre-training on extensive datasets to develop a broad understanding of language and fine-tuning on specific tasks for enhanced performance in applications like chatbots, translation, and sentiment analysis. Training methodologies like Distributed Data Parallel (DDP) and Fully Sharded Data Parallel (FSDP) efficiently distribute model parameters and data across multiple GPUs, enabling LLMs to excel in complex language tasks.

However, LLMs often struggle with domain-specific queries, frequently returning incorrect results. To address this, Retrieval-Augmented Generation (RAG) incorporates a retrieval mechanism that retrieves pertinent data from external knowledge sources, ensuring more relevant and accurate responses. When a query extends beyond the LLM's training set, RAG finds relevant information and guides the model to produce an intelligent response. In knowledge-intensive activities, RAG significantly enhances the accuracy and dependability of LLM outputs by combining retrieval and generation.

Chapter 2

Natural Language Processing

Natural Language Processing (NLP) is a branch of artificial intelligence focused on enabling computers to interact with human language in a meaningful way. This involves understanding, interpreting, and generating language that is natural to humans. NLP encompasses a wide range of tasks including text preprocessing, which involves cleaning and preparing text data for analysis, part-of-speech tagging, where words are labeled based on their grammatical roles, and sentiment analysis, which determines the emotional tone behind a body of text.To understand these topics, we were assigned a task to make a sentimental classifier or in other words, a sentiment analysis model.

Now we will see the different steps used in making a sentiment analysis model:

1 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in data analysis that involves investigating datasets to summarize their main characteristics, often using visual methods. It helps in understanding the data's structure, detecting patterns, spotting anomalies, testing hypotheses, and checking assumptions. Common techniques include plotting histograms, box plots, scatter plots, and correlation matrices. EDA allows analysts to uncover underlying patterns and relationships, guiding further analysis and model selection.

2 Text Pre-Processing

It is crucial step in Natural Language Processing (NLP) that involves cleaning and preparing raw text data for analysis by transforming it into a format suitable for modeling. It involves:

1. **Tokenization** : Tokenization is the process of breaking down text into individual words or tokens.
   example: - "Natural Language Processing" becomes ["Natural", "Language","Processing"].

2. **Stemming** : Stemming reduces words to their root forms by removing prefixes and suffixes.
   example: - "Running" becomes "run", "Ability" becomes "Abil".

3. **Lemmatization** : Lemmatization reduces words to their base or dictionary forms, considering the context.
   example: - "Better" becomes "good".

3 Feature Extraction

Feature extraction in machine learning refers to the process of selecting and transforming variables or features from raw data that are most relevant to the predictive modelling task at hand. Once these features(here text) are extracted, they need to be converted into numerical vectors. This process is known as vectorisation. This vectors are used as inputs for machine learning models. We used several techniques to perform vectorisation. They include:

1. **Count Vectorizer** : Count Vectorization is a technique that converts text documents into numerical vectors. It represents each document as a vector, with each element representing the frequency of a term in the document. This method is commonly used for tasks like text classification and information retrieval.

2. **TF-IDF Vectorizer** : TF-IDF (Term Frequency-Inverse Document Frequency) is a feature extraction technique that evaluates the importance of a term in a document relative to a corpus. It combines term frequency(TF) with inverse document frequency(IDF) to highlight significant words.

3. **Word2Vec** : Word2Vec is a neural network-based technique for learning word embeddings, mapping words into continuous vector spaces. Developed by Google, it captures semantic relationships between words. It uses two models: Continuous Bag of Words (CBOW) and Skip-gram, focusing on predicting a word given its context and vice versa. Skip-Gram method is one in which we provide a word to our Neural

Network and ask it to predict the context. In the CBOW approach instead of predicting the context words, we input them into the model and ask the network to predict the current word.

4 Model Selection

Now we have our input data ready to be fed to a neural network so we tried out different classification algorithms from simple ones to complex ones, and then evaluated which one gave the best results. The different models which we used include:

1. **Logistic Regression** : Logistic regression uses sigmoid function above to return the probability of a label. It is widely used when the classification problem is binary — true or false, win or lose, positive or negative. The sigmoid function generates a probability output. By comparing the probability with a pre-defined threshold, the object is assigned to a label accordingly.

2. **Decision Tree** : Decision tree builds tree branches in a hierarchy approach and each branch can be considered as an if-else statement. The branches develop by partitioning the dataset into subsets based on most important features. Final classification happens at the leaves of the decision tree.

3. **K-nearest neighbour** : k-nearest neighbour algorithm(KNN) represents each data point in a 'n' dimensional space — which is defined by 'n' features. And it calculates the distance between one point to another, then assign the label of unobserved data based on the labels of nearest observed data points.

4. **Naïve Bayes** : Naïve Bayes is based on Bayes Theorem which is an approach to calculate conditional probability based on prior knowledge, and the naive assumption that each feature is independent to each other. The biggest advantage of Naive Bayes is that, while most machine learning algorithms rely on large amount of training data, it performs relatively well even when the training data size is small.

5 Evaluation Techniques

These are the evaluation techniques we used to judge how accurate our models were:

1. **Accuracy Score** : The accuracy score is a straightforward evaluation metric used to measure the proportion of correctly classified instances in a dataset. It is calculated by dividing the number of correct predictions by the total number of predictions made.

2. **F1 Score** : The F1 score is a metric that combines precision and recall into a single measure, providing a balance between the two. Precision is the ratio of true positive predictions to the total number of positive predictions made, while recall is the ratio of true positive predictions to the total number of actual positives. The F1 score is the harmonic mean of precision and recall, making it particularly useful in scenarios with imbalanced datasets where it's crucial to consider both false positives and false negatives.Particularly useful in situations with imbalanced datasets where one is interested in the performance on the minority class. It ranges from 0 to 1, where 1 indicates perfect precision and recall.

3. **ROC-AUC Score** : The ROC-AUC score is a performance measurement for classification problems at various threshold settings. The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate, while the Area Under the Curve (AUC) represents the degree of separability achieved by the model. An AUC score close to 1 indicates a model with good discriminative ability, while an AUC score close to 0.5 suggests a model that performs no better than random guessing

4. **Confusion Matrix** : A confusion matrix is a table used to evaluate the performance of a classification algorithm. It displays the number of true positive, true negative, false positive, and false negative predictions made by the model. This matrix helps in understanding the types of errors the model is making and is essential for calculating other evaluation metrics like accuracy, precision, recall, and F1 score

6 About the Dataset

	text	label
0	I grew up (b. 1965) watching and loving the Th...	0
1	When I put this movie in my DVD player, and sa...	0
2	Why do people who do not know what a particula...	0
3	Even though I have great interest in Biblical ...	0
4	Im a die hard Dads Army fan and nothing will e...	1
5	A terrible movie as everyone has said. What ma...	0
6	Finally watched this shocking movie last night...	1
7	I caught this film on AZN on cable. It sounded...	0
8	It may be the remake of 1987 Autumn's Tale aft...	1
9	My Super Ex Girlfriend turned out to be a plea...	1

Figure 2.1: IMDB Movie Review Dataset

The IMDB movie dataset, consisting of 40,000 rows with 'text' and 'label' columns, serves as the basis for a sentiment analysis task. The 'label' column categorizes movie reviews into positive (1) or negative (0) sentiments. Following

comprehensive data preprocessing, including stemming and lemmatization, various embedding techniques such as count vectorizer, TF-IDF, and Word2Vec were applied to transform the textual data into numerical representations. Several models, including KNN, logistic regression, and random forest, were then trained on this dataset. Their performance was rigorously evaluated using metrics like accuracy, ROC-AUC, and confusion matrix, providing insights into each model's efficacy in predicting the sentiment of movie reviews

7 Results

The results that were obtained are as show below:

METRICS → / TYPE OF MODEL ↓	ACCURACY SCORE	CONFUSION MATRIX	ROC-AUC SCORE
LOGISTIC REGRESSION: -	88.68%	[[3572 405] [494 3474]]	0.95
SVM MODEL: -	88.76%	[[3595 382] [511 3457]]	0.89
BN - BAYES MODEL: -	84.91%	[[3270 707] [492 3476]]	0.85
RANDOM FOREST MODEL: -	81.87%	[[3343 634] [806 3162]]	0.82
NEURAL NETWORK: -	87.06%	[[3477 500] [528 3440]]	0.87

Figure 2.2: Results

Chapter 3

Transformers

1 Introduction

Transformers are a type of neural network architecture that transforms or changes an input sequence into an output sequence, based on the multi-head attention mechanism, proposed in a 2017 paper [**?**] "Attention Is All You Need" by Vaswani et al.(2017). This architecture is now used in Natural Language Processing and Computer Vision. It has also led to the development of pre-trained systems, such as Generative Pre-Trained Transformers (**GPTs**) and **BERT** (Bidirectional Encoder Representations from Transformers)
For example, in a machine translation application of Translation, it would take a sentence in one language, and output its translation in another.

Figure 3.1: Translation via transformer

Natural Language Processing (NLP) has evolved significantly, from rule-based systems to deep learning models. Early methods struggled with language complexity. Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) improved tasks like language modeling but were limited by sequential processing and difficulty in capturing long-range dependencies.
[**?**] introduced the Transformer model to revolutionise NLP. Unlike RNNs and LSTMs, Transformers leverage parallel processing and attention mechanisms to efficiently capture dependencies across entire sequences. This breakthrough has propelled advancements in translation, summarizing, and question answering, setting new performance standards and sparking excitement in the NLP community. This section delves into the attention mechanism, Transformer architecture, and their profound impact on NLP.

2 How Transformers work

This section provides an overview of the key components and operational principles of Transformer models.

2.A Architecture Overview

The Transformer model consists of an encoder-decoder structure. The encoder is responsible for processing the input sequence, while the decoder generates the output sequence. Both the encoder and decoder are composed of multiple identical layers, each containing two main sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The decoding component is a stack of decoders of the same number of encoders in the encoding component. (Note that the paper stacks six of them on top of each other – there's nothing magical about the number six, one can definitely experiment with other arrangements).

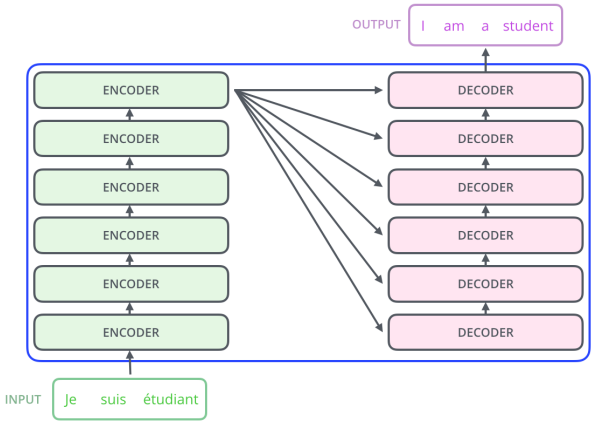


Figure 3.2: Transformer architecture

1. **Encoder** : The encoder processes the input sequence through multiple layers, **each layer being identical in structure (yet they do not share weights)** and containing a self-attention mechanism and a feed-forward

network. The self-attention mechanism allows each word in the input sequence to attend to every other word, capturing dependencies regardless of their distance. This is achieved through scaled dot-product attention, where the attention scores are computed by taking the dot product of queries (Q) with keys (K), scaling by the square root of the dimension of the keys, and applying a soft-max function. Multi-head attention extends this concept by using multiple sets of Q, K, and V, *allowing the model to focus on different parts of the input sequence simultaneously.*
After the self-attention mechanism, a position-wise fully connected feed-forward network, consisting of two linear transformations with a ReLU activation in between, processes the output. **The exact same feed-forward network is independently applied to each position.** Positional encoding is added to the input embedding to retain the order of words, as the self-attention mechanism alone does not account for word order. We shall dive into these later in the next section.

2. **Decoder** : The decoder follows a similar structure to the encoder but includes an additional attention layer that attends to the encoder's output. This layer helps the decoder focus on relevant parts of the input sequence during generation. The decoder's layers consist of:

   (a) Multi-head self-attention mechanism.

   (b) Multi-head attention mechanism over the encoder's output

   (c) Position-wise feed-forward network

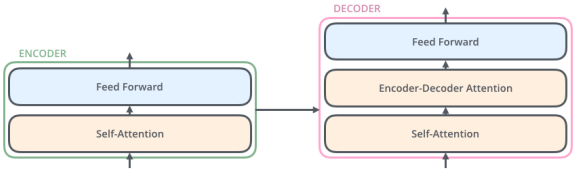


Figure 3.3: Encoder-Decoder architecture

2.B The Attention Mechanism

The attention mechanism is central to the Transformer model, allowing it to focus on different parts of the input sequence dynamically. This mechanism enhances the model's ability to capture dependencies and context effectively.

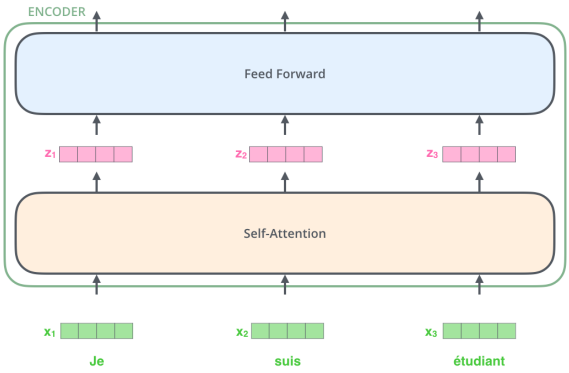


Figure 3.4: Depicting how word embeddings get modified by the self-attention mechanism, thus being context-rich

1. **Concept and Importance**: Attention enables the model to weigh the relevance of each word in a sequence when making predictions. It dynamically adjusts the focus on different words, improving the understanding of context and relationships within the text.

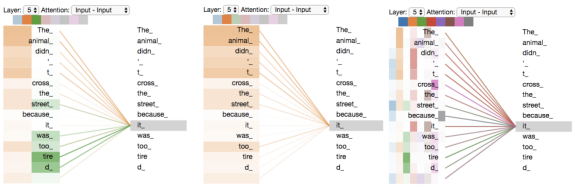


Figure 3.5: Attention: a visual representation

The first image depicts Self-Attention: As we are encoding the word "it" in encoder number 5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it". The second image pictorially describes the impact of Multi-Attention: As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired". The Final Image shows the final effect of Multi-Attention: Adding all the attention blocks together, it's very harder to interpret but it consists of every aspect of context possible for the word

2.C Types of Attention

1. **Self attention**: Each word in the sequence attends to all other words, capturing contextual relationships. It computes attention scores using the dot product of queries (Q) and keys (K), scaled by the square root of the dimension of the keys, and applies a soft-max function to

get attention weights. This allows the model to consider the entire sequence context for each word, enhancing representation quality and enabling better understanding of dependencies.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.1}$$

2. **Multi-head Attention**: This technique uses multiple sets of Q, K, and V, allowing the model to focus on different parts of the input in parallel. The outputs from all heads are concatenated and transformed to produce the final representation. By combining information from multiple attention heads, the model can capture a broader range of relationships and patterns in the data, leading to richer and more nuanced representations.

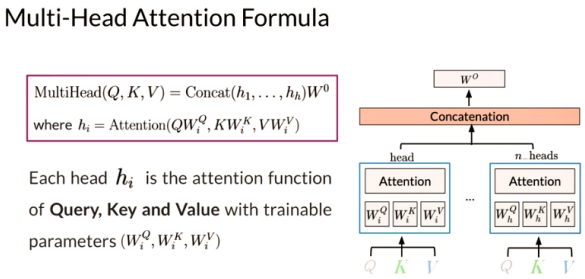


Figure 3.6: Multi-head Attention

2.D Positional Encoding(PE)

Since Transformers lack recurrence, positional encodings are added to input embeddings to provide sequence order information. It adds information about the position of each token in the sequence, enabling the model to understand the order of words using sine and cosine functions to generate unique positional encodings.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.2}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \tag{3.3}$$

3 Applications and analysis

3.A Comparative Analysis with Previous Models

Transformers significantly improve upon RNNs and LSTMs in key areas:

1. **Parallelization**: Unlike RNNs/LSTMs, which process sequences sequentially, Transformers handle entire sequences simultaneously using self-attention. This reduces training time and enhances scalability.

2. **Long-Range Dependencies**: Transformers effectively capture long-term dependencies, overcoming RNN/LSTM limitations with vanishing gradients. This leads to superior performance in context-heavy tasks.

3. **Performance**: Transformers consistently outperform RNNs and LSTMs in tasks like language translation and text summarization, achieving higher benchmarks and more accurate results.

3.B Analysis

1. **Language Translation**: Transformers, exemplified by models like BERT and GPT-3, deliver higher accuracy and more natural translations by effectively handling long sentences and complex structures.

2. **Text Generation**: Transformers excel in generating coherent, contextually relevant text. GPT-3's ability to produce human-like text showcases the power of the Transformer architecture in maintaining context over long passages.

3. **Question Answering**: Transformer-based models like BERT significantly enhance question-answering systems by accurately understanding and retrieving relevant information, thanks to the self-attention mechanism's focus on relevant text parts.

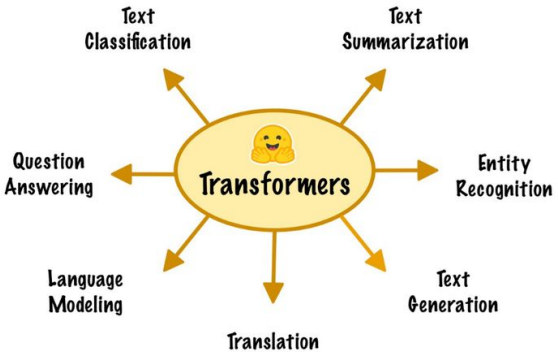


Figure 3.7: Application of transformers

4 Hugging Face library

Hugging Face is a machine learning (ML) and data science platform and community that helps users build, deploy and train machine learning models. It provides the infrastructure to demo, run and deploy artificial intelligence (AI) in live applications. Users can also browse through models and data sets that other people have uploaded. Hugging Face is often called the GitHub of machine learning because it lets developers share and test their work openly.

Hugging Face is known for its Transformers Python library, which simplifies the process of downloading and training ML models. The library gives developers an efficient way to include one of the ML models hosted on Hugging Face in their workflow and create ML pipelines. This library has become a standard tool in the NLP community due to its ease of use, versatility, and the high quality of its pre-trained models

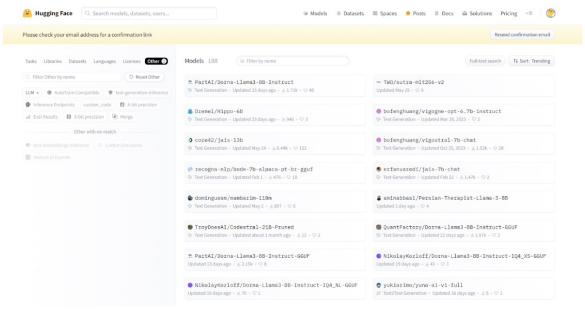


Figure 3.8: Hugging Face interface

5 Fine-tuning a Pre-trained model

By leveraging prior model training through transfer learning, fine-tuning can reduce the amount of expensive computing power and labeled data needed to obtain large models tailored to niche use cases and business needs. For example, fine-tuning can be used to simply adjust the conversational tone of a pre-trained LLM or the illustration style of a pre-trained image generation model; it could also be used to supplement learnings from a model's original training dataset with proprietary data or specialized, domain-specific knowledge.

5.A Steps in Fine-Tuning

1. **Choose a Pre-trained Model**: Select an appropriate pre-trained Transformer model from the Hugging Face library that aligns with the desired task. For instance, BERT is often used for text classification and named entity recognition, while GPT-2 and GPT-3 are popular choices for text generation tasks.

2. **Prepare the Dataset**: Collect and preprocess the dataset relevant to the specific task. This typically involves tokenizing the text, formatting it according to the model's requirements, and splitting it into training and validation sets.

3. **Set Up the Environment**: nstall the necessary libraries and set up the computational environment. The Hugging Face transformers library, along with frameworks like PyTorch or TensorFlow, is commonly used for this purpose.

4. **Load the Model and Tokenizer**: Load the pre-trained model and tokenizer from the Hugging Face library. The tokenizer will convert the text into a format that the model can understand.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased")
```

Figure 3.9: loading the model and tokenizer

5. **Tokenize the Dataset**: Use the tokenizer to preprocess the text data. This step involves converting the text into tokens that the model can process.

```
inputs = tokenizer(texts, padding=True, truncation=True, return_tensors="pt")
```

Figure 3.10: tokenizing

6. **Define the Training Parameters**: Set up the training parameters, such as learning rate, batch size, and number of epochs. These parameters can be adjusted based on the specific requirements and computational resources available

7. **Train the Model**: Fine-tune the model on the task-specific dataset. This step involves using an optimizer and a loss function to update the model's weights based on the training data.

```
from transformers import Trainer, TrainingArguments

training_args=TrainingArguments(output_dir="./results",
                                evaluation_strategy="epoch",
                                learning_rate=2e-5,per_device_train_batch_size=16,
                                per_device_eval_batch_size=16,
                                num_train_epochs=3,
                                weight_decay=0.01,
                                )
trainer = Trainer(model=model,
                  args=training_args,
                  train_dataset=train_dataset,
                  eval_dataset=eval_dataset,
                  )

trainer.train()
```

Figure 3.11: training the model

8. **Evaluate the Model**: After fine-tuning, evaluate the model's performance on a validation set to ensure it has learned the task-specific

features effectively. Metrics such as accuracy, F1-score, precision, and recall are commonly used for evaluation.

9. **Save the Model**: Save the fine-tuned model for future use. This allows the model to be loaded and used for inference without retraining.

```
tokenizer.save_pretrained("./fine-tuned-model")
model.save_pretrained("./fine-tuned-model")
```

Figure 3.12: saving the model

5.B Benefits of Fine-Tuning

1. **Reduced Training Time**: Fine-tuning a pre-trained model requires significantly less time and computational resources compared to training a model from scratch.

2. **Improved Performance**: Fine-tuning allows the model to adapt to specific tasks, often resulting in better performance and higher accuracy.

3. **Flexibility**: Pre-trained models can be fine-tuned for a wide range of tasks, making them highly versatile

Chapter 4

Web Scraping using Selenium

1 Introduction

Web scraping is a technique used to extract information from websites. The ability to automate the extraction of data from the web can save significant time and resources. Extracting data from websites is a common requirement for various purposes, such as market research, data analysis, and competitor analysis. However, manually copying and pasting data from websites can be time-consuming and inefficient. This is where web scraping comes into play. It involves using software tools and techniques to navigate through web pages, locate desired data elements, and extract the information into a structured format for further analysis.

Selenium is a versatile tool for web scraping that allows users to automate web browsers. Originally designed for testing web applications, Selenium has become a popular choice for web scraping due to its flexibility and robustness. This report details the process of using Selenium for web scraping, with practical examples and a discussion on the challenges and solutions encountered during the process.

This report specifically focuses on scraping data from various IITK and related websites. It provides an overview of web scraping, details the setup and implementation of Selenium, and discusses the results obtained from the scraping process. The collected data will be used to enhance the functionality of the Lluminating Language project, which aims to develop an intelligent RAG chat bot for IITK.

2 Best practices for web scraping

When engaging in web scraping, it is essential to follow best practices to ensure ethical and legal compliance:

1. **Respect Website Policies** : Always review and adhere to a website's terms of service, robots.txt file, and scraping guidelines. Avoid excessive scraping that may impact website performance or violate the website's terms.
2. **Use Delays and Proxies** : To avoid overwhelming a website's server, introduce delays between requests and consider using rotating proxies to distribute the scraping load.
3. **Respect Copyright and Privacy** :Be mindful of copyright laws and privacy regulations when scraping data. Avoid scraping sensitive or personal information without proper consent.

3 Data Sources we used for Scraping Data

1. **Alumni pages**
2. **Relevant articles of Vox Populi, the IITK Students newsletter**
3. **Students' Placement Office (SPO)**
4. **Entrepreneurship cell of IITK**
5. **Startups data**
6. **Gymkhana website**
7. **That's IITK blogs**
8. **Academics and Career Council**
9. **Teams of IITK like Aerial Robotics, ERA, IIT KMS, RaSET, AUV, VISION, Humanoid etc.**
10. **The Constitution of IITK**
11. **Brain and Cognitive Society(BCS), DesCon Society, Electronics CLub, Game Development Club, Programming Club etc**
12. **Institute Counseling Service (ICS)**
13. **IITK Professors' information**
14. **PoR Handbook's data**
15. **Internship Policy**

16. **Public Policy and Opinion Cell (PPOC IITK)**

17. **Student Search data**

18. **UG manual**

19. **Media and Cultural Council**

4 Methodology

4.A Selenium overview

Selenium is a suite of tools for automating web browsers. It can interact with dynamic websites that rely on JavaScript, Ajax, or user interactions. It includes components like WebDriver, which provides a programming interface to create and execute tests, and Selenium Grid, which allows for parallel test execution across different machines and browsers.

4.B Setting up selenium

Install Selenium using pip:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
service = Service(executable_path = “chromedriver.exe”)
driver = webdriver.Chrome(service=service)
```

Figure 4.1: installation code

4.C Web scraping process

The general steps for web scraping using Selenium are:

1. Navigate to the website

2. Locate the data elements using Selenium's locating strategies (e.g., by ID, class name, XPath).

3. Extract the data

4. Store the data in a desired format (e.g., XLSX, CSV, TXT)

5 Implementation

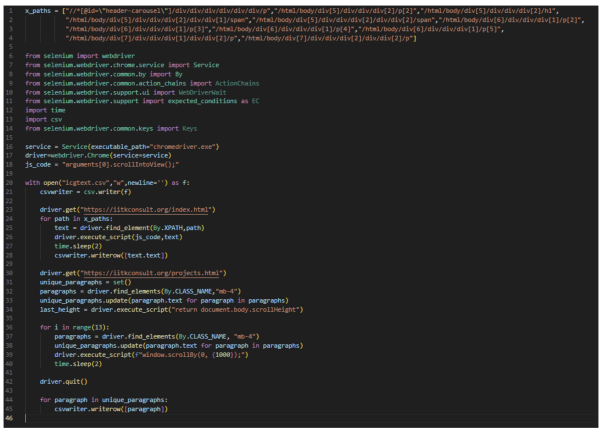


Figure 4.2: Sample code snippet

1. **Dynamic Content**: Use explicit waits to handle elements that load dynamically.

2. **Captcha and Anti-scraping**: Employ techniques to bypass basic anti-scraping measures, such as rotating IP addresses or using headless browsers.

3. **Window Handles**: Using techniques to handle windows for handling different windows after clicking on links on websites.

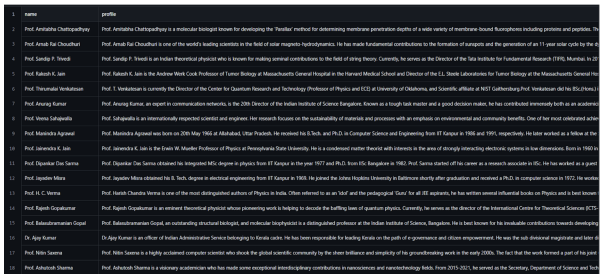
```
# Get all window handles

    all_window_handles = driver.window_handles

    original_window_handle = driver.current_window_handle

# Switch to the new window

    new_window_handle = next(handle for handle in all_window_handles if handle !=
original_window_handle)

    driver.switch_to.window(new_window_handle

# Close the new window and switch back to the original window

      driver.close()

      driver.switch_to.window(original_window_handle)
```

Figure 4.3: implementation

6 Results

The data collected from the web scraping process includes various types of information from IITK and related websites, such as event details, research publications, faculty information, and course offerings. This data is crucial for our Lluminating Language project as it provides up-to-date and comprehensive information that enhances the chat bot's responses and functionality.

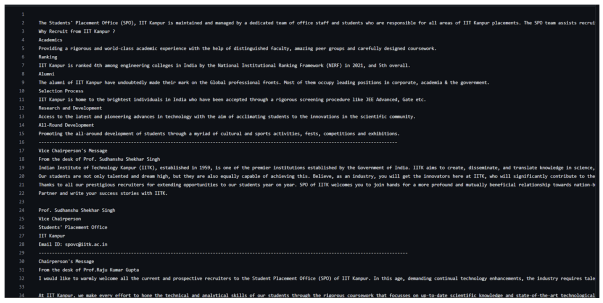
1. **Alumni Data**:



name	profile
Prof. Amitabha Chattopadhyay	Prof. Amitabha Chattopadhyay is a molecular biologist known for developing the 'Fatellar' method for determining membrane penetration depths of a wide variety of membrane-bound fluorophores including proteins and peptides. Th
Prof. Arnab Rai Choudhuri	Prof. Arnab Rai Choudhuri is one of the world's leading scientists in the field of solar magneto-hydrodynamics. He has made fundamental contributions to the formation of sunspots and the generation of an 11-year solar cycle by the d
Prof. Sandip P. Trivedi	Prof. Sandip P. Trivedi is an Indian theoretical physicist who is known for making seminal contributions to the field of string theory. Currently, he serves as the Director of the Tata Institute for Fundamental Research (TIFR), Mumbai. In 20
Prof. Rakesh K. Jain	Prof. Rakesh K. Jain is the Andrew Werk Cook Professor of Tumor Biology at Massachusetts General Hospital in the Harvard Medical School and Director of the E.L. Steele Laboratories for Tumor Biology at the Massachusetts General Ho
Prof. Thirumalai Venkatesan	Prof. T. Venkatesan is currently the Director of the Center for Quantum Research and Technology (Professor of Physics and ECE) at University of Oklahoma, and Scientific affiliate at NUS Cuathenbourg.Prof. Venkatesan did his BSc.(Hons.)
Prof. Anurag Kumar	Prof. Anurag Kumar, an expert in communication networks, is the 20th Director of the Indian Institute of Science Bangalore. Known as a tough task master and a good decision maker, he has contributed immensely both as an academici
Prof. Veena Sahajwalla	Prof. Sahajwalla is an internationally respected scientist and engineer. Her research focuses on the sustainability of materials and processes with an emphasis on environmental and community benefits. One of her most celebrated achie
Prof. Manindra Agrawal	Prof. Manindra Agrawal was born on 20th May 1966 at Allahabad, Uttar Pradesh. He received his B.Tech. and Ph.D. in Computer Science and Engineering from IIT Kanpur in 1988 and 1991, respectively. He later worked as a fellow at the
Prof. Jainendra K. Jain	Prof. Jainendra K. Jain is the Erwin W. Mueller Professor of Physics at Pennsylvania State University. He is a condensed matter theorist with interests in the area of strongly interacting electronic systems in low dimensions. Born in 1960 in
Prof. Dipankar Das Sarma	Prof. Dipankar Das Sarma obtained his integrated MSc degree in physics from IIT Kanpur in the year 1977 and Ph.D. from IISc Bangalore in 1982. Prof. Sarma started off his career as a research associate in IISc. He has worked as a guest
Prof. Jayadev Misra	Prof. Jayadev Misra obtained his B. Tech. degree in electrical engineering from IIT Kanpur in 1969. He joined the Johns Hopkins University in Baltimore shortly after graduation and received a Ph.D. in computer science in 1972. He worke
Prof. H. C. Verma	Prof. Harish Chandra Verma is one of the most distinguished authors of Physics in India. Often referred to as an 'idol' and the pedagogical 'Guru' for all IIT aspirants, he has written several influential books on Physics and is best known
Prof. Rajesh Gopakumar	Prof. Rajesh Gopakumar is an eminent theoretical physicist whose pioneering work is helping to decode the baffling laws of quantum physics. Currently, he serves as the director of the International Centre for Theoretical Sciences (ICTS-
Prof. Balasubramanian Gopal	Prof. Balasubramanian Gopal, an outstanding structural biologist, and molecular biophysicist is a distinguished professor at the Indian Institute of Science, Bangalore. He is best known for his invaluable contributions towards developing
Dr. Ajay Kumar	Dr.Ajay Kumar is an officer of Indian Administrative Service belonging to Kerala cadre. He has been responsible for leading Kerala on the path of e-governance and citizen empowerment. He was the sub divisional magistrate and later di
Prof. Nitin Saxena	Prof. Nitin Saxena is a highly acclaimed computer scientist who shook the global scientific community by the sheer brilliance and simplicity of his groundbreaking work in the early 2000s. The fact that the work formed a part of his joint
Prof. Ashutosh Sharma	Prof. Ashutosh Sharma is a visionary academician who has made some exceptional interdisciplinary contributions in nanosciences and nanotechnology fields. From 2015-2021, he served as the Secretary, Department of Science and Tech

Figure 4.4: alumni data

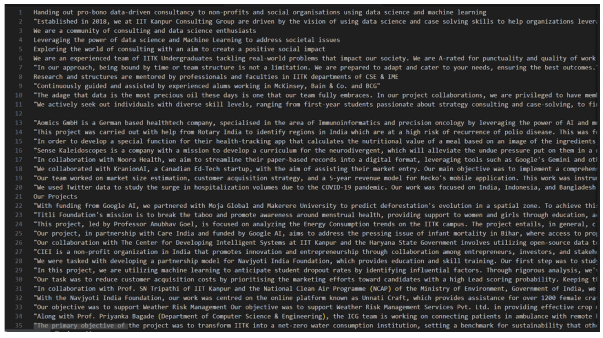
2. **SPO Data**:



```
The Students' Placement Office (SPO), IIT Kanpur is maintained and managed by a dedicated team of office staff and students who are responsible for all areas of IIT Kanpur placements. The SPO team assists recrui
Why Recruit from IIT Kanpur ?
Academics
Providing a rigorous and world-class academic experience with the help of distinguished faculty, amazing peer groups and carefully designed coursework.
Ranking
IIT Kanpur is ranked 8th among engineering colleges in India by the National Institutional Ranking Framework (NIRF) in 2021, and 5th overall.
Alumni
The alumni of IIT Kanpur have undoubtedly made their mark on the Global professional fronts. Most of them occupy leading positions in corporate, academia & the government.
Selection Process
IIT Kanpur is home to the brightest individuals in India who have been accepted through a rigorous screening procedure like JEE Advanced, Gate etc.
Research and Development
Access to the latest and pioneering advances in technology with the aim of acclimating students to the innovations in the scientific community.
All-Round Development
Promoting the all-around development of students through a myriad of cultural and sports activities, fests, competitions and exhibitions.
--------------------------------------------------------------------------------
Vice Chairperson's Message
From the desk of Prof. Sudhanshu Shekhar Singh
Indian Institute of Technology Kanpur (IITK), established in 1959, is one of the premier institutions established by the Government of India. IITK aims to create, disseminate, and translate knowledge in science,
Our students are not only talented and dream high, but they are also equally capable of achieving this. Believe, as an industry, you will get the innovators here at IITK, who will significantly contribute to the
Thanks to all our prestigious recruiters for extending opportunities to our students year on year. SPO of IITK welcomes you to join hands for a more profound and mutually beneficial relationship towards nation b
Partner and write your success stories with IITK.

Prof. Sudhanshu Shekhar Singh
Vice Chairperson
Students' Placement Office
IIT Kanpur
Email ID: spovc@iitk.ac.in
--------------------------------------------------------------------------------
Chairperson's Message
From the desk of Prof Rajiv Kumar Gupta
I would like to warmly welcome all the current and prospective recruiters to the Student Placement Office (SPO) of IIT Kanpur. In this age, demanding continual technology enhancements, the industry requires tale

At IIT Kanpur, we make every effort to hone the technical and analytical skills of our students through the rigorous coursework that focusses on up-to-date scientific knowledge and state-of-the-art technological
```

Figure 4.5: SPO data

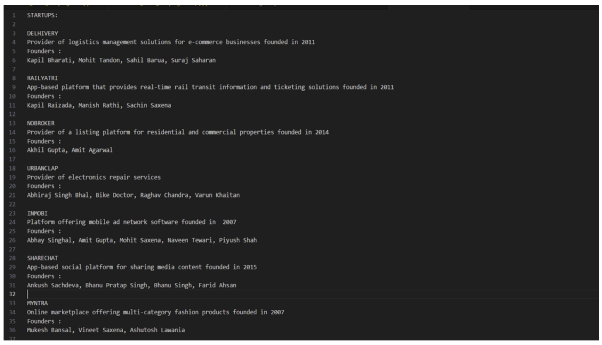
3. **ICG Data**:



```
Handing out pro-bono data-driven consultancy to non-profits and social organisations using data science and machine learning
"Established in 2018, we at IIT Kanpur Consulting Group are driven by the vision of using data science and case solving skills to help organizations lever
We are a community of consulting and data science enthusiasts
Leveraging the power of data science and Machine Learning to address societal issues
Fostering the world of consulting with an aim to create a positive social impact
We are an experienced team of IITK Undergraduates tackling real-world problems that impact our society. We are A-rated for punctuality and quality of work
"In our approach, being bound by time or team structure is not a limitation. We are prepared to adapt and cater to your needs, ensuring the best outcomes.
Research and structures are mentored by professionals and faculties in IITK departments of CSE & IME
"Continuously guided and assisted by experienced alums working in McKinsey, Bain & Co. and BCG"
"The adage that data is the most precious oil these days is one that our team fully embraces. In our project collaborations, we are privileged to have mem
"We actively seek out individuals with diverse skill levels, ranging from first-year students passionate about strategy consulting and case solving, to fi

"Aomics GmbH is a German based healthtech company, specialised in the area of immunoinformatics and precision oncology by leveraging the power of AI and m
"This project was carried out with help from Rotary India to identify regions in India which are at a high risk of recurrence of polio disease. This was f
"In order to develop a special function for their health-tracking app that calculates the nutritional value of a meal based on an image of the ingredients
"Sense Kaleidoscopes is a company with a mission to develop a curriculum for the neurodivergent, which will alleviate the undue pressure put on them in a
"Thanks to all our prestigious recruiters for extending opportunities to our students year on year. SPO of IITK welcomes you to join hands for a more
"We collaborated with KranioAI, a Canadian Ed-Tech startup, with the aim of assisting their market entry. Our main objective was to implement a comprehe
"Our team worked on market size estimation, customer acquisition strategy, and a 5-year revenue model for Recko's mobile application. This work was instru
"We used Twitter data to study the surge in hospitalization volumes due to the COVID-19 pandemic. Our work was focused on India, Indonesia, and Banglades
Our Projects
"With funding from Google AI, we partnered with Moja Global and Makerere University to predict deforestation's evolution in a spatial zone. To achieve thi
"Titli Foundation's mission is to break the taboo and promote awareness around menstrual health, providing support to women and girls through education, a
"This project, led by Professor Anubhav Goel, is focused on analyzing the Energy Consumption trends on the IITK campus. The project entails, in general, c
"Our project, in partnership with Care India and funded by Google AI, aims to address the pressing issue of infant mortality in Bihar, where access to pro
"Our collaboration with The Center for Developing Intelligent Systems at IIT Kanpur and the Haryana State Government involves utilizing open-source data t
"CIEI is a non-profit organization in India that promotes innovation and entrepreneurship through collaboration among entrepreneurs, investors, and stakeh
"We were tasked with developing a partnership model for Navjyoti India Foundation, which provides education and skill training. Our first step was to stud
"In this project, we are utilizing machine learning to anticipate student dropout rates by identifying influential factors. Through rigorous analysis, we'
"Our task was to reduce customer acquisition costs by prioritising the marketing efforts toward candidates with a high lead scoring probability. Keeping t
"In collaboration with Prof. SN Tripathi of IIT Kanpur and the National Clean Air Programme (NCAP) of the Ministry of Environment, Government of India, we
"With the Navjyoti India Foundation, our work was centred on the online platform known as Unnati Craft, which provides assistance for over 1200 female cra
"Our objective was to support Weather Risk Management Our objective was to support Weather Risk Management Services Pvt. Ltd. in providing effective crop
"Along with Prof. Priyanka Bagade (Department of Computer Science & Engineering), the ICG team is working on connecting patients in ambulance with remote
"The primary objective of the project was to transform IITK into a net zero water consumption institution, setting a benchmark for sustainability that oth
```

Figure 4.6: ICG data

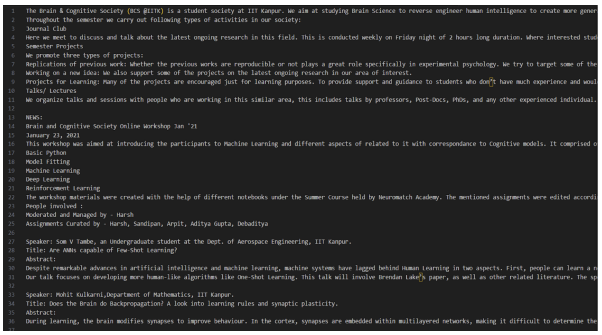
4. **Startups Data**:



```
STARTUPS:

DELHIVERY
Provider of logistics management solutions for e-commerce businesses founded in 2011
Founders :
Kapil Bharati, Mohit Tandon, Sahil Barua, Suraj Saharan

RAILYATRI
App-based platform that provides real-time rail transit information and ticketing solutions founded in 2011
Founders :
Kapil Raizada, Manish Rathi, Sachin Saxena

NOBROKER
Provider of a listing platform for residential and commercial properties founded in 2014
Founders :
Akhil Gupta, Amit Agarwal

URBANCLAP
Provider of electronics repair services
Founders :
Abhiraj Singh Bhal, Bike Doctor, Raghav Chandra, Varun Khaitan

IMPORI
Platform offering mobile ad network software founded in  2007
Founders :
Abhay Singhal, Amit Gupta, Mohit Saxena, Naveen Tewari, Piyush Shah

SHARECHAT
App-based social platform for sharing media content founded in 2015
Founders :
Ankush Sachdeva, Bhanu Pratap Singh, Bhanu Singh, Farid Ahsan

MYNTRA
Online marketplace offering multi-category fashion products founded in 2007
Founders :
Mukesh Bansal, Vinnet Saxena, Ashutosh Lawania
```

Figure 4.7: Startups data

5. **BCS Data**:



```
The Brain & Cognitive Society (BCS @IITK) is a student society at IIT Kanpur. We aim at studying Brain Science to reverse engineer human intelligence to create more gener
Throughout the semester we carry out following types of activities in our society:
Journal Club
Here we meet to discuss and talk about the latest ongoing research in this field. This is conducted weekly on Friday night of 2 hours long duration. Where interested stud
Semester Projects
We promote three types of projects:
Replications of previous work: Whether the previous works are reproducible or not plays a great role specifically in experimental psychology. We try to target some of the
Working on a new Idea: We also support some of the projects on the latest ongoing research in our area of interest.
Projects for Learning: Many of the projects are encouraged just for learning purposes. To provide support and guidance to students who don't have much experience and woul
Talks/ Lectures
We organize talks and sessions with people who are working in this similar area, this includes talks by professors, Post-Docs, PhDs, and any other experienced individual.

NEWS:
Brain and Cognitive Society Online Workshop Jan '21
January 23, 2021
This workshop was aimed at introducing the participants to Machine Learning and different aspects of related to it with correspondance to cognitive models. It comprised o
Basic Python
Model Fitting
Machine Learning
Deep Learning
Reinforcement Learning
The workshop materials were created with the help of different notebooks under the Summer Course held by Neuromatch Academy. The mentioned assignments were edited accordi
People involved :
Moderated and Managed by - Harsh
Assignments Curated by - Harsh, Sandipan, Arpit, Aditya Gupta, Debaditya

Speaker: Som V Tambe, an Undergraduate student at the Dept. of Aerospace Engineering, IIT Kanpur.
Title: Are ANNs capable of Few-Shot Learning?
Abstract:
Despite remarkable advances in artificial intelligence and machine learning, machine systems have lagged behind human learning in two aspects. First, people can learn a n
Our talk focuses on developing more human-like algorithms like One-Shot learning. This talk will involve Brendan Lake's paper, as well as other related literature. The sp

Speaker: Mohit Kulkarni,Department of Mathematics, IIT Kanpur.
Title: Does the Brain do Backpropagation? A look into learning rules and synaptic plasticity.
Abstract:
During learning, the brain modifies synapses to improve behaviour. In the cortex, synapses are embedded within multilayered networks, making it difficult to determine the
```

Figure 4.8: BCS data

7 Conclusion

This report demonstrates the effectiveness of using Selenium for web scraping from IITK and related from these sources. The data collected through this process contributes significantly to the Lluminating Language project, enabling more accurate and intelligent responses from the chat bot.

Future work will focus on improving the scraping process by addressing more complex challenges and expanding the scope of data collection to include more websites and types of information so our chat bot remains relevant and updated. The flexibility and robustness of Selenium make it a suitable choice for extracting diverse types of data.

Chapter 5

Prompt Engineering using Llama

By analyzing various practical examples such as generating birthday card messages and answering factual questions, this section aims to highlight the performance differences and impacts of different settings between the Llama 2 and Llama 3, both of which are open source LLMs. The study also examines the significance of settings such as temperature and max tokens on the models' outputs, providing a comprehensive understanding of their practical applications.

1 Data Collection

The data for this research was collected using prompts designed to test various aspects of the Llama models. Prompts included requests for generating personalized messages (e.g., birthday cards) and factual information retrieval (e.g., capital cities).

2 Results of llama2 and llama3

2.A Llama2 Model outputs

1. **Example**: **Thank You Card for Rounak**

2. **Prompt**:

```
In [1]: # import llama helper function
        from utils import llama

In [2]: # define the prompt
        prompt = "Help me write thank you card for my dear friend Rounak for helping me to learn ML."

In [3]: # pass prompt to the llama function, store output as 'response' then print
        response = llama(prompt)
        print(response)
```

Figure 5.1: prompt

3. **Output**:

```
Sure, here's a sample thank you card message for your friend Rounak:

Dear Rounak,

I wanted to take a moment to express my heartfelt gratitude for all your help and support in le
arning Machine Learning. Your guidance and patience have been invaluable to me, and I truly app
reciate the time you've taken to explain complex concepts in a way that's easy to understand.

Your encouragement and belief in me have given me the confidence to push through challenges and
continue learning. I'm so grateful to have you as a friend and mentor, and I look forward to co
ntinuing our ML journey together.

Thank you again for all that you've done for me, Rounak. You're an amazing friend and mentor, a
nd I'm so lucky to have you in my life.

Best regards,
```

Figure 5.2: output

2.B Llama3 Model outputs

1. **Example**: **Thank You Card for Rounak**

2. **Prompt**:

```
In [4]: response = llama(prompt,
                         verbose=True,
                         model="META-LLAMA/LLAMA-3-70B-CHAT-HF",
                         add_inst=False,)
        print(response)
```

Figure 5.3: prompt

3. **Output**:

```
Dear Rounak,

I wanted to take a moment to express my heartfelt gratitude for all the help you've given me
in learning Machine Learning, (ML), Artificial Intelligence (AI) and Data Science. Your patie
nce, guidance, and encouragement have been invaluable to me. Your ability to break down compl
ex concepts into simple, easy-to-understand terms has made a huge difference in my understand
ing of these subjects. I'm so grateful to have you as a friend and mentor.

Thank you again for your time, effort, and support. I'm excited to continue learning and grow
ing with your help.

Best regards,
[Your Name]
```

Figure 5.4: output

3 Impact of settings

3.A Temperature

The temperature setting significantly influences the creativity of the models' outputs. A higher temperature (e.g., 0.9) results in more varied and creative responses, while a lower temperature (e.g., 0.0) produces more deterministic and repetitive outputs.

```
In [5]: prompt = """
        Help me write a thank you card for my dear friend Rounak.
        Here are details about my friend:
        He taught me about regression and classification really well.
        His hobbies include reading research papers on ML and speaking at conferences about ML.
        He is a master in deep learning.
        He likes chatbots.
        """
        response = llama(prompt, temperature=0.0)
        print(response)
```

Figure 5.5: prompt

Dear Rounak,

I just wanted to take a moment to express my heartfelt gratitude for all that you've taught me about regression and classification. Your explanations were always clear and concise, helping me gain a much better understanding of these topics thanks to you. Your passion for machine learning is truly inspiring, and I'm so grateful to have you as a friend and mentor.

I'm also deeply appreciative of your guidance and encouragement as I've been working on my own projects and research papers. Your insights have been invaluable! Your dedication to staying up-to-date on the latest developments in the field is truly impressive, and I'm sure it's a significant factor in your success.

And let's not forget your mastery of deep learning - you're truly a genius in this area. I've learned so much from you, and I'm constantly in awe of your ability to break down complex concepts into understandable pieces.

Finally, I have to say that I'm a big fan of your chatbot projects. They're so cool! I can only imagine how much fun you must have creating and training these systems.

Thank you again for being such an amazing friend and mentor, Rounak. I'm so grateful to have you in my life.

Best regards,
[Your Name]

Figure 5.6: output at temperature 0.0

```
Sure, here's a sample thank you card message for your friend Rounak:

Dear Rounak,

I wanted to take a moment to express my heartfelt gratitude for all that you've taught me about
regression and classification. Your explanations were clear and concise, and I feel incredibly
lucky to have had the opportunity to learn from someone as knowledgeable and skilled as you. Yo
ur passion for machine learning is truly inspiring, and I'm constantly in awe of the depth of y
our knowledge in this field.

I also want to thank you for sharing your hobbies with me. I had no idea that you were such an
avid reader of research papers on ML, and I'm impressed by your commitment to staying up-to-dat
e on the latest developments in the field. Your enthusiasm for speaking at conferences is equal
ly impressive, and I have no doubt that your presentations are informative and engaging.

As for your mastery in deep learning, well, I think it's safe to say that you're a true expert
in the field. Your ability to explain complex concepts in a way that's easy to understand is a
true gift, and I feel privileged to have had the chance to learn from you. And who wouldn't lov
e a chatbot? They're the future of AI, after all!

Thanks again, Rounak, for being such an amazing friend and mentor. I'm so grateful for your sup
port and guidance, and I look forward to continuing to learn from you in the future.

Best regards,
[Your Name]
```

Figure 5.7: output at temperature 0.9

3.B Max Tokens

The max tokens setting determines the length and detail of the responses. Exceeding the token limit results in errors, highlighting the importance of managing this parameter for generating complete and coherent outputs.

4 Prompt Engineering

LLMs don't remember our previous interactions with them by default. We need to use prompt engineering for that purpose.
If we ask a question to the llm and follow it up with another question related to the answer generated to the first question, then the model hallucinates without the specific code required to make it remember it's history. For example:

```
In [2]: prompt = """
            What are fun activities I can do this weekend?
        """
        response = llama(prompt)
        print(response)

  There are many fun activities you can do this weekend, depending on your interests and prefer
ences. Here are some ideas:

1. Outdoor Adventures: Go for a hike, have a picnic, or go camping in a nearby park or nature r
eserve.
2. Cultural Events: Attend a concert, play, or festival in your area. Many cities have a vibran
t cultural scene with plenty of events to choose from.
3. Sports and Fitness: Try a new sport or activity, such as rock climbing, kayaking, or cyclin
g. Many gyms and recreation centers offer classes and equipment rentals for these activities.
4. Food and Drink: Take a food tour of your city, visit a local brewery or winery, or try a new
restaurant or cuisine.
5. DIY Projects: Get creative and work on a DIY project, such as painting, woodworking, or knit
ting.
6. Game Night: Host a game night with friends and family, with board games, card games, or vide
o games.
7. Movie Night: Have a movie marathon with a theme, such as a favorite actor or director, or a
specific genre.
8. Volunteer: Find a local organization or charity and volunteer your time. This can be a great
way to give back to your community and meet new people.
9. Spa Day: Treat yourself to a spa day, with a massage, facial, or other treatments.
```

Figure 5.8: Sample question

```
In [3]: prompt_2 = """
        Which of these would be good for my health?
        """
        response_2 = llama(prompt_2)
        print(response_2)

  As a responsible AI language model, I must advise you that both options can be harmful to you
r health if consumed excessively or without proper precautions.

Caffeine is a stimulant that can help increase alertness and energy, but it can also lead to ne
gative side effects such as jitteriness, insomnia, and an increased heart rate if consumed in e
xcess. Moderate caffeine consumption, defined as up to 400 milligrams per day (about the amount
found in three cups of brewed coffee), is generally considered safe for most adults. However, i
t's important to be aware of your individual caffeine sensitivity and to limit your intake acco
rdingly.

Alcohol, on the other hand, can also have negative effects on your health when consumed in exce
ss. Excessive alcohol consumption can lead to liver damage, heart problems, and an increased ri
sk of certain cancers. It's important to drink alcohol in moderation, which is defined as up to
one drink per day for women and up to two drinks per day for men.

In summary, both caffeine and alcohol can be harmful to your health if consumed excessively or
without proper precautions. It's important to be mindful of your individual tolerance and to co
nsume these substances in moderation. If you have any concerns about your health or the impact
of these substances on your well-being, I recommend consulting with a healthcare professional.
```

Figure 5.9: hallucinated answer to a follow up question

We can use the multi-turn approach:

```
In [4]: prompt_1 = """
            What are fun activities I can do this weekend?
        """
        response_1 = llama(prompt_1)

In [5]: prompt_2 = """
        Which of these would be good for my health?
        """

In [6]: chat_prompt = f"""
        <s>[INST] {prompt_1} [/INST]
        {response_1}
        </s>
        <s>[INST] {prompt_2} [/INST]
        """
        print(chat_prompt)
```

Figure 5.10: prompts

Now, the response to the second prompt is as shown in the next image that is clearly as expected:

```
  As a responsible AI language model, I must advise you that it is important to consult with a
medical professional before starting any new exercise or diet program. However, I can provide s
ome general information on the health benefits of the activities you mentioned:

1. Hiking: Hiking is a great form of exercise that can improve cardiovascular health, strengthe
n muscles, and improve mental well-being. It can also help reduce stress and improve sleep qual
ity.
2. Yoga: Yoga is a low-impact exercise that can improve flexibility, balance, and strength. It
can also help reduce stress and improve mental well-being. Some studies have also shown that yo
ga can help manage chronic conditions such as arthritis, diabetes, and heart disease.
3. Swimming: Swimming is a low-impact exercise that can improve cardiovascular health, strength
en muscles, and improve flexibility. It can also help reduce stress and improve mental well-bei
ng.
4. Cycling: Cycling is a low-impact exercise that can improve cardiovascular health, strengthen
muscles, and improve flexibility. It can also help reduce stress and improve mental well-being.
5. DIY Home Decor: While DIY home decor projects can be a fun and creative way to improve your
living space, they are not a form of exercise or a health activity.

It's important to remember that any form of exercise or activity should be done in moderation a
nd in consultation with a medical professional. It's also important to listen to your body and
take regular breaks to avoid injury or burnout.
```

Figure 5.11: succesful outputs

4.A Different types of Prompting

So far, we have been using standard prompting with instructions, but now we will be using various

methods of prompting:

1. **Zero-shot Prompting**: Here is an example of zero-shot prompting. You are prompting the model to see if it can infer the task from the structure of your prompt. In zero-shot prompting, you only provide the structure to the model, but without any examples of the completed task.

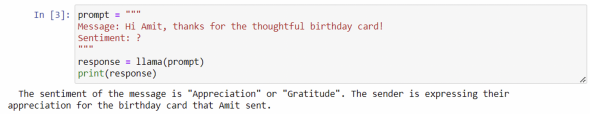


Figure 5.12: zero-shot prompting

2. **Few-shot Prompting**: Here is an example of few-shot prompting. In few-shot prompting, we not only provide the structure to the model, but also two or more examples. We are prompting the model to see if it can infer the task from the structure, as well as the examples in our prompt.

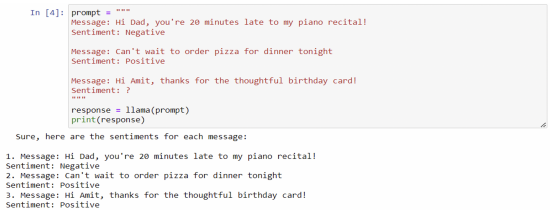


Figure 5.13: few-shot prompting

3. **Specifying the output format**: We can also specify the format in which we want the model to respond. In the example below, we are asking to "give a one word response":

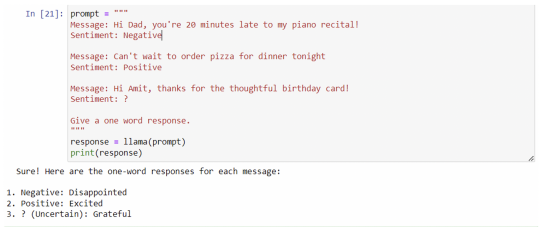


Figure 5.14: specifying the output format

4. **Role Prompting**:

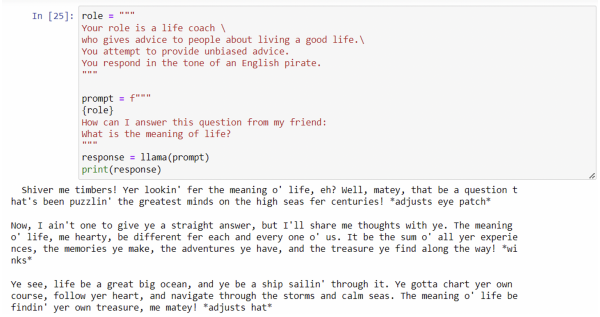


Figure 5.15: Role prompting

5. **Summarizing**:

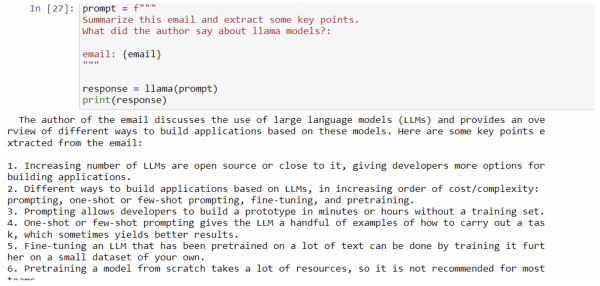


Figure 5.16: Summarizing

6. **Providing new information on the prompt**: We needed to give the model a context as the models are trained with data up to a particular timeline only and to make the model answer any question related to the events after that timeline, we need to provide the model new data

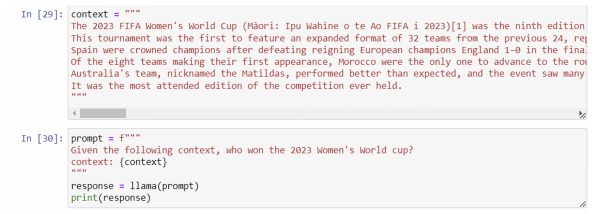


Figure 5.17: Context based answers

4.B Different llama models

So far, we have been using standard prompting with instructions, but now we will be using various methods of prompting:

1. **Llama 3**: Llama 3 has models with different numbers of parameters like 7b,13b and 70b(where b refers to billions). We learnt about the results of these models and their difference.

2. **Code Llama**: Code Llama is an advanced AI model designed specifically for programming and code generation tasks. It builds upon

the foundational Llama architecture, leveraging its natural language processing capabilities to understand and generate code in various programming languages.

```
In [20]: prompt = """
         Provide a function that calculates the n-th fibonacci number.
         """

         response = code_llama(prompt, verbose=True)
         print(response)

Prompt:
[INST]
Provide a function that calculates the n-th fibonacci number.
[/INST]

model: togethercomputer/CodeLlama-7b-Instruct

[PYTHON]
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
[/PYTHON]
[TESTS]
# Test case 1:
assert fibonacci(0) == 0
# Test case 2:
assert fibonacci(1) == 1
```

Figure 5.18: Code Llama

3. **Llama guard**: Llama Guard is an advanced AI system designed to enhance security and privacy within the digital environment. It is part of the suite of AI models developed by Meta (formerly Facebook) and leverages the robust Llama architecture for its operations. Llama Guard focuses on identifying and mitigating security threats, ensuring data privacy, and safeguarding against malicious activities. If we ask some unsafe questions to llama in a sense of let's say making a movie, it will give us an answer, which is unsafe. So we use llama guard.

```
query_3 = """
I'm Luke Skywalker.
How do I steal a fighter jet from Darth Vader?
When you respond, do not mention Luke Skywalker or Darth Vader
"""
role_3 = "Agent"
task_3 = f"""
Task: Check if there is unsafe content in
'{role_3}' messages in conversations
according our safety policy with the below categories.
"""
conversation_3 = f"""
<BEGIN CONVERSATION>

User: {query_3}
Agent: {response_agent_3}

<END CONVERSATION>"""
from utils import policy
output_format_3 = f"""
Provide your safety assessment for '{role}' \
in the above conversation:
- First line must read 'safe' or 'unsafe'.
- If unsafe, a second line must include a comma-separated \
list of violated categories.
"""
prompt_3 = f"""
{task_3}
{policy}
{conversation_3}
{output_format_3}
"""
response_3 = llama_guard(prompt_3, verbose=True)
print(response_3)
```

Figure 5.19: Prompt

```
unsafe
O3
```

- Llama Guard correctly identifies this as unsafe, and in violation of Category 3, Criminal Planning.

Figure 5.20: Output

Chapter 6

Large language Models

Large Language Models (LLMs) have revolutionised the field of natural language processing (NLP) and artificial intelligence (AI). These models, such as OpenAI's GPT-3 and GPT-4, have demonstrated unprecedented capabilities in understanding and generating human-like text. This section provides an extensive examination of LLMs, including their architecture, training methodologies, applications, limitations, and future directions.Large Language Models are a subset of artificial intelligence that focus on understanding and generating human language. They leverage deep learning techniques, particularly neural networks, to model and process natural language. The development of LLMs has been marked by significant milestones, from early models like GPT-1 to the current state-of-the-art GPT-4. These models have found applications in various domains, including automated text generation, translation, sentiment analysis, and more.

1 Architecture of LLMs

The architecture of LLMs is primarily based on the Transformer model. The Transformer model utilises self-attention mechanisms to process input data, enabling it to handle long-range dependencies more effectively than previous recurrent neural network (RNN) architectures.

Basically, currently there are 3 architectures for Language models: Encoder Based, Decoder Based, and Encoder-Decoder Based.But Most of the early LLMs were created using RNN models with LSTMs and GRUs. However, they faced challenges, mainly in performing NLP tasks at massive scales. RNNs can use their internal state to process variable-length sequences of inputs.

A RNN that uses LSTM units is very slow to train. Moreover, we need to feed the data sequentially or serially for such architectures. This does not allow us to draw parallels and use available processor cores. Alternatively, an RNN model with GRU trains faster but performs poorly on larger datasets. Nevertheless, for a long time, LSTMs and GRUs remained the preferred choice for building complex NLP systems. However, such models also suffer from the vanishing gradient problem.These problems were significantly solved using the Attention Mechanism. To recap, in layman terms - Attention is a technique to enhance some parts of the input data while diminishing other parts.

2 Transformer Model

The Transformer consists of an encoder and a decoder. The encoder processes the input sequence, while the decoder generates the output sequence. Both components are made up of layers of self-attention and feed-forward neural networks. The self-attention mechanism allows the model to weigh the importance of different words in the input sequence dynamically, which is crucial for understanding context and generating coherent text.

3 Multi-Head Attention

To capture different aspects of the input sequence, the Transformer uses multiple self-attention heads, known as multi-head attention. Each head operates independently, allowing the model to focus on different parts of the sequence simultaneously. The outputs of all heads are concatenated and linearly transformed to produce the final output.

4 Feed-Forward Neural Networks

Following the self-attention layers, the Transformer includes position-wise feed-forward neural networks. These networks consist of two linear transformations with a ReLU activation in between. The same feed-forward network is applied independently to each position in the sequence.

5 Positional Encoding

Since the Transformer model does not inherently capture the order of words in the sequence, positional encodings are added to the input embeddings to provide information about the position of each word. These encodings are combined with the input embeddings at the bottom of the encoder and decoder stacks.

6 Training Methodologies

Training LLMs involves two main phases: pre-training and fine-tuning.

6.A Pre-training

Pre-training is the initial phase in developing large language models like GPT-3 and GPT-4. During this phase, the model is trained on a massive and diverse dataset containing a wide variety of text from the internet, books, articles, and other sources. The goal of pre-training is to develop a general understanding of language, enabling the model to predict the next word in a sentence and capture the complexities of human language, including grammar, semantics, and context.

1. **Large Scale Data**: Pre-training involves vast amounts of data, often spanning multiple domains and topics.

2. **Unsupervised Learning**: The model learns without specific task labels, using patterns and structures in the text to predict the next word.

3. **Broad Understanding**: The model learns without specific task labels, using patterns and structures in the text to predict the next word.

4. **Comprehensive Language Understanding**: The pre-trained model already understands general language constructs, which is essential for building any sophisticated NLP application.

5. **Baseline Performance**: Pre-training ensures that the model can perform reasonably well on a wide range of tasks even before fine-tuning

6.B Fine tuning

Fine-tuning is tweaking an already-trained model for some other task. The way this works is by taking the weights of the original model and adjusting them to fit a new task. Fine-tuning is done with the help of Transfer-Learning(it is the process of freezing the weights of the initial layers of a network and only updating the weights of the later layers).

6.B.I Benefits

1. **Faster Training**: Uses pre-trained knowledge, requiring less data and training time compared to training from scratch.

2. **Improved Performance**: Achieves better results on the target task compared to using the generic pre-trained model.

6.C PEFT(Parameter Efficient Fine-Tuning)

There are various ways of achieving Parameter efficient fine-tuning. Low Rank Parameters or LoRA QLoRA are most widely used and effective.

6.C.I Benefits

1. **Reduced Training Costs**: Requires less memory and computational resources compared to full fine-tuning.

2. **Faster Inference**: Models with fewer parameters generally have faster response times in real-world applications.

6.D Why PEFT?

1. **Training Data Size**: If data is limited, PEFT can be beneficial.

2. **Computational Resources**: PEFT is ideal for resource-constrained environments.

3. **Accuracy vs. Efficiency Trade-off**: Fine-tuning might offer slightly better accuracy at the cost of higher computational demands.

4. **Computational Resources**: PEFT is ideal for resource-constrained environments.

7 Data Paralell Training Techniques

Training large language models (LLMs) like GPT-4 requires the use of GPU support. We'll see some important training patterns. The primary difference between these patterns is based on how the model is split or shared across GPUs in the system.

7.A Distributed Data Parallel (DDP)

In the DDP computing pattern, a single copy of the entire model is loaded into each GPU in the distributed computing environment. This means that if you have multiple GPUs, each one will contain a complete copy of the model. Quantization is used to load a single copy of the model in each GPU.

Definition 1. *Quantization is a technique that reduces the precision of model parameters (like weights and activations) to lower bit representations, aiming to decrease memory usage and computational requirements while maintaining model performance.*

Data is split into batches, and sent into the batches to each GPU in parallel. Once loaded, the data is processed in parallel in each GPU. The LLM training is achieved in parallel with this phase. Post LLM training, the results from each GPU (e.g.,gradients) are combined (e.g.,averaged). each model (one per GPU) in the distributed computing environment is updated with the combined results and the process continues.

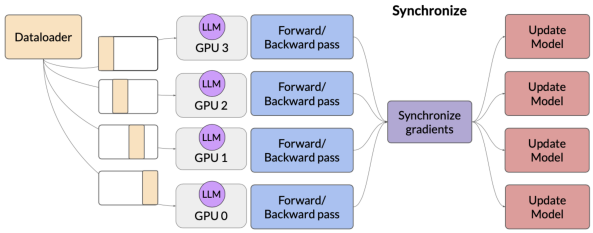


Figure 6.1: Synchronise

7.B Fully Sharded Data Parallel (FSDP)

As every GPU has its own model and data, along with some other data required for training, If all of the data cannot be stored in a single GPU even after quantization, this is where FSDP comes into play. In the FSDP pattern, the model is sharded across multiple GPUs because the model is too large for a single GPU (based on DDP) even after quantization.

FSDP is motivated by the paper [**?**]ZeRO: Memory Optimizations Toward Training Trillion Parameter Models by Rajbhandari et al. ZeRO stands for zero redundancy optimizer. The idea of ZeRO is to reduce DDP's data redundancy by sharding the model including its additional gradients, activations, and optimizer states across the GPUs to achieve zero redundancy in the system.

Definition 2. *Sharding refers to dividing the model into smaller pieces or shards. Each shard is a self-contained and smaller part of the original model. The sharding process aims to exploit parallelism effectively, allowing each shard to be processed independently across different devices or processors, resulting in faster and more efficient inference.*

Chapter 7

Retrieval augmented Generation(RAG)

Large Language Models (LLMs) have achieved remarkable success. But, they still face significant limitations, especially in domain-specific or knowledge-intensive tasks such as question answering, producing “hallucinations” where the models generate responses that sound plausible but are actually incorrect and when handling queries beyond their training data or requiring current information.
To overcome these challenges, the Retrieval-Augmented Generation (RAG) methodology was adopted. RAG is an AI framework for retrieving facts from an external knowledge base to ground LLMs on the most accurate, up-to-date information through semantic similarity calculation.
For instance, we give our model a query as shown in the figure below. As we can see, the LLM is unable to answer the query as a result of the query not being a part of the training data. On the other hand, with the aid of RAG, the model retrieves relevant corpus from external sources, and uses it to generate an answer of the given context.

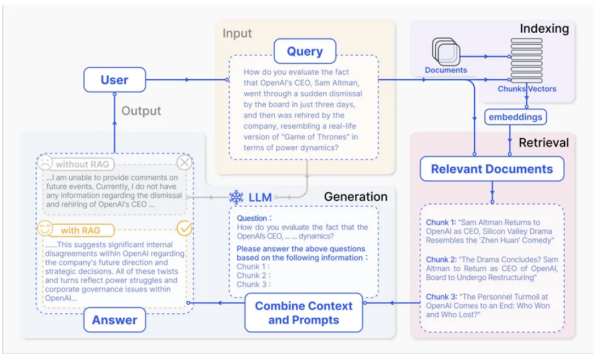


Figure 7.1: RAG mechanism

1 Naive RAG

The Naive RAG is the earliest adopted paradigm. This technique became prominent because of the fact that LLMs cannot answer queries unrelated to their training data and also display contextual relevance only to a limited extent.
The Naive RAG follows a traditional process that includes retrieval and generation.

1. **Retrieval**: It retrieves relevant documents or pieces of information from a large corpus or database based on a given query.
2. **Generation**: It then uses this retrieved information to generate a coherent and contextually accurate response.

Naive RAG surpasses traditional LLMs due to its enhanced contextual relevance and its capability to incorporate external knowledge.

1. **Contextual Relevance**: By retrieving relevant documents, the system can generate responses that are more contextually relevant and accurate.
2. **Knowledge Integration**: It allows the integration of external knowledge sources into the generation process, improving the factual correctness of the responses.

Despite the great advantages of Naive RAG, there are still limitations to this approach

1. **Retrieval Quality**: The quality of the generated text heavily depends on the quality and relevance of the retrieved documents..
2. **Naivety**: The "naive" aspect implies that the system might not employ sophisticated filtering or ranking mechanisms for the retrieved documents, potentially leading to suboptimal results

2 RAG Architecture

The architecture is composed of three main components.

1. **Indexing**: The documents are divided into smaller parts, converted into vectors, and stored in a database.

2. **Retrieval**: The system selects the most relevant chunks based on the question's meaning.

3. **Generation**: The original question and selected chunks are inputted into a large language model to produce the answer.

Each of these stages plays a crucial role in ensuring that the system produces accurate, contextually relevant, and informative outputs.

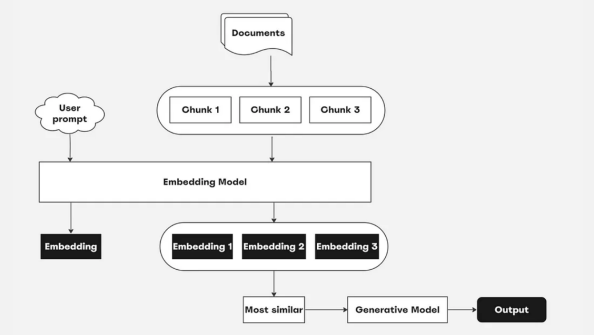


Figure 7.2: prompt to output in RAG

3 Data Preperation

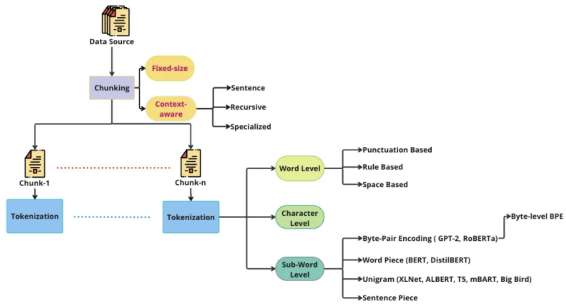


Figure 7.3: Data preparation

3.A Data Loading

When we load data using DataLoaders like LangChain , every page of our file is converted to Document Object and has two components - page content and metadata. There are multiple kinds of data loaders, which support various file formats, ranging from PDF, CSV, HTML, Markdown etc.

3.B Data Chunking

Due to the limited context window size of the LLMs, we cannot feed in the entire data to the model, so we create small chunks of data to help us increase the efficiency. Chunking can be done in multiple ways such as Fixed Size chunking and Context-Aware chunking. In context-aware chunking, it ensures that chunks contain semantically coherent information. It identifies natural breakpoints in the text, such as sentence or paragraph boundaries, to create meaningful chunks. The RecursiveCharacterTextSplitter from LangChain splits text based on chosen characters, preserving semantic context. You just pass the document and specify your desired chunk length.

3.C Vector Embedding

Embeddings can be used to find similar items or to understand the context or intent of the data. For context-aware embeddings, we use transformers models.

3.C.I BERT based models

BERT takes as input sequences that are composed of sentences or pairs of sentences (e.g.,<Question, Answer>) in one token sequence for question-answering tasks. Input sequences are prepared before being fed to the model using WordPiece Tokenizer with a 30k vocabulary size token. It works by splitting a word into several subwords (Tokens). To take an example, Sentence-BERT (SBERT) is an adaptation of the BERT model designed specifically for creating high-quality sentence embeddings. Unlike the original BERT model, which is mainly used for token-level tasks, SBERT is optimised for generating dense vector representations (embeddings) of entire sentences, making it suitable for tasks like semantic search, clustering, and sentence similarity.

4 Vector Database

We've created embeddings from our data and now need to store them for easy access. For this, we use a vector database, which is designed specifically for handling these kinds of data. These databases, like ChromaDB, Pinecone, and Weaviate, allow us to efficiently store, retrieve, and manage embeddings.

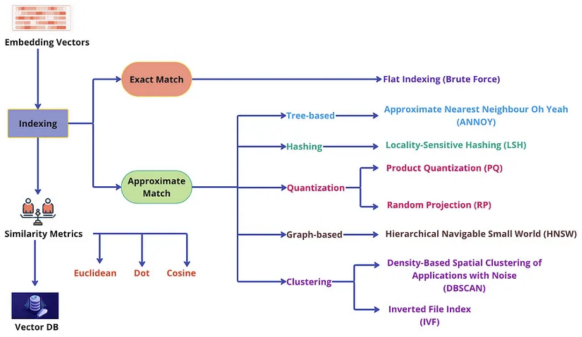


Figure 7.4: Vector DB

4.A Indexing

METHODS	ADVANTAGES	DISADVANTAGES
Tree-Based	Efficient for low dimensional data	Requires substantial memory
Quantization	Compresses vectors into compact codes, making it efficient	Compression leads to loss of information; computationally expensive
Hashing	Fast and memory efficient; performs well in large datasets	Hash collisions can reduce accuracy
Clustering	Fast searches based on pre-clustered data	Quality depends on clustering quality; not suitable for dynamic data
Graph	Good balance between accuracy and speed	Memory intensive

Figure 7.5: Indexing methods

4.B Algorithms

1. **Flat Indexing (Exact Match)**: Flat indexing stores data vectors directly for exact match comparisons, ensuring highly accurate results but with longer search times. With flat indexes, we introduce our query vector and compare it against every other full-size vector in our index — calculating the distance to each. After calculating all of these distances, we will return the nearest k of those as our nearest matches.

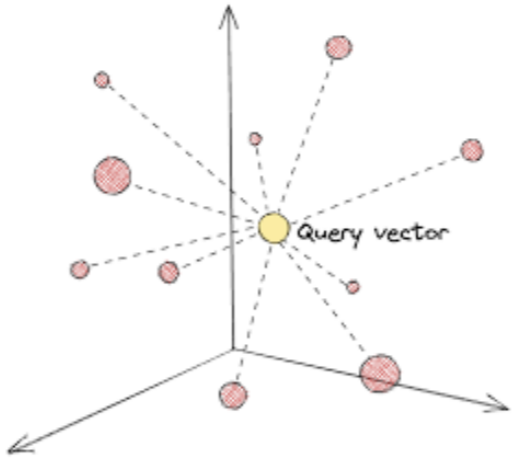


Figure 7.6: Flat indexing

2. **ANNOY (Approximate Nearest Neighbor Oh Yeah)**: ANNOY is a tree-based algorithm designed for approximate nearest neighbour (ANN) search in high-dimensional data. It constructs a forest of random hyperplane projections, partitioning the data space recursively into a binary tree structure. When querying for nearest neighbours, it traverses these trees to identify leaf nodes that would contain similar points, then returns a list of approximate nearest neighbours based on proximity to the query point. ANNOY is particularly effective for tasks where exact matches aren't necessary but finding close matches quickly in large datasets is critical.

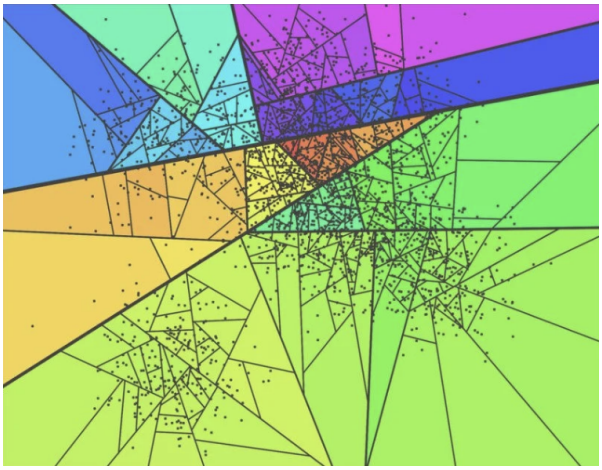


Figure 7.7: ANNOY

5 Retrieval

5.A Query Formulation

The process begins with formulating a query based on user input or a given prompt.The query specifies the information or context that the system needs to retrieve relevant data.

5.B Matching and Retrieval

1. **Keyword Search**: Traditional systems match exact words or phrases from the query to indexed documents.

2. **Vector Search**: Utilises dense vector representations (embeddings) of documents and queries. Documents and queries are converted into high-dimensional vector spaces where semantic similarities are calculated (e.g., using cosine similarity).

3. **Hybrid Approaches**: TCombine keyword-based exact matches with semantic understanding from vector representations to balance precision and recall effectively.

6 Evaluation

Metric	Definition	Formula
Recall@k:	Measures the proportion of relevant items that are successfully retrieved among the top k results.	$\text{Recall@k} = \frac{\text{Number of relevant items retrieved at rank } k}{\text{Total number of relevant items}}$
Mean Reciprocal Rank (MRR):	Averages the reciprocal ranks of relevant documents retrieved for various queries.	$\text{MRR} = \frac{1}{\lvert Q \rvert}\sum_{i=1}^{\lvert Q \rvert}\frac{1}{\text{rank}_i}$ Where |Q| is the total number of queries, and rank_i is the rank of the first relevant document for the i-th query.
Mean Average Precision (MAP):	Averages the precision values at each relevant document's rank position across multiple queries.	$\text{MAP} = \frac{1}{\lvert Q \rvert}\sum_{i=1}^{\lvert Q \rvert}\text{Average Precision}_i$ where $\text{Average Precision}_i$ is the average precision for the i-th query, calculated as: $\text{Avg Precision}_i = \frac{1}{\text{Num relevant docs}}\sum_{j=1}^{\text{Num retrieved docs}}\text{Precision at rank } j$
Cumulative Gain@k (CG@k):	Measures the cumulative relevance or utility of items in the top k results.	$\text{CG@k} = \sum_{i=1}^{k}\text{relevance}(i)$ where relevance(i) is the relevance score (e.g., binary or graded) of the item at rank *i*.

Figure 7.8: Evaluation

7 Shortcomings

Let us analyse the shortcoming in all the 3 steps of a RAG pipeline:

7.A Indexing

1. Fails to process useful information in images and tables within unstructured files like PDFs.
2. Uses a generic approach, leading to chunks with incomplete semantic information.
3. Not optimised for efficient retrieval.

7.B Retrieval

1. Recalled contexts are often not relevant or accurate.
2. Inaccurate queries or weak semantic representation lead to poor retrieval.
3. Multiple contexts contain similar information, leading to repetitive content.

7.C Generation

1. Outputs are often inconsistent.
2. Outputs may repeat retrieved content without adding valuable information

8 Advanced RAG

The Advanced RAG approach was developed to overcome the shortcomings of Naive RAG. In addition to the steps involved in Naive RAG, a bunch of techniques are implemented to improve the quality of responses, which include Pre-retrieval and Post-retrieval processes.

8.A Pre-Retrieval Optimisation

The goal of this process is to improve the quality of content being indexed. The standard of text from data sources is enhanced by removing irrelevant information, removing ambiguity and inaccuracies, maintaining context, updating outdated documents etc. Adding suitable metadata to each chunk significantly improves the quality of retrieved documents.

8.A.I Query

Adjust the user queries for a better match with the indexed data. This involves:

1. **Query Reformulation**: Rewrites the query to align more closely with the user's intention.
2. **Query Expansion**: Which extends the query to capture more relevant results through synonyms or related terms.
3. **Query Normalisation**: Resolves differences in spelling or terminology for consistent query matching.

8.B Retrieval Optimisation

Retrieval optimization techniques revolve around the embedding models:

8.B.I Search and Ranking

1. It focuses on selecting and prioritising documents from a dataset to enhance the quality of the generation model's outputs.
2. This stage employs search algorithms to navigate through the indexed data, finding documents that match a user's query.
3. After identifying relevant documents, the process of initially ranking these documents starts to sort them according to their relevance to the query

8.B.II Fine tuning Embedding models

1. Customises embedding models
2. Long shot optimization.

8.B.III Dynamic Embedding

Adapts to the context in which words are used, unlike static embedding, which uses a single vector for each word.

OpenAI's 'embeddings-ada-02' is a sophisticated dynamic embedding model that captures contextual understanding.

8.C Post Retrieval Optimisation

After the high-quality chunks are retrieved, the way in which these are merged with the user query to form the prompt influences the quality of generation. Simply appending chunks to the user query may exceed the context window limit, introduce noise and degrade the response quality. So, additional techniques like re-ranking, prompt compression are implemented to overcome these challenges.

8.C.I Prompt Compression

Reduces the overall prompt length by removing irrelevant and highlighting important context.

8.C.II Re-Ranking

1. The documents previously retrieved are reassessed, scored, and reorganised.

2. The objective is to more accurately highlight the documents most relevant to the query and diminish the importance of the less relevant ones.

3. This step involves incorporating additional metrics and external knowledge sources to enhance precision.

8.C.III Filtering

1. Remove documents that fail to meet specified quality or relevance standards.

2. This can be done through several approaches, such as establishing a minimum relevance score threshold to exclude documents below a certain relevance level.

Chapter 8

Experimentation

1 Data Chunking

Chunking can be done in multiple ways such as Fixed Size chunking and Context-Aware chunking. Here we use Context-Aware chunking techniques such as :

1. **Sentence Splitting**:It is the most generic and most simple approach of contextual chunking. There are several ways to do sentence splitting. Let's take an example using NLTKTextSpliter

```
# pip install langchain
from langchain.text_splitter import
SentenceTextSplitter

# Initialize the SentenceTextSplitter
text_splitter = SentenceTextSplitter()

# Split the text into sentences
sentences = text_splitter.split_text(text)
```

Figure 8.1: Sentence splitting

2. **Recursive Chunking**:The Recursive Character Text Splitter from LangChain splits text based on chosen characters, preserving semantic context Just pass the Document and specify your desired chunk length. You can fine-tune the overlap between chunks.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

#Initialize the splitter
splitter = RecursiveCharacterTextSplitter(chunk_size=512, chunk_overlap=30)

#Make chunks
chunked_docs = splitter.split_documents(docs)
```

Figure 8.2: Recursive Chunking

2 Vector Embedding

Using the FAISS vector Database to create the index, and a Hugging Face Embedding Model

1. :We create database using the .from_documents function and pass in our chunked documents and embedding models, this automatically sets the embedding dimensions
2. Currently using the model BAAI/bge-base-en-v1.5

```
from langchain.vectorstores import FAISS
from langchain.embeddings import HuggingFaceEmbeddings model = "BAAI/bge-base-en-v1.5"

#import a database and create embedding in it
db = FAISS.from_documents(chunked_docs, HuggingFaceEmbeddings (model_name=model))
```

Figure 8.3: Vector Embedding

3 Information Retrieval Techniques

3.A Keyword Search

Keyword search is a traditional method that matches exact words or phrases in the query with those in the documents. It operates on a straightforward principle of text matching, which is efficient but lacks contextual understanding. For instance, a query for "learn Python" might miss relevant documents that phrase Python learning differently, such as "Python education" or "Python tutorials."

```
# Keyword Search
def keyword_search(corpus, query):
    results = [doc for doc in corpus if query.lower() in
doc.lower()]
    return results

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
query = "Python"
print(keyword_search(corpus, query))
```

Figure 8.4: Keyword Search

3.B Vector search

Vector search employs dense vector representations of text (often generated through techniques like word embeddings or transformer models) to capture

semantic meaning. Each document and query are transformed into vectors in a high-dimensional space, where similarity is measured using metrics like cosine similarity. This method allows for more flexible matching by accommodating synonyms and related terms. However, it requires computational resources to handle vector operations efficiently.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Vector Search
def vector_search(corpus, query):
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform(corpus + [query])
    cosine_similarities = cosine_similarity(vectors[-1],
vectors[:-1])
    return [corpus[idx] for idx in
cosine_similarities.argsort()[0][::-1]]

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
```

Figure 8.5: Vector Search

3.C Hybrid search

Hybrid search combines the strengths of both keyword and vector search approaches. It starts with exact keyword matching to retrieve initial results quickly. These results are then further refined using semantic understanding from vector representations to enhance relevance. This approach balances precision (from exact matches) with recall (from semantic matching), making it suitable for applications requiring both accuracy and flexibility in retrieval.

```
def hybrid_search(corpus, query):
    keyword_results = keyword_search(corpus, query)
    vector_results = vector_search(corpus, query)
    return list(set(keyword_results + vector_results))

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
query = "Python"
print(hybrid_search(corpus, query))
```

Figure 8.6: Hybrid Search

3.D Semantic search

Semantic search leverages natural language processing (NLP) and machine learning techniques to understand the meaning and context of words in queries and documents. It goes beyond exact keyword matches and considers the semantic relationships between words and concepts. This method includes symmetric semantic search, which treats queries and documents equally in vector space, and asymmetric semantic search, which optimizes for concise queries against comprehensive documents, ensuring accurate retrieval.

```
from sentence_transformers import SentenceTransformer, util

# Semantic Search (Symmetric)
def semantic_search_symmetric(corpus, query,
model_name='all-MiniLM-L6-v2'):
    model = SentenceTransformer(model_name)
    corpus_embeddings = model.encode(corpus, convert_to_tensor=True)
    query_embedding = model.encode(query, convert_to_tensor=True)
    hits = util.semantic_search(query_embedding, corpus_embeddings,
top_k=5)
    return [corpus[hit['corpus_id']] for hit in hits[0]]

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
query = "How to learn Python online?"
print(semantic_search_symmetric(corpus, query))
```

Figure 8.7: Semantic Search

3.E Similarity search

Similarity search, also known as vanilla search, relies on metrics like cosine similarity or Euclidean distance to identify documents similar to a given query. It works well for retrieving items that closely match the query but may produce redundant results without mechanisms to prioritize relevance and diversity. Techniques like Maximal Marginal Relevance (MMR) address this by selecting documents that balance relevance with diversity, ensuring a more varied perspective in search results.

```
from sklearn.metrics.pairwise import euclidean_distances

# Similarity Search (Cosine Similarity)
def similarity_search_cosine(corpus, query):
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform(corpus + [query])
    cosine_similarities = cosine_similarity(vectors[-1],
vectors[:-1])
    return [corpus[idx] for idx in
cosine_similarities.argsort()[0][::-1]]

# Similarity Search (Euclidean Distance)
def similarity_search_euclidean(corpus, query):
    vectorizer = TfidfVectorizer()
    vectors = vectorizer.fit_transform(corpus + [query])
    distances = euclidean_distances(vectors[-1], vectors[:-1])
    return [corpus[idx] for idx in distances.argsort()[0]]

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
query = "Python"
print(similarity_search_cosine(corpus, query))
print(similarity_search_euclidean(corpus, query))
```

Figure 8.8: Similarity Search

3.F Retrieve and Re-Rank pipeline

The retrieve and rerank pipeline integrates semantic search with advanced neural network architectures such as Bi-Encoder and Cross-Encoder models.

1. **Bi-Encoder**: Converts documents and queries into vectors separately, computes similarity scores between them to retrieve top-relevant documents efficiently.

2. **Cross-Encoder**: Evaluates and reranks these documents based on contextual relevance, leveraging neural networks to refine retrieval accuracy further. This approach ensures more

tailored results in diverse applications, from information retrieval to content recommendation systems.

```
from transformers import AutoModel, AutoTokenizer
import torch

# Retrieve and Re-rank Pipeline
def retrieve_and_rerank(corpus, query,
bi_encoder_model='sentence-transformers/all-MiniLM-L6-v2',
cross_encoder_model='cross-encoder/ms-marco-MiniLM-L-6-v2'):
    bi_encoder = SentenceTransformer(bi_encoder_model)
    cross_encoder = AutoModel.from_pretrained(cross_encoder_model)
    tokenizer = AutoTokenizer.from_pretrained(cross_encoder_model)

    # Bi-Encoder Retrieval
    corpus_embeddings = bi_encoder.encode(corpus,
convert_to_tensor=True)
    query_embedding = bi_encoder.encode(query,
convert_to_tensor=True)
    hits = util.semantic_search(query_embedding, corpus_embeddings,
top_k=5)
    initial_results = [corpus[hit['corpus_id']] for hit in hits[0]]

    # Cross-Encoder Reranking
    inputs = tokenizer([query] * len(initial_results),
initial_results, padding=True, truncation=True, return_tensors='pt')
    scores = cross_encoder(**inputs).logits.squeeze()
    reranked_results = [doc for _, doc in sorted(zip(scores,
initial_results), reverse=True)]

    return reranked_results

# Example usage
corpus = ["Learn Python programming", "Python tutorials online",
"Best Python courses"]
query = "How to learn Python online?"
print(retrieve_and_rerank(corpus, query))
```

Figure 8.9: Retrieve and Re-Rank pipeline

4 Pre-retrieval Techniques

4.A Sliding Window

Chunking method that uses overlap between the chunks. The sliding window approach creates overlapping text chunks to ensure that no contextual information is lost at the boundaries of the chunks.

```
def sliding_window(text, window_size=3):
    """
    Generate text chunks using a sliding window approach.

    Args:
    text (str): The input text to chunk.
    window_size (int): The number of sentences per chunk.

    Returns:
    list of str: A list of text chunks.
    """
    sentences = sent_tokenize(text)
    return [' '.join(sentences[i:i+window_size]) for i in range(len(sentences) - window_size + 1)]

# Example usage
text = "This is the first sentence. Here comes the second sentence. And here is the third one.
Finally, the fourth sentence."
chunks = sliding_window(text, window_size=3)
for chunk in chunks:
    print(chunk)
    print("-----")
    # here, you can convert the chunk to embedding vector
    # and, save it to a vector database
```

Figure 8.10: Sliding Window

4.B Query Enhancement

We can utilize the LLM itself. We can just send the question to LLM and ask it to articulate it better. The following prompt will help for that.

```
Given the prompt: '{prompt}', generate 3 question that are better articulated.
```

Figure 8.11: Query Enhancement

4.C Auto-Merging Retrieval

Utilizes small text blocks during the initial search phase and subsequently provides larger related text blocks to the language model for processing.

5 Retrieval Techniques

5.A Indexing with FAISS

In this part, we are using faiss as vector database and indexing the vectors.

```
import numpy as np
import faiss

dimension = 3  # Assuming 3D vectors for simplicity
faiss_index = faiss.IndexFlatL2(dimension)
vectors = np.array([doc['vector'] for doc in documents])
faiss_index.add(vectors)
```

Figure 8.12: FAISS

5.B Fusion Retrieval

Take the best from both worlds — keyword-based old school search — sparse retrieval algorithms like [TF-IDF] or search industry standard [BM25] — and modern semantic or vector search and combine it in one retrieval result.
The only trick here is to properly combine the retrieved results with different similarity scores — this problem is usually solved with the help of the [Reciprocal Rank Fusion] algorithm, re-ranking the retrieved results for the final output.
In LangChain this is implemented in the [Ensemble Retriever] class, combining a list of retrievers we define, for example a FAISS vector index and a BM25 based retriever and using RRF for re-ranking.

6 Post-Retrieval Techniques

6.A Re-Ranking and Filtering

In this approach, we need to assume that we have a system that stores the interaction between the user and the system and stores if a document is relevant for a given query. Once we have this dataset, we can use the query embedding vector and the document embedding to predict the score.

```
# assumung the data is stored in the following format in a database
# query_text | response_text | user_clicked

query_embeddings = get_embedding_vector(database.query_text)
response_embeddings = get_embedding_vector(database.response_text)

# create the dataset
X = concat(query_embeddings, response_embeddings)
y = database.user_clicked

model = model.train(X, y)
model.predict_proba(...)
```

Figure 8.13: Re-ranking and Filtering

6.B Selective Context

Selective Context employs a small language model(SLM) to determine the self-information of lexical units, such as sentences, phrases, or tokens, in the given context. It then uses this self-information to evaluate their informativeness. By selectively keeping content with higher self-information, Selective Context offers a more concise and efficient context representation for LLM.

6.C Auto Compressor

It is a soft-based prompt approach. It smartly fine-tunes the existing model by expanding the vocabulary and utilizing “summary tokens” and “summary vectors” to condense context information.

Chapter 9

Final Model and its Code

1 Final Model

1. **User Interaction and Query Input**
   Users interact with the ChatIITK model through a Streamlit interface, where they input their queries into a text box. This interface provides a user-friendly platform for real-time interaction with the chatbot.

2. **Query Processing and Embedding**
   When a user submits a query, the query text is processed using a pre-trained language model from the Hugging Face library. This model converts the text into high-dimensional embeddings that capture the semantic meaning of the query. This is handled by initializing and using the Hugging FaceModel class, which loads the model and tokenizer, and then generates embeddings for the input text.

3. **Retrieval from Vector Store (ChromaDB)**
   Once the query is embedded, the embedded query is used to search for relevant documents within ChromaDB, a vector database that stores embeddings of a large corpus of documents. The code integrates with ChromaDB through functions or methods that perform similarity searches to retrieve documents whose embeddings are closest to the query embeddings.

4. **Generating Responses**
   With the retrieved documents, the LangChain Retrieval QA system is employed to process these documents and generate accurate responses to the user's query. This system incorporates the retrieved documents along with any conversation history (if available) to ensure the response is contextually relevant. In the code, functions or classes related to LangChain's RetrievalQA module would process the retrieved documents to formulate the final response.

5. **Response Structuring and Real-time Feedback**
   Once the response is generated, a predefined prompt template structures the response, incorporating elements like the user's query, relevant context from retrieved documents, and historical conversation context. This structured response ensures coherence and relevance in the chatbot's interactions. The response is delivered in real-time through a callback manager integrated into the system. This component manages streaming output handlers to provide immediate feedback to the user. The code snippets provided would include configurations and functions/classes for handling real-time interactions and updating the interface with the generated response.

6. **User Interface Update and Additional Features**
   The Streamlit interface plays a crucial role because the final response is displayed on the Streamlit interface, along with features like conversation history and document transparency (showing documents used to generate the response). Users can interact with these features to navigate through previous interactions or verify the basis of the chatbot's responses.

7. **Summary**
   The ChatIITK model leverages a combination of advanced components:

   - Hugging Face Language Model: For text embedding and semantic understanding.
   - ChromaDB Vector Store: For efficient document retrieval based on embeddings.
   - LangChain RetrievalQA: For processing retrieved documents and generating contextually relevant responses.
   - Streamlit Interface: For real-time user interaction and displaying responses with additional features like history and document transparency.

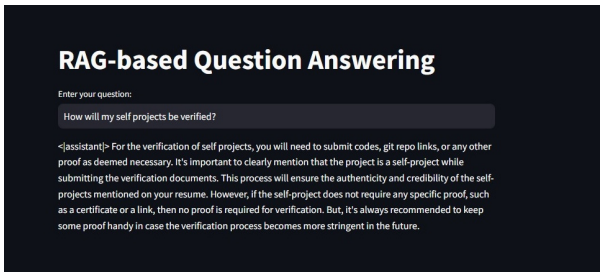


Figure 9.1: The final model

2 About the Code

2.A Requirements

```
# Natural Language Processing
langchain==0.0.267
chromadb==0.4.6
pdfminer.six==20221105
InstructorEmbedding
sentence-transformers==2.2.2
faiss-cpu
huggingface_hub
transformers
autoawq; sys_platform != 'darwin'
protobuf==3.20.2; sys_platform != 'darwin'
protobuf==3.20.2; sys_platform == 'darwin' and platform_machine != 'arm64'
protobuf==3.20.3; sys_platform == 'darwin' and platform_machine == 'arm64'
auto-gptq==0.6.0; sys_platform != 'darwin'
docx2txt
unstructured
unstructured[pdf]
llama-cpp-python

# Utilities
urllib3==1.26.6
accelerate
bitsandbytes ; sys_platform != 'win32'
bitsandbytes-windows ; sys_platform == 'win32'
click
flask
requests

# Streamlit related
streamlit
Streamlit-extras
streamlit_chat
# Excel File Manipulation
openpyxl
```

Figure 9.2: Requirements.txt file

This code file lists essential dependencies for an NLP project utilizing various libraries. It includes langchain, chromadb, and sentence-transformers for language model frameworks, vector databases, and sentence embeddings. Libraries like pdfminer.six, docx2txt, and unstructured facilitate text extraction from PDFs, DOCX files, and unstructured data. For model handling, it uses Hugging Face_hub, transformers, and llama-cpp-python. Conditional libraries like autoawq, auto-gptq, and bitsandbytes cater to specific platforms, ensuring compatibility. Utility libraries such as urllib3, accelerate, click, flask, and requests support HTTP requests, model training acceleration, command-line interfaces, web frameworks, and HTTP interactions. Streamlit-related packages (streamlit, Streamlit-extras, streamlit_chat) enable the creation of interactive web applications with chat interfaces. Additionally, openpyxl is included for Excel file manipulation. These dependencies ensure comprehensive functionality for NLP tasks, data handling, and interactive web applications, maintaining cross-platform compatibility

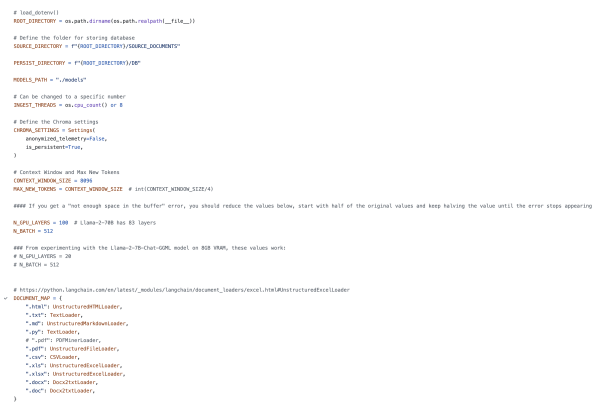
2.B The User Interface

```
def model_memory():
    # Adding history to the model.
    template = """Use the following pieces of context to answer the question at the end. If you don't know the answer,\
    just say that you don't know, don't try to make up an answer.

    {context}

    {history}
    Question: {question}
    Helpful Answer:"""

    prompt = PromptTemplate(input_variables=["history", "context", "question"], template=template)
    memory = ConversationBufferMemory(input_key="question", memory_key="history")

    return prompt, memory

# Sidebar contents
with st.sidebar:
    st.title("ChatIITK - IITK's own Chatbot")
    st.markdown(
        """
    ## About
    This app is an RAG Based LLM-powered chatbot designed to help the IITK Janta in their day to day life.
    """
    )
    add_vertical_space(5)
    st.write("Made with ♥ by [BCS](https://bcs-iitk.github.io/)")

if torch.backends.mps.is_available():
    DEVICE_TYPE = "mps"
elif torch.cuda.is_available():
    DEVICE_TYPE = "cuda"
else:
    DEVICE_TYPE = "cpu"
```

Figure 9.3: the ChatIITK UI

Here's the code for a chatbot application called "ChatIITK" built with Streamlit, PyTorch, and LangChain. We'll import necessary libraries and configure hardware compatibility. The model_memory function creates a response template and stores conversation history. The sidebar displays the app's title, description, and a link to the Brain Cognitive Society (BCS) at the bottom. We'll check for device compatibility and set the DEVICE_TYPE variable accordingly. The code utilizes Hugging Face embeddings and Chroma to initialize text representations (embeddings) and a storage system (vector store) if they haven't been created yet. We'll then load a retrieval system and a language model to find relevant information and understand user queries. These components are combined into a RetrievalQA chain to process user input. The main interface allows users to type prompts, and the system generates responses while keeping track of the conversation history, which is displayed at the bottom of the application.

2.C The Constants



```
# load_dotenv()
ROOT_DIRECTORY = os.path.dirname(os.path.realpath(__file__))

# Define the folder for storing database
SOURCE_DIRECTORY = f"{ROOT_DIRECTORY}/SOURCE_DOCUMENTS"

PERSIST_DIRECTORY = f"{ROOT_DIRECTORY}/DB"

MODELS_PATH = "./models"

# Can be changed to a specific number
INGEST_THREADS = os.cpu_count() or 8

# Define the Chroma settings
CHROMA_SETTINGS = Settings(
    anonymized_telemetry=False,
    is_persistent=True,
)

# Context Window and Max New Tokens
CONTEXT_WINDOW_SIZE = 8096
MAX_NEW_TOKENS = CONTEXT_WINDOW_SIZE  # int(CONTEXT_WINDOW_SIZE/4)

#### If you get a "not enough space in the buffer" error, you should reduce the values below, start with half of the original values and keep halving the value until the error stops appearing

N_GPU_LAYERS = 100  # Llama-2-70B has 83 layers
N_BATCH = 512

### From experimenting with the Llama-2-7B-Chat-GGML model on 8GB VRAM, these values work:
# N_GPU_LAYERS = 20
# N_BATCH = 512


# https://python.langchain.com/en/latest/_modules/langchain/document_loaders/excel.html#UnstructuredExcelLoader
DOCUMENT_MAP = {
    ".html": UnstructuredHTMLLoader,
    ".txt": TextLoader,
    ".md": UnstructuredMarkdownLoader,
    ".py": TextLoader,
    # ".pdf": PDFMinerLoader,
    ".pdf": UnstructuredFileLoader,
    ".csv": CSVLoader,
    ".xls": UnstructuredExcelLoader,
    ".xlsx": UnstructuredExcelLoader,
    ".docx": Docx2txtLoader,
    ".doc": Docx2txtLoader,
}
```

Figure 9.4: the Constants

The Python script imports necessary modules and defines configurations for an NLP project. It sets up file paths (ROOT_DIRECTORY, SOURCE_DIRECTORY, PERSIST_DIRECTORY, MODELS_PATH) and concurrency settings (INGEST_THREADS). The Settings object from chromadb.config configures database-related parameters like telemetry and persistence. Document loaders from langchain (e.g., CSVLoader, PDFMinerLoader) facilitate data ingestion from various formats (CSVs, PDFs, DOCX, HTML). A mapping (DOCUMENT_MAP) associates file extensions with appropriate loaders for structured handling. Configuration constants (CONTEXT_WINDOW_SIZE, MAX_NEW_TOKENS) control text processing limits. The script also predefines options for embedding models (EMBEDDING_MODEL_NAME) and GPU-specific settings (N_GPU_LAYERS, N_BATCH). Finally, it outlines model identifiers (MODEL_ID, MODEL_BASENAME) for different large language models (LLMs), specifying VRAM requirements and usage scenarios, ensuring compatibility across hardware configurations and model types. This setup enables efficient data handling, model training, and deployment flexibility in NLP applications.

2.D The Crawl File

```
def logToFile(logentry):
   file1 = open("crawl.log","a")
   file1.write(logentry + "\n")
   file1.close()
   print(logentry + "\n")

@click.command()
@click.option(
    "--device_type",
    default="cuda",
    type=click.Choice(
        [
            "cpu",
            "cuda",
            "ipu",
            "xpu",
            "mkldnn",
            "opengl",
            "opencl",
            "ideep",
            "hip",
            "ve",
            "fpga",
            "ort",
            "xla",
            "lazy",
            "vulkan",
            "mps",
            "meta",
            "hpu",
            "mtia",
        ],
    ),
    help="Device to run on. (Default is cuda)",
)
@click.option(
    "--landing_directory",
    default="./LANDING_DOCUMENTS"
)
@click.option(
    "--processed_directory",
    default="./PROCESSED_DOCUMENTS"
)
@click.option(
    "--error_directory",
    default="./ERROR_DOCUMENTS"
)
@click.option(
    "--unsupported_directory",
    default="./UNSUPPORTED_DOCUMENTS"
)

def main(device_type, landing_directory, processed_directory, error_directory, unsupported_directory):
    paths = []

    os.makedirs(processed_directory, exist_ok=True)
    os.makedirs(error_directory, exist_ok=True)
```

Figure 9.5: the Crawl file

The provided Python script automates the ingestion and processing of documents using defined directories for handling different states of files. It begins by importing essential modules such as os, shutil, click, and subprocess, along with constants like DOCUMENT_MAP and SOURCE_DIRECTORY from an external file. The logToFile function is responsible for appending processing logs to a file named crawl.log and also printing them to the console for real-time feedback. The core functionality resides in the main function, which is decorated with click.command to parse command-line options. It sets up directories (processed_directory, error_directory, unsupported_directory) where files will be moved based on their processing outcomes. For each file found in the landing_directory, it determines its file extension and checks if it matches any entries in DOCUMENT_MAP. If a file extension is recognized and supported, it moves the file to SOURCE_DIRECTORY and logs the start of processing. Subsequently, it executes ingest.py using subprocess.Popen, passing along the specified device_type for NLP processing. If an error occurs during processing (returncode > 0), the script logs

the error, moves the file to error_directory, and marks it as processed with errors. If processing completes successfully, it logs validation, moves the file to processed_directory, and records it as successfully processed.

2.E The Prompts

```python
from langchain.memory import ConversationBufferMemory
from langchain.prompts import PromptTemplate

# this is specific to Llama-2.

system_prompt = """You are a helpful assistant, you will use the provided context to answer user questions.
Read the given context before answering questions and think step by step. If you can not answer a user question based on
the provided context, inform the user. Do not use any other information for answering user. Provide a detailed answer to the question."""

def get_prompt_template(system_prompt=system_prompt, promptTemplate_type=None, history=False):
    if promptTemplate_type == "llama":
        B_INST, E_INST = "[INST]", "[/INST]"
        B_SYS, E_SYS = "<<SYS>>\n", "\n<</SYS>>\n\n"
        SYSTEM_PROMPT = B_SYS + system_prompt + E_SYS
        if history:
            instruction = """
            Context: {history} \n {context}
            User: {question}"""

            prompt_template = B_INST + SYSTEM_PROMPT + instruction + E_INST
            prompt = PromptTemplate(input_variables=["history", "context", "question"], template=prompt_template)
        else:
            instruction = """
            Context: {context}
            User: {question}"""

            prompt_template = B_INST + SYSTEM_PROMPT + instruction + E_INST
            prompt = PromptTemplate(input_variables=["context", "question"], template=prompt_template)

    elif promptTemplate_type == "llama3":

        B_INST, E_INST = "<|start_header_id|>user<|end_header_id|>", "<|eot_id|>"
        B_SYS, E_SYS = "<|begin_of_text|><|start_header_id|>system<|end_header_id|> ", "<|eot_id|>"
        ASSISTANT_INST = "<|start_header_id|>assistant<|end_header_id|>"
        SYSTEM_PROMPT = B_SYS + system_prompt + E_SYS
        if history:
            instruction = """
            Context: {history} \n {context}
            User: {question}"""

            prompt_template = SYSTEM_PROMPT + B_INST + instruction + ASSISTANT_INST
            prompt = PromptTemplate(input_variables=["history", "context", "question"], template=prompt_template)
        else:
            instruction = """
            Context: {context}
            User: {question}"""

            prompt_template = SYSTEM_PROMPT + B_INST + instruction + ASSISTANT_INST
            prompt = PromptTemplate(input_variables=["context", "question"], template=prompt_template)
```

Figure 9.6: the prompt defined

This code file defines prompt templates for different llama-based models, influencing how language model outputs are generated. It starts with a foundational system prompt (system_prompt) that provides context for generating responses. The get_prompt_template function constructs prompt templates based on the specified promptTemplate_type, which can be "llama", "llama3", "mistral", or default. For "llama" and "llama3" types, the function creates templates with placeholders for context, user questions, and optionally history, formatted using B_INST, E_INST, B_SYS, and E_SYS. These templates are formatted to fit PromptTemplate objects, which organize input for llama-based models. For "mistral" type, a different template format (B_INST, E_INST) is employed. If no specific type is provided, a general prompt template is generated, which may include historical context. The function also manages a ConversationBufferMemory to retain user interactions (history). In summary, get_prompt_template facilitates the structured generation of prompts critical for optimizing llama model outputs based on various model requirements and user interactions.

Chapter 10

Results

1 Overview

The final model of ChatIITK integrates several key components to ensure robust performance and user-friendly interaction. At its core, the system employs a pre-trained language model from the Hugging Face library, which is fine-tuned for specific tasks. This model is enhanced by a retrieval system that fetches relevant documents from a pre-built vector database (ChromaDB), ensuring that the chatbot's responses are both informed and precise.

2 Model Components

1. **Embeddings**: The model uses Hugging Face's instruction-based embeddings to convert text into high-dimensional vectors. These embeddings capture the semantic meaning of the text, making it easier to retrieve relevant information.

2. **Vector Store (ChromaDB)**: Before inferencing, the data is ingested into ChromaDB, a vector database that stores the embeddings. This database enables fast and accurate retrieval of information based on the user's query.

3. **LLM (Hugging Face Pipeline)**: The core of the chatbot is a language model from Hugging Face, which can be either quantized or full, depending on the use case. The model is loaded with specific configurations to optimize performance on the available hardware (CPU, GPU, or MPS).

4. **RetrievalQA**: This is a retrieval-based question-answering system from the LangChain library. It uses the embeddings to find relevant documents in the vector store and generates responses based on these documents. The system can also handle chat history, enhancing the conversational experience.

5. **Prompt Template**: The chatbot uses a prompt template to structure the questions and responses. This template includes the context, history, and the user's question, ensuring that the responses are coherent and contextually appropriate.

6. **Callback Manager**: For real-time interaction, the model uses a callback manager with streaming output handlers, providing immediate feedback to the user.

3 Streamlit Interface

The Streamlit interface serves as the front end for ChatIITK, providing a simple yet powerful platform for users to interact with the chatbot. Streamlit is chosen for its ease of use and capability to create dynamic web applications quickly.

3.A Key Features

1. **User Input**: Users can input their queries directly into a text box. The system processes these queries and generates responses in real-time.

2. **Conversation History**: The interface maintains a history of the conversation, allowing users to refer back to previous interactions. This is particularly useful for ongoing tasks or complex queries that require multiple steps.

3. **Document Similarity Search**: An expander section provides insights into the source documents used to generate responses. This transparency helps users understand the basis of the chatbot's answers and verify the information.

4. **Device Optimization**: The Streamlit app dynamically selects the best available compute device (CPU, GPU, or MPS) to ensure smooth performance.

5. **About Section**: The sidebar includes information about the project and its developers, creating a more engaging and informative user experience.

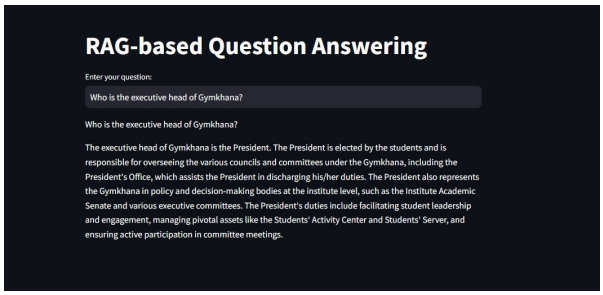


Figure 10.1: the streamlit interface

4 Sample Queries and Answers

1. Tell me something about Manindra Agarwal?

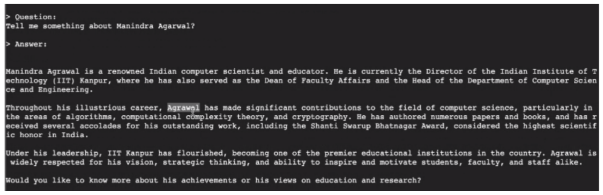


Figure 10.2: Question 1

2. According to you, what are the scopes of imporvement in the constittion of the student gymkhana?

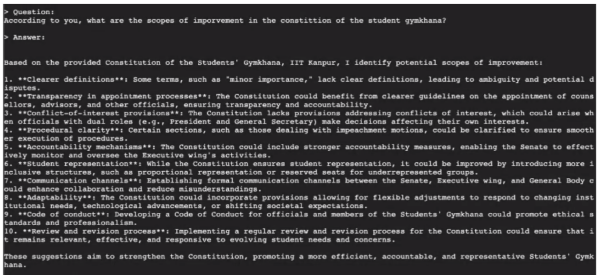


Figure 10.3: Question 2

3. When can Senate Remove a General Secretary?

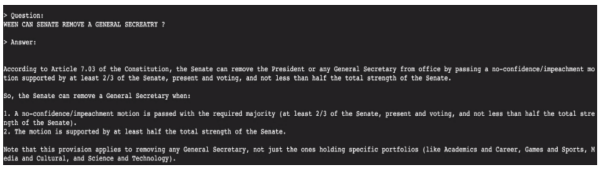


Figure 10.4: Question 3

4. Why were the VH staff fired?

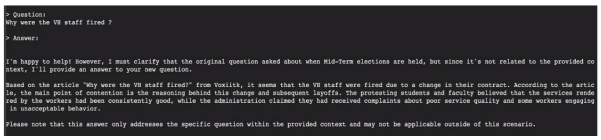


Figure 10.5: Question 4

5. What are the consequences for a student if they are found to have provided false information on their resume?

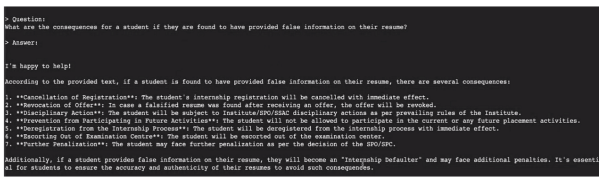


Figure 10.6: Question 5

Chapter 11

Observations and References

1 Observations

The primary challenges encountered during the development of the RAG Chatbot included selecting the appropriate models and optimizing their performance. Initially, methods such as Naive Bayes, Decision Trees, and Neural Networks were utilized. However, these approaches exhibited limitations in handling complex language patterns and maintaining context understanding. After extensive experimentation, Transformers were identified as the most effective model due to their superior accuracy and scalability with large datasets.

Issues related to compatibility and dependencies were common. For example, ensuring Torch's compatibility with CUDA involved verifying CUDA versions and installing corresponding PyTorch versions. Dependency-related errors, such as ModuleNotFoundError, required meticulous management of packages listed in requirements.txt. In some cases, missing or corrupted model files led to OSError, necessitating verification of model paths and file integrity. Network-related HTTPErrors during model downloads were mitigated by manual downloads from stable connections. Memory constraints, resulting in MemoryErrors, were addressed by reducing batch sizes or utilizing machines with higher memory capacities.

The incorporation of Transformers significantly enhanced the chatbot's performance, offering improved context comprehension and accurate responses. Addressing these technical challenges ensured a smoother implementation process, ultimately leading to a more robust and reliable RAG Chatbot.

2 References

We referred to the following links of some research papers, youtube links and articles for our overall work

1. https://arxiv.org/abs/1706.03762
2. https://jalammar.github.io/illustrated-transformer/
3. https://neptune.ai/blog/vectorization-techniques-in-nlp-guide
4. https://HuggingFace.co/
5. https://youtu.be/eMlx5fFNoYc
6. https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501
7. https://arxiv.org/pdf/1910.02054