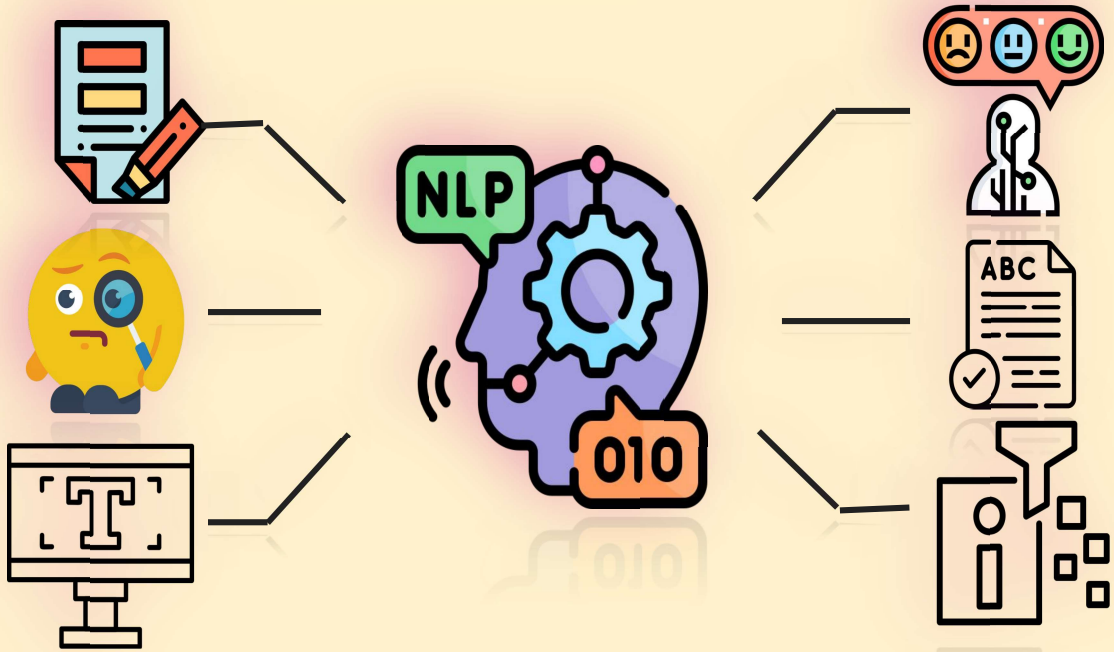
Natural Language Processing Applications



Presented by:

hu Mishra, GHRCE, Nagpur

Bankar, GHRCE, Nagpur

Guide:-

Dr. Mangala Madankar

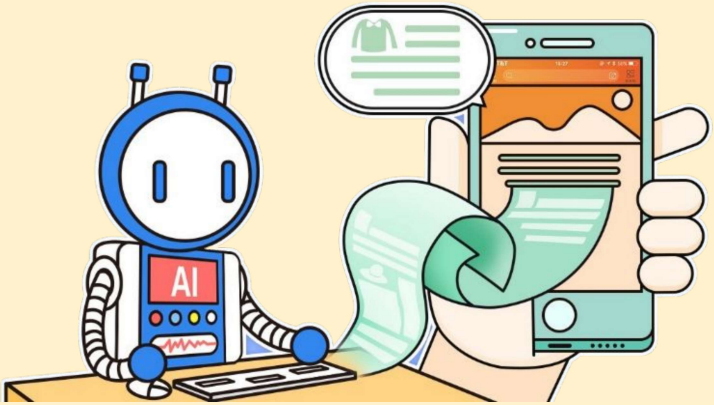
HOD of the Department of AI, G.H. Raisoni College of Engineering, Nagpur

Index

Index

Introduction

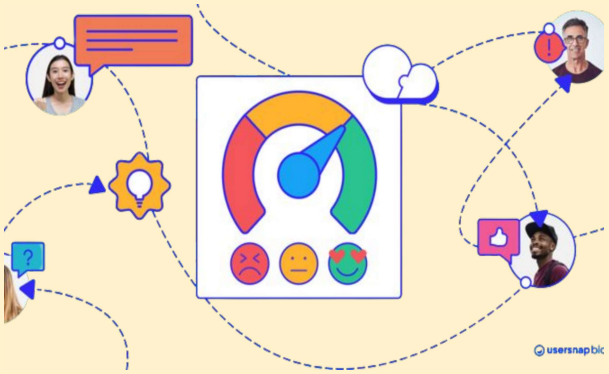
Textual data still is the major source of data share in today's world. The data need to be saturated and be precise in it's vastness. Working with huge raw textual data is a time consuming task. To deal with the situation we can build a structure "Text Processing Hub" that will simplify the applications of NLP on a single platform for the user to process their textual data quickly and note efficiently.



"Text Processing Hub", our project is to provide a easy way to interact with text data. Whether it's understanding sentiments, extracting insights, generating human-like responses, or unlocking the hidden patterns within vast volumes of text, our versatile NLP-powered platform is designed to excel. We've merged state-of-the-art algorithms with user-friendly design to make the marvels of NLP accessible to all.

Abstract

Textual data plays a pivotal role in information sharing, analysis, and communication, "Text Processing Hub" emerges as a comprehensive and user-friendly platform designed to address various natural language processing (NLP) tasks. This innovative web application harnesses the power of NLP models to offer six distinct functionalities within a single interface.



The Text Processing Hub facilitates “Text Summarization”, “Text Generation”, “Emotion Detection”, “Sentiment Analysis”, “Spelling Correction”, and “Information Extraction”. Users are empowered to interact with textual data more effectively, leveraging the capabilities of advanced NLP models and algorithms.

The project's core components include a well-structured web interface built using modern web development frameworks, a robust backend that handles user input and manages the integration of NLP models, and a set of NLP techniques tailored for specific tasks.

Objectives

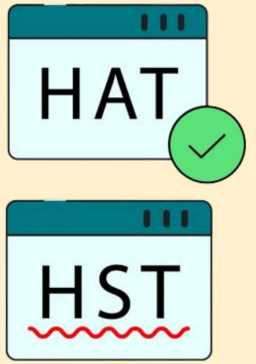
The primary objectives of "Text Processing Hub" revolve around leveraging the capabilities of Natural Language Processing to enhance communication, foster understanding, drive innovation and aid the user with the flexibility to perform different NLP operations over a single platform easily.

This project seeks to achieve the following key goals:

- **Tool Accessibility and Usability**
- **Sentiment Analysis Precision**
- **Contextual Text Generation**
- **Structured Information Extraction**
- **Correction in the textual data**
- **Summarization of the textual data**

By achieving these objectives, Text Processing Hub aims to revolutionize how we interact with text, unlocking new dimensions of communication, analysis, and ugh the transformative capabilities of NLP.

Literature Survey



In the realm of natural language processing (NLP), several key developments and prior research initiatives have laid the groundwork for the "Text Processing Hub" project. This literature survey explores the relevant studies and projects that have influenced the development of this comprehensive platform.

- NLP Advancements: Modern NLP builds on the foundational work of researchers like Jurafsky and Martin, revolutionizing text processing for diverse applications.
- Text Summarization: Innovations in text summarization, including TextRank and BERT-based models, enable efficient distillation of lengthy texts into concise summaries.
- Text Generation Breakthroughs: Text generation has made significant strides, thanks to technologies like recurrent neural networks (RNNs) and transformers, as seen in Vaswani et al.'s work on "Attention Is All You Need."
- Emotion Detection and Sentiment Analysis: Emotion detection, inspired by Ekman and s research on facial expressions, and sentiment analysis, as outlined by Pang 's work, provide valuable insights into human expressions and public opinion text

Literature Survey

- Spelling Correction: The development of spelling correction tools has deep roots in both rule-based systems and probabilistic methods. Notable works include Norvig's "How to Write a Spelling Corrector," which introduced the idea of edit distance. The "Text Processing Hub" project incorporates these principles to offer efficient and effective spelling correction capabilities to its users.

- Information Extraction: Information extraction techniques have evolved considerably, driven by the need to mine structured data from unstructured text. Researchers like Banko and Etzioni's "The Tradeoffs Between Open and Traditional Relation Extraction" have made significant contributions. The "Text Processing Hub" project embraces these advancements, allowing users to extract structured information from text data efficiently.

- Web-Based NLP Tools: The idea of hosting NLP tools on web platforms has e increasingly popular. Notable projects such as spaCy, NLTK, and the rd NLP Group's suite of tools have paved the way for accessible and iendly NLP applications.

System Architecture

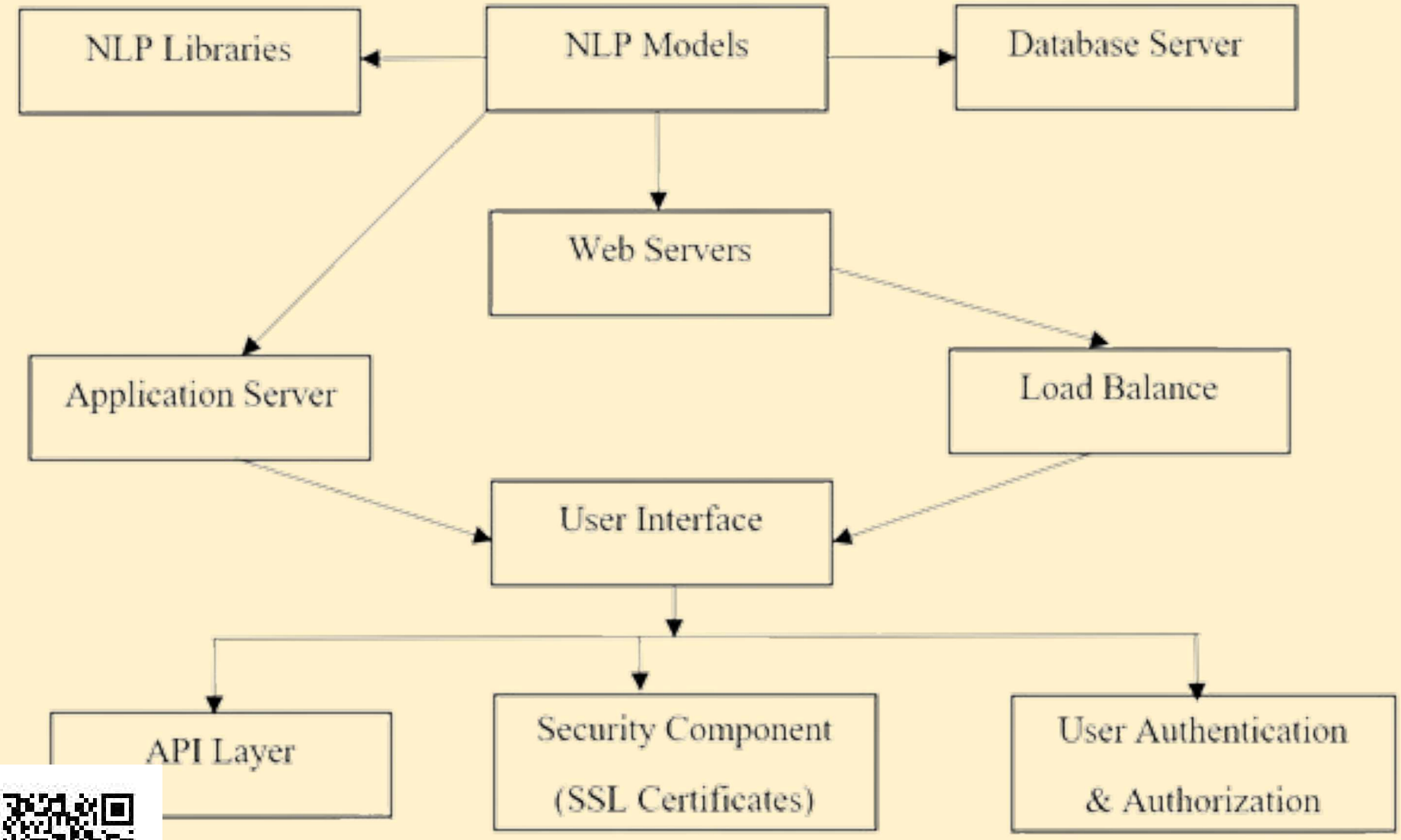


Figure 1

Data Flow of the Project

The flow of data in text processing hub from one point to another, which involves the processes, paths, and transformations that data undergo in text processing hub is shown in the following diagram.

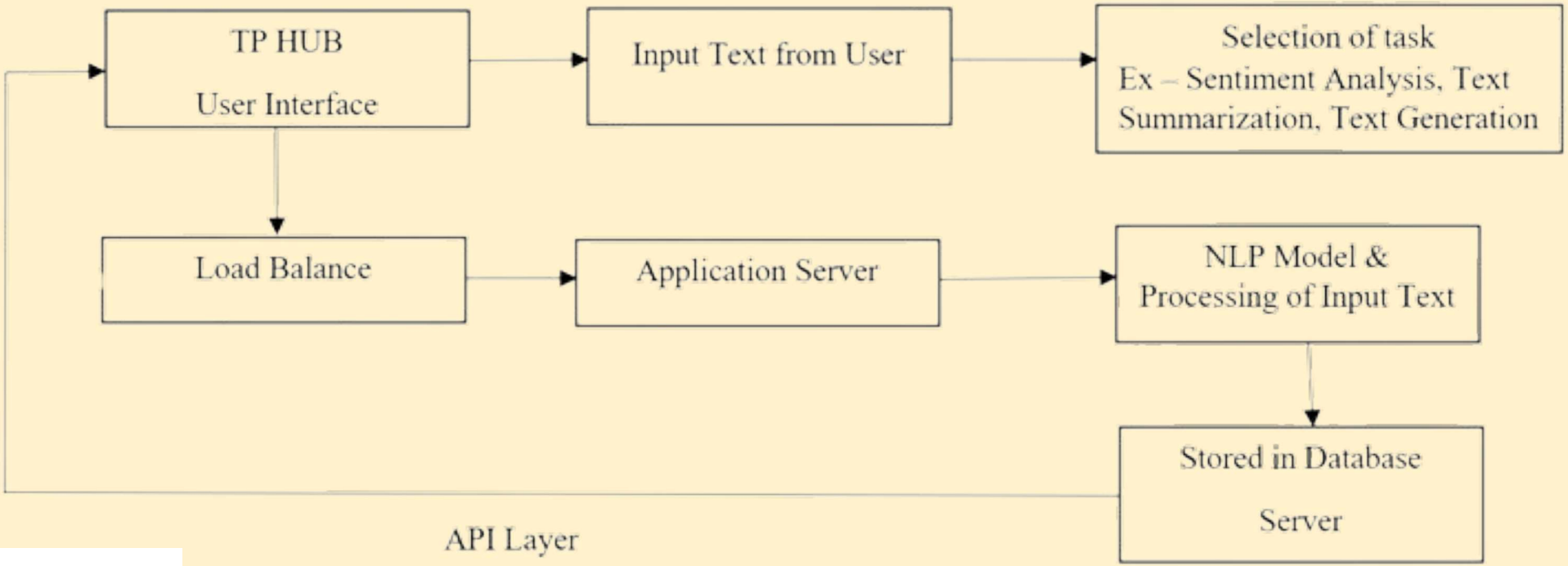


Figure 2

Hardware / Software Specification

Hardware Specifications:

- Server Infrastructure: The "Text Processing Hub" project requires a robust server infrastructure to host the web application.
- Processor: A multi-core processor, such as an Intel Xeon or AMD EPYC
- Memory (RAM): A minimum of 16GB RAM is recommended
- Storage: Adequate storage capacity, in the form of SSDs or HDDs
- Network Connectivity: High-speed internet connectivity is essential
- GPU (Graphics Processing Unit): A dedicated GPU, such as an NVIDIA GeForce or Tesla series GPU

Hardware / Software Specification

Software Specifications:

- Operating System: Window-11 or Ubuntu OS to provide stability and security.
- Web Server: Apache is used to serve the web application to users.
- Database Management System: RDBMS like PostgreSQL, MySQL are used.
- Programming Languages: Python is the primary language for NLP tasks. Web development done using languages such as JavaScript, HTML, and CSS.
- NLP Frameworks and Libraries: Integration of NLP libraries like spaCy, NLTK, Gensim, and NLP frameworks like TensorFlow or PyTorch are required.
- Web Application Framework: A web framework, Flask (Python-based), is used for building the user interface and managing user interactions.
- Version Control: Version control systems like Git are used to manage the codebase and collaborate with other developers.
- Security Tools: Security tools, including SSL certificates (HTTPS), firewalls, and intrusion detection systems (IDS), are used to protect user data and ensure a secure environment.
- User Authentication: Google authentication is used as user authentication
- ment and Containerization: Containerization tools like Docker and container ration platforms like Kubernetes facilitate deployment and scaling of the

Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a natural language processing technique that identifies the sentimental tone behind the text, such as sentence, paragraph, or the entire document. It is used to determine whether the text is expressing the positive, negative or neutral feelings. Sentiment analysis is done to understand the feelings of an individual behind the text. It basically gains the insights into people's opinions, attitudes, and emotions regarding the topic or event. It can applied to wide range of text sources such as social media posts, customers reviews, news articles, comments etc.

Sentiment analysis is implemented using various NLP techniques, including machine learning models like logistic regression, support vector machines(SVM), and deep learning models such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers-based models.

Sentiment analysis is a valuable NLP technique that helps to get the peoples opinions and the context behind the text. Following is one of the way of performing sentiment

Proposed Methodology/System Architecture

Sentiment Analysis:

Implementation of the sentiment analysis model is done using the Hugging Face Transformers library.

1. Importing Libraries

2. Data Preprocessing: a dataset from Kaggle and extracts it, dataset into a Pandas DataFrame and encodes the labels using LabelEncoder.The dataset is split into training, validation, and test sets.

3. Data Loading and Tokenization: The DataLoader class is defined to load and tokenize the text data using a pre-trained tokenizer from Hugging Face's Transformers. It returns the tokenized input data, attention masks, and labels.

4. Model Configuration:, AutoModelForSequenceClassification, is created with the configured architecture.

5 Training Configuration: A Trainer is instantiated with the model, training data, and g arguments.The final layer of the model typically applies a softmax nction to the output logits. Softmax converts the raw model output into ores for each sentiment class.

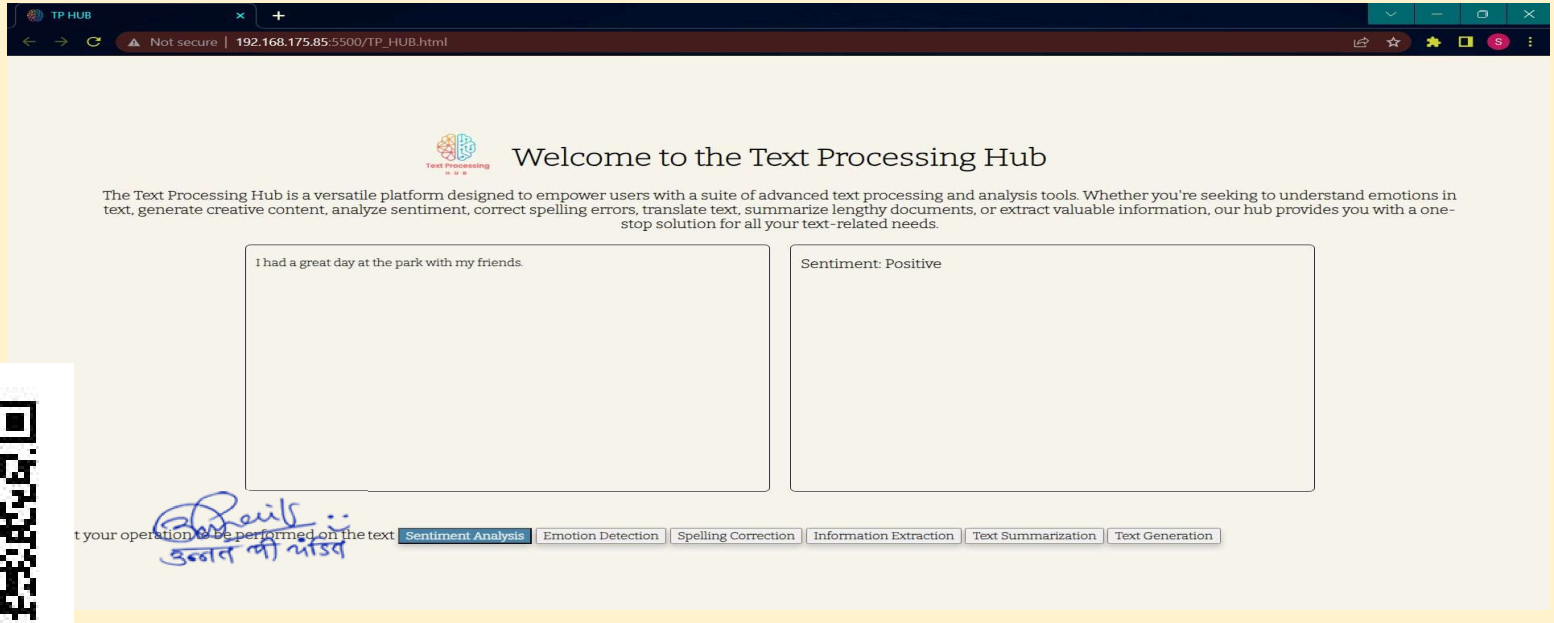
Model Outputs / Results :- (Prototype)

6. Model Training: The model is trained using the Trainer with the specified training arguments. Metrics like F1-score, accuracy, precision, and recall are computed during training.

7. Model Evaluation: The trained model is used to make predictions on the test dataset.Classification metrics are computed, such as classification report and accuracy.

8. Saving the Model: The trained model is saved to a specified directory.

9. Inference with a Saved Model: A class named SentimentModel is defined for inference with a saved model.



Text Summarization

Text summarization is the task of creating the summary of larger piece of text. It can be used to quickly get the essential information or the main ideas in the text by creating the shorter version of the larger text by still retaining it's essence. The goal of text summarization is to summarize the text that can be easily understood and have the key points of the original text.

The process of text summarization is divided into two categories:

- Extractive summarization - Works by selecting and combining existing sentences from the original text to form a summary. This type of summarization is possible with the help of techniques like TF-IDF, TextRank, or other machine learning models.
- Abstractive summarization - It aims to generate a summary in it's own words by paraphrasing and rephrasing the content, it has the ability to have deep understanding of text and generate human like language. Performing all this complex task was possible due to the sequence-to-sequence models like RNNs or transformers based models like BERT.

is the process of performing the text summarization.

Proposed Methodology/System Architecture

Text Summarization Model:

1. The model is fine-tuned on the SAMSum dataset.
2. The PEGASUS model is used for abstractive text summarization.
3. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is used as the evaluation metric, which measures the quality of summaries by comparing them to reference summaries.
4. The code leverages the Hugging Face Transformers library for working with transformer-based models.
5. GPU acceleration is used for model training and evaluation.
6. It downloads and preprocesses the SAMSum dataset for training and evaluation.

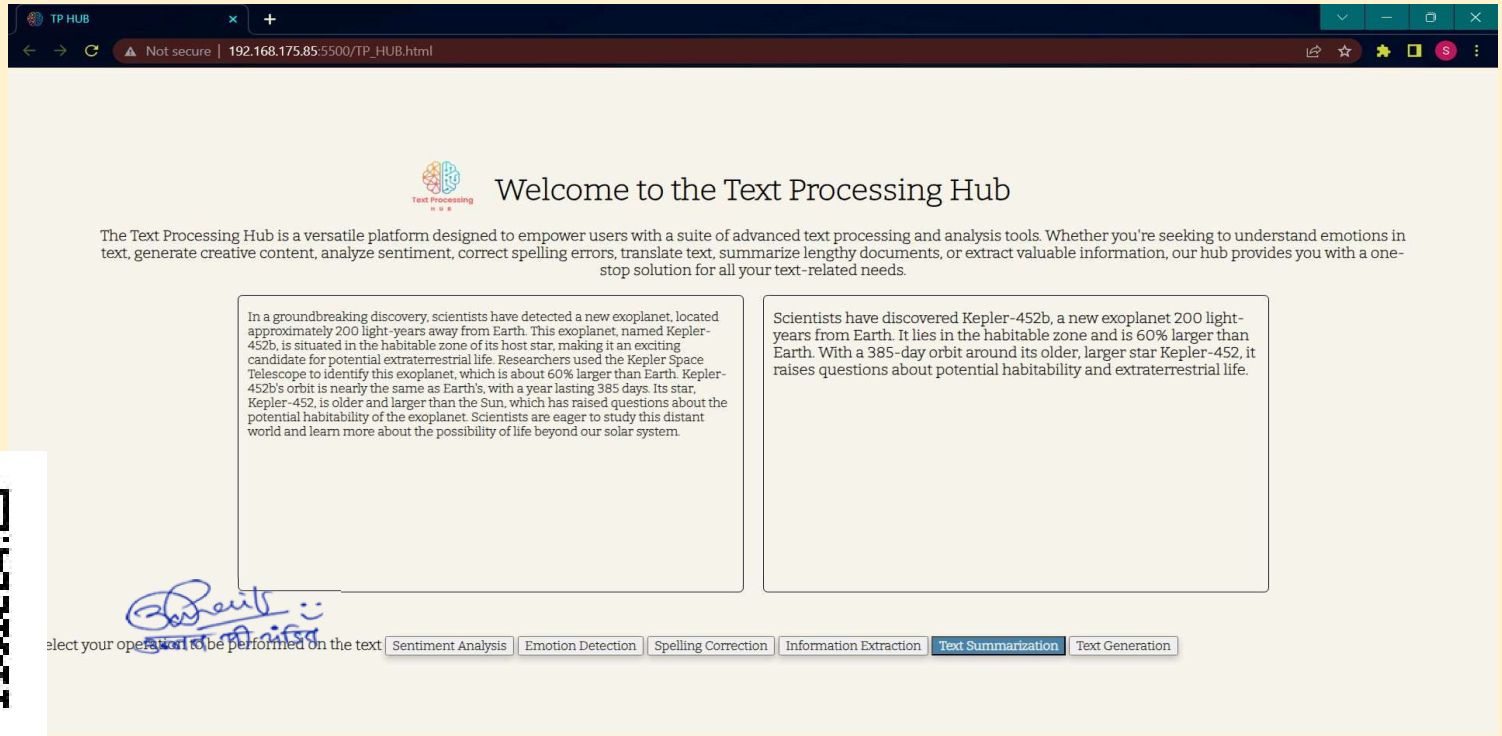
Model Training: The code sets up training for the PEGASUS model using the Hugging Face Transformers library.
Mathematical operations: Training of a sequence-to-sequence model with a particular configuration, loss minimization, model saving.

ation: This section evaluates the model's performance, specifically calculating es for summarization quality.
l operations: ROUGE metric calculations for text summarization quality.

Model Outputs / Results :- (Prototype)

```
In [1]: from transformers import pipeline, set_seed
        from datasets import load_dataset, load_from_disk
        import matplotlib.pyplot as plt
        from datasets import load_dataset
        import pandas as pd
        from datasets import load_dataset, load_metric

        from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

        import nltk
        from nltk.tokenize import sent_tokenize

        from tqdm import tqdm
        import torch

        nltk.download("punkt")

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Out[1]: True
```

```
Hannah: Hey, do you have Betty's number?
Amanda: Lemme check
Hannah: <file_gif>
Amanda: Sorry, can't find it.
Amanda: Ask Larry
Amanda: He called her last time we were at the park together
Hannah: I don't know him well
Hannah: <file_gif>
Amanda: Don't be shy, he's very nice
Hannah: If you say so..
Hannah: I'd rather you texted him
Amanda: Just text him
Hannah: Urgh.. Alright
Hannah: Bye
Amanda: Bye bye

Reference Summary:
Hannah needs Betty's number but Amanda doesn't have it. She needs to contact Larry.

Model Summary:
Amanda: Ask Larry Amanda: He called her last time we were at the park together .<n>Hannah: I'd rather you texted him .<n>Amand
a: Just text him .
```



Emotion Detection

Emotion detection is a natural language processing technique that involves analysing text and classifying the emotions behind the text such as happiness, sadness, anger, fear, and more. It is a subset of sentiment analysis as it helps to predict the unique emotion rather than just stating the text tone is positive, negative or neutral. It helps to understand the feelings of the author behind the text. The main goal of emotion detection is to determine the emotional responses or emotional tone of the text.

Emotion detection has two main approaches:

- Lexicon-based emotion detection works by using a dictionary of words and phrases that are associated with different emotions.
- Machine learning-based emotion detection works by training a model on a dataset and the model learns to identify the patterns in the text that are associated with different emotions.

Emotion detection is implemented by using machine learning or deep learning models such as SVM. deep neural networks or the pre trained models like BERT or GPT.

s one of the way of performing emotion detection.

Proposed Methodology/System Architecture

Emotion Detection Model:

The implementation of an Emotion Detection model is to detect textual expressions and determine the associated emotion, using a Convolutional Neural Network (CNN) and OpenCV. Here are some highlights and operations reflected in the implementation:

1. Data Preprocessing: The code processes a dataset of facial expressions, which are provided as pixel values in the "fer2013.csv" file.
It converts the pixel values into images and saves them in folders corresponding to different emotions (e.g., 'angry', 'happy') for training and testing.

2. Model Training: The code uses a CNN for emotion classification. The model architecture includes convolutional layers, max-pooling layers, dropout layers, and fully connected layers. It compiles the model with categorical cross-entropy loss and the Adam optimizer.
The model is trained on the training dataset (e.g., 'train/angry', 'train/happy').

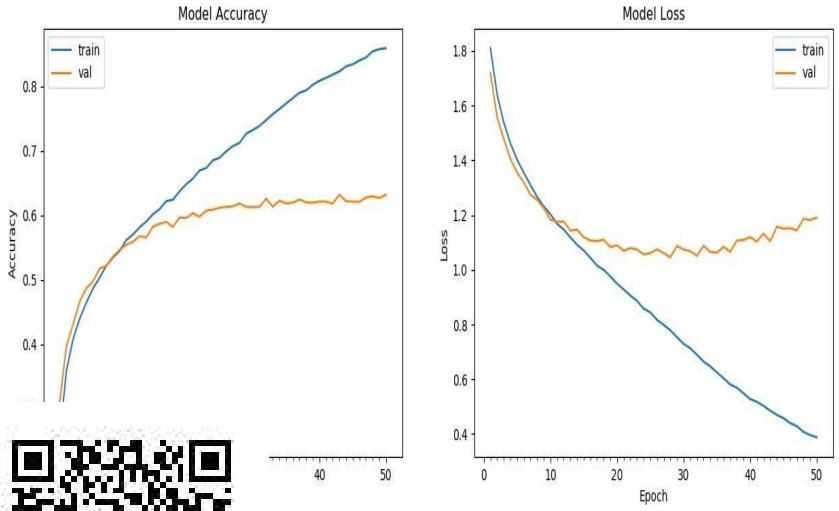
3. Displaying Emotions: The code defines a dictionary emotion_dict that maps emotion
r corresponding integer codes.
s the trained model to predict emotions and displays them on the website feed.

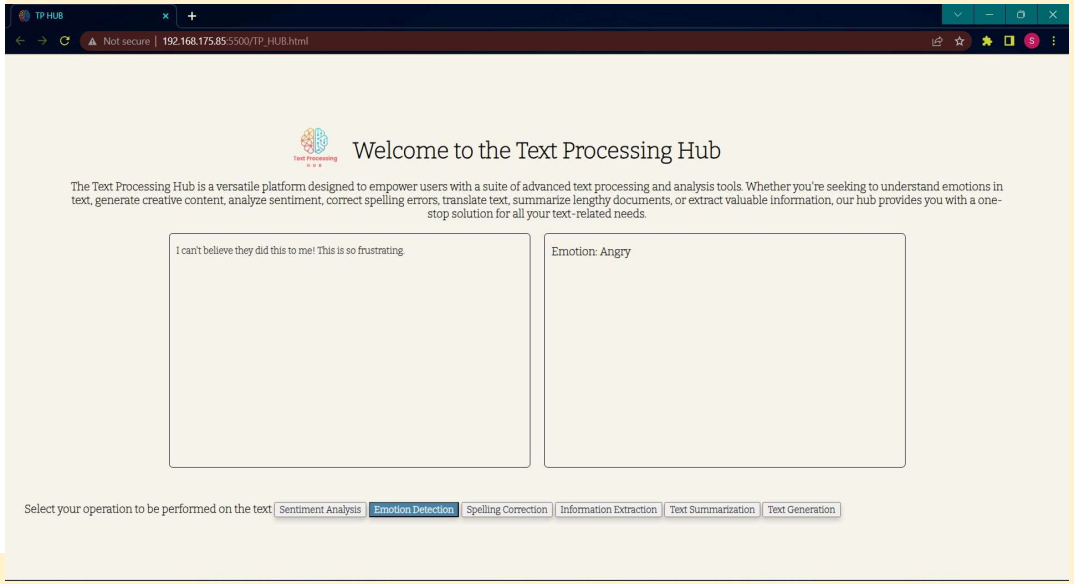
Model Outputs / Results :- (Prototype)

4. Dependencies: The code uses various libraries, including NumPy, pandas, PIL (Pillow), OpenCV, and TensorFlow (Keras) for model training and prediction.

5. Training Parameters: The model specifies training parameters, such as the number of training epochs, batch size, and learning rate.

6. Data Augmentation: Data augmentation techniques are not explicitly used in the code, but you can incorporate them to improve model generalization.





Text Generation

Text generation is the process of generating the new text just like human text based on the some inputs or prompts. It's a very critical task of natural language processing to perform it. It can be used for various purposes such as generating creative content, or the informative text.

The various models that can be used in text generation are:

- Recurrent neural network (RNNs) - Due to it's sequence-to-sequence task performing makes it popular to use. Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) are the popular RNN architecture.
- Transformers models such as GPT and BERT are excellent at capturing contextual relationships in text.
- Markov models or Hidden Markov Models (HMMs) or n-gram can be used for simple text generation.

Following is the process of performing the text generation.

Proposed Methodology/System Architecture

Text Generation Model:

The model is an implementation of a text generation model using a Recurrent Neural Network (RNN) with TensorFlow. Here are some highlights and operations reflected in the implementation:

1. Data Preparation: The code loads a text dataset from a URL (specifically, a Shakespearean text file) and preprocesses it.

2. Vocabulary Creation: It identifies unique characters in the text and creates a vocabulary. This vocabulary is used to map characters to numeric IDs and vice versa.

3. Sequence Processing: The text data is divided into sequences of a fixed length (seq_length), and the dataset is created from these sequences. Each sequence is split into input and target sequences for training.

4. Model Architecture: The code defines a custom RNN model (MyModel) using The model consists of an embedding layer, a GRU (Gated Recurrent Unit) dense layer. The model is designed to predict the next character in a sequence evious characters.

Proposed Methodology/System Architecture

5. Training: The model is trained using a custom training loop. The loss is calculated using the Sparse Categorical Cross-Entropy loss, and the model is optimized with the Adam optimizer.
Training is done in epochs, and model checkpoints are saved periodically.

6. One-Step Text Generation: The code defines a one-step model (OneStep) that allows generating text character by character. It uses the trained RNN model to predict the next character given an input sequence.

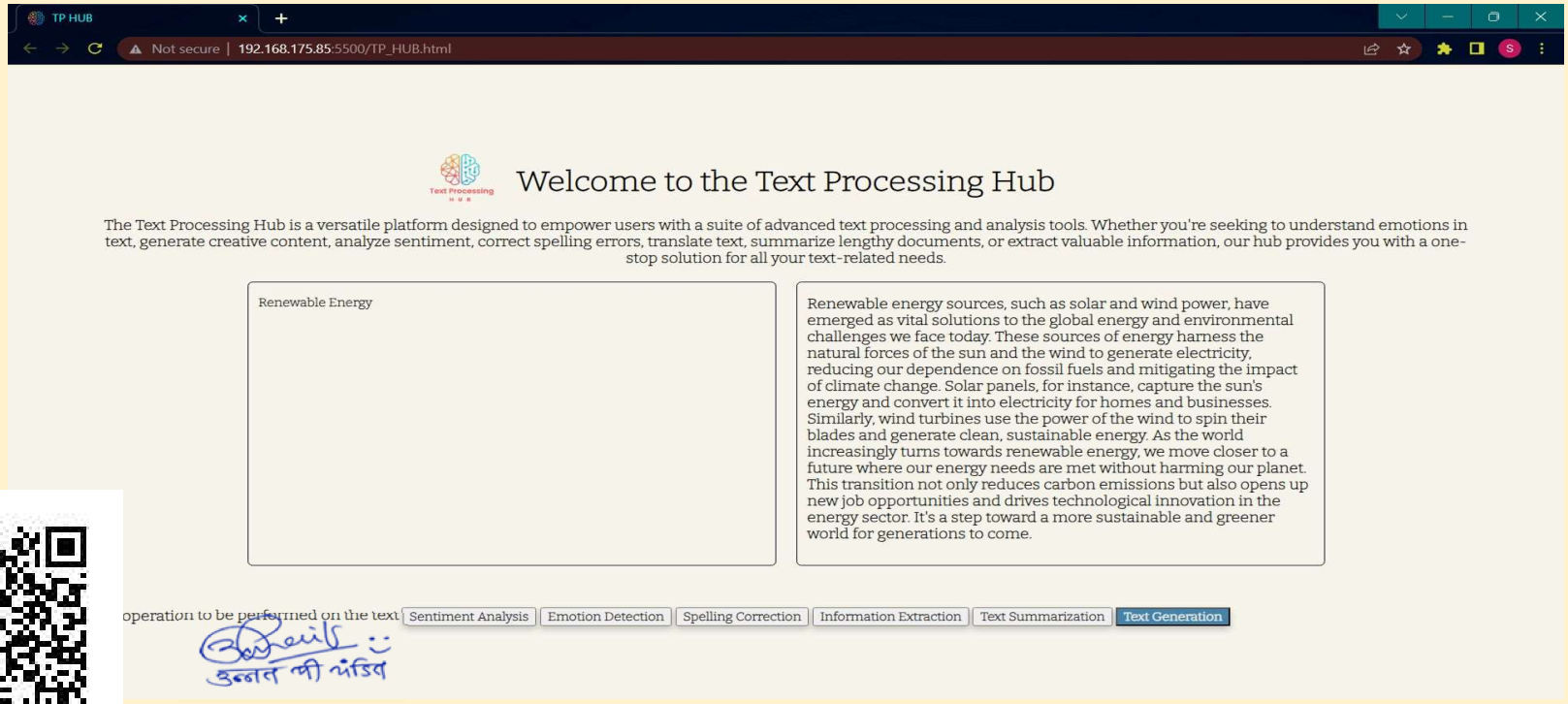
7.Training Customization: A custom training class (CustomTraining) is created, which inherits from MyModel. It overrides the train_step method to define a custom training step.

8. Training Loop: The model is trained using a loop where it goes through multiple epochs and batches. The training loop is designed to save checkpoints and print training progress.

Model Outputs / Results :- (Prototype)

```python
class CustomTraining(MyModel):
  @tf.function
  def train_step(self, inputs):
      inputs, labels = inputs
      with tf.GradientTape() as tape:
          predictions = self(inputs, training=True)
          loss = self.loss(labels, predictions)
      grads = tape.gradient(loss, model.trainable_variables)
      self.optimizer.apply_gradients(zip(grads, model.trainable_variables))

      return {'loss': loss}
```

```python
class MyModel(tf.keras.Model):
  def __init__(self, vocab_size, embedding_dim, rnn_units):
    super().__init__(self)
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(rnn_units,
                                   return_sequences=True,
                                   return_state=True)
    self.dense = tf.keras.layers.Dense(vocab_size)

  def call(self, inputs, states=None, return_state=False, training=False):
    x = inputs
    x = self.embedding(x, training=training)
    if states is None:
      states = self.gru.get_initial_state(x)
    x, states = self.gru(x, initial_state=states, training=training)
    x = self.dense(x, training=training)

    if return_state:
      return x, states
```



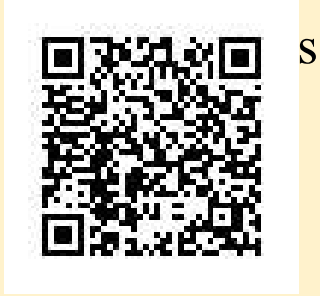
Spelling Correction

Spelling correction also known as spell checking is process of identifying and correcting the misspelled words or errors in text. Spell correction aim to improve the accuracy and readability of text by fixing errors and misspelled words. It is the essential component of text processing because if the spelling are incorrect or have the errors in them then it makes for the system or machine hard to understand it and hard to deal with the text. Misspelling can lead to drive to the wrong way of understanding and lack of communication, that's why it is necessary.

There are various models and methods used for spelling correction:

- Edit distance method - Measure the similarity between two words by calculating the minimum number of edit operations needed to transform one word into another.
- Language models - N-gram models and more modern neural language models like BERT and GPT-3 are used.
- Rule-Based Models - It rely on predefined spelling and grammar rules to correct misspelled words.
- Deep Learning Models - Transformers can be used for spell correction.

s the way of performing the spelling correction.



Proposed Methodology/System Architecture

Spelling Correction Model:

The model implements spelling correction using the SymSpell library. SymSpell is a Python library for efficient and accurate spell checking and correction.

1. Library Setup

2. Load Dictionary

3. Lookup Suggestions for Single-Word Input

4. Load Bigram Dictionary

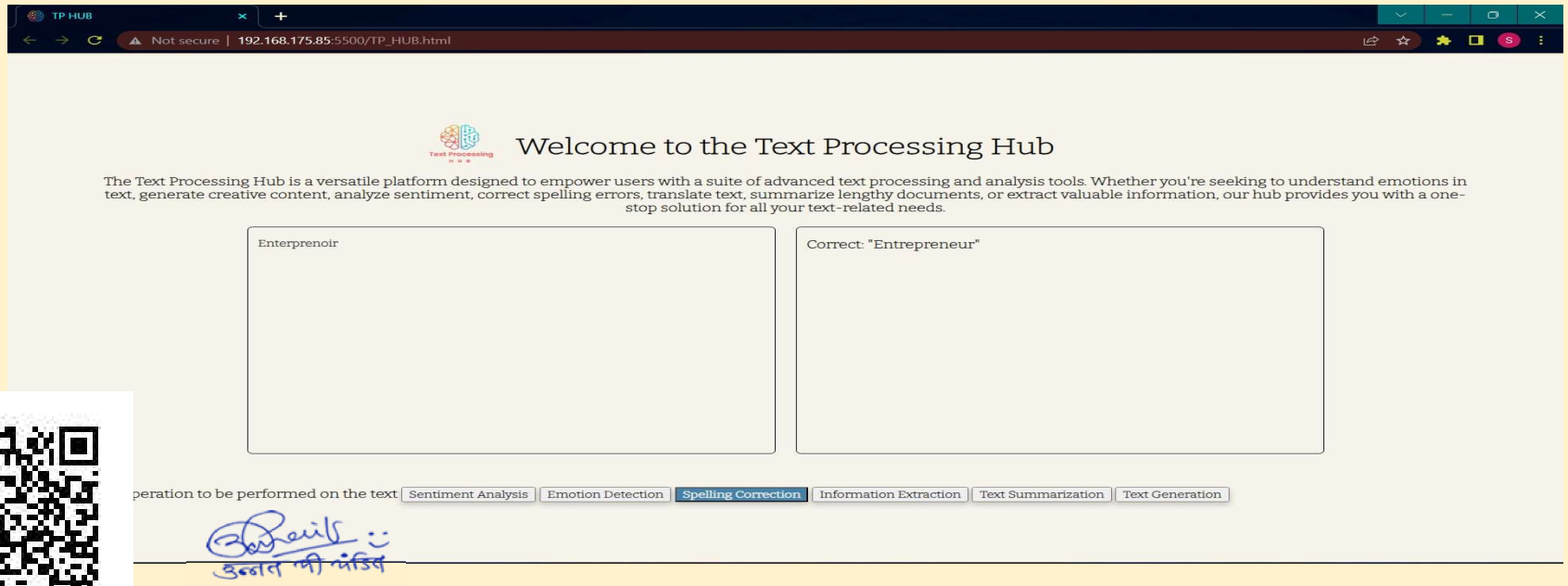
5. Lookup Suggestions for Multi-Word Input

nentation and Correction

Program

Model Outputs / Results :- (Prototype)

```
//lookup suggestions for single-word input strings
string inputTerm="house";
int maxEditDistanceLookup = 1; //max edit distance per lookup (maxEditDistanceLookup<=maxEditDistanceDictionary)
var suggestionVerbosity = SymSpell.Verbosity.Closest; //Top, Closest, All
var suggestions = symSpell.Lookup(inputTerm, suggestionVerbosity, maxEditDistanceLookup);

//display suggestions, edit distance and term frequency
foreach (var suggestion in suggestions)
{
  Console.WriteLine(suggestion.term +" "+ suggestion.distance.ToString() +" "+ suggestion.count.ToString("N0"));
}


//load bigram dictionary
string dictionaryPath= baseDirectory + "../../../../SymSpell/frequency_bigramdictionary_en_243_342.txt";
int termIndex = 0; //column of the term in the dictionary text file
int countIndex = 2; //column of the term frequency in the dictionary text file
if (!symSpell.LoadBigramDictionary(dictionaryPath, termIndex, countIndex))
{
  Console.WriteLine("File not found!");
  //press any key to exit program
  Console.ReadKey();
  return;
}
```



Information Extraction

Information extraction is a natural language processing task in which the important information ia being extracted from the text data. The aim of information extraction is to identify and extract the information from the text such as facts from the text or the entities like names of people, organizations, locations, relationship between the entities, and occurring of th events. Performing information extraction is necessary because it allows us to extract the valuable information from the data and this information can be used to improve the understanding of the world.

There are various models that can be used for information extraction:

- Information retrieval models such as TF-IDF (Term Frequency Inverse Document Frequency) and BM25.
- Name entity recognition (NER) models such as machine learning based NER models, Conditional Random Fields (CRF) and sequence-to-sequence models.
- Deep learning models such as RNNs, CNNs and transformers based models like BERT and GPT are the mostly used models for information extraction.

Following is one of the way of performing the information extraction.

Proposed Methodology/System Architecture

Information Extraction model :

The model performs information extraction and question answering using the spaCy library and pre-trained language models. Below are the key highlights and operations reflected in the implementation:

1. Library Setup: The code starts by checking the version of the NVIDIA CUDA compiler (nvcc) and installs/upgrades relevant libraries using pip.

2. Download Language Models: The script downloads and loads spaCy language models for text processing, including en_core_web_lg, en_core_web_sm, and en_core_web_trf.

3. Data Loading: The script loads data from a JSON Lines file named "cleaned_masdar.jsonl" using the jsonlines library. This data is expected to contain articles.

4. Text Processing with spaCy: The loaded data is processed using spaCy models. For instance, the "body" of each article is processed using spaCy's language models.
es and their labels are visualized using displacy.

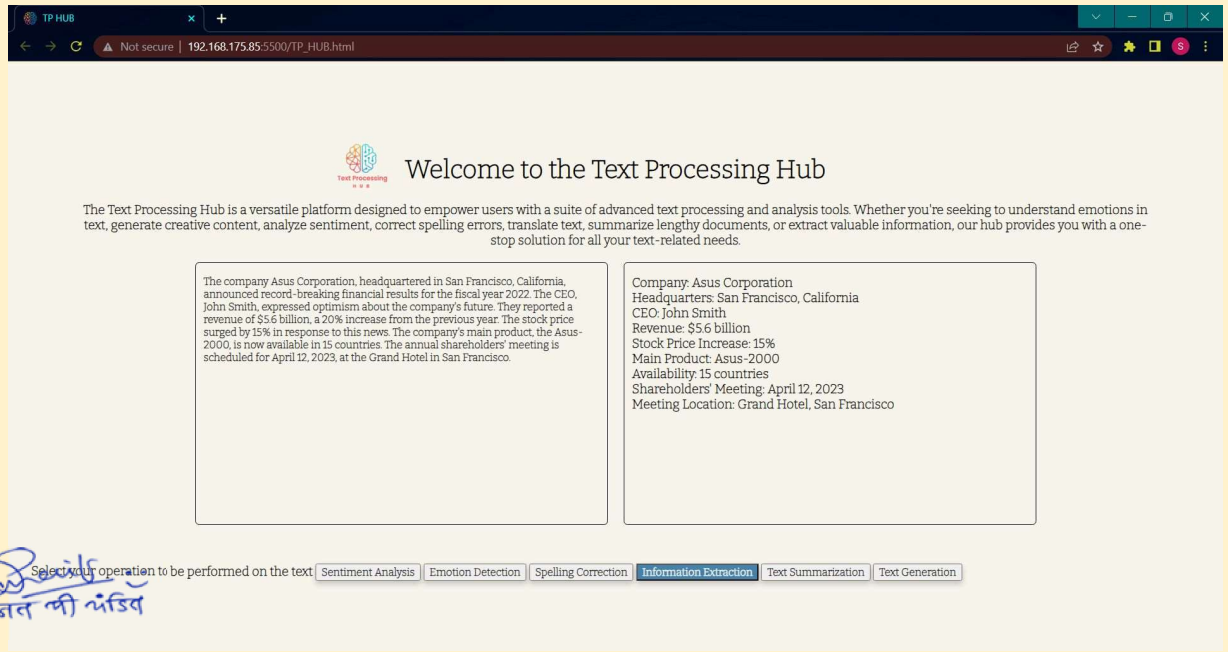
n Extraction: The script performs named entity recognition (NER) to extract (ORG) entities from the text.

Model Outputs / Results :- (Prototype)

6. Text Analysis: The code performs text analysis to find organizations involved in negotiations based on certain keywords such as "negotiat," "ceasefire," or "talks."

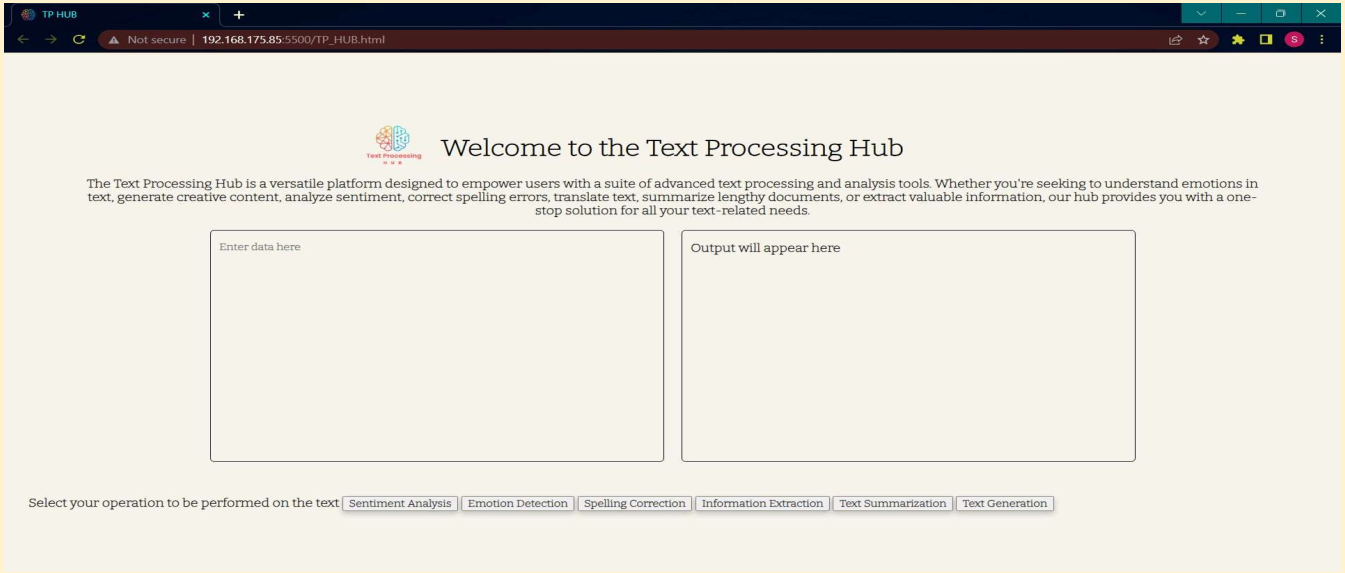
7. Dependency Parsing: Dependency parsing is used to extract the relationship between tokens. In one example, the code demonstrates how to find verbs related to a specific location ("Aleppo") and their modifiers.

8. Question Answering: The script utilizes the Hugging Face Transformers library to perform question answering. It uses a pre-trained model ("deepset/roberta-base-squad2") to answer a question related to a context (in this case, a sentence from an article).

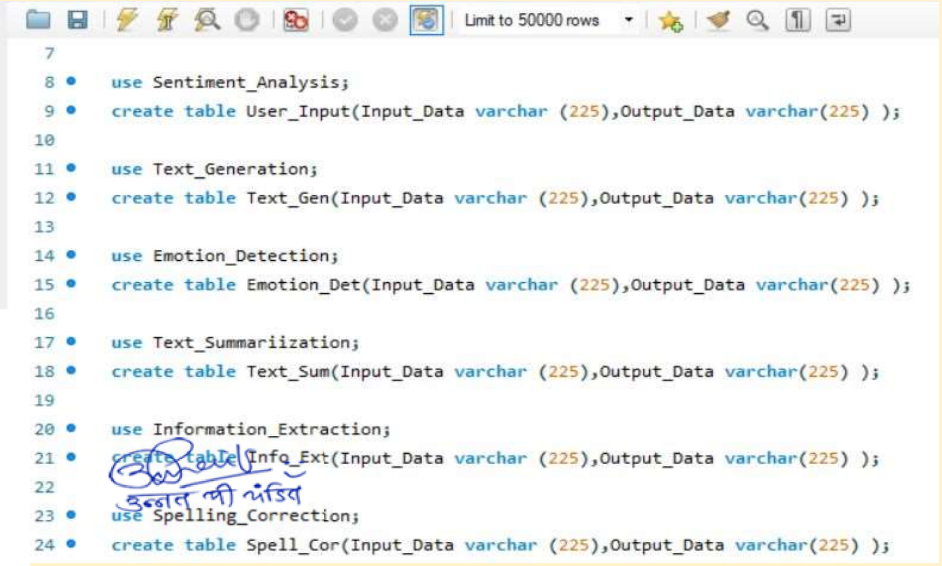


Model Outputs / Results :- (Prototype)

Interface of the front-end design i.e. website design:



Database design:



```
Limit to 50000 rows

7
8   use Sentiment_Analysis;
9   create table User_Input(Input_Data varchar (225),Output_Data varchar(225) );
10
11  use Text_Generation;
12  create table Text_Gen(Input_Data varchar (225),Output_Data varchar(225) );
13
14  use Emotion_Detection;
15  create table Emotion_Det(Input_Data varchar (225),Output_Data varchar(225) );
16
17  use Text_Summariization;
18  create table Text_Sum(Input_Data varchar (225),Output_Data varchar(225) );
19
20  use Information_Extraction;
21  create table Info_Ext(Input_Data varchar (225),Output_Data varchar(225) );
22
23  use Spelling_Correction;
24  create table Spell_Cor(Input_Data varchar (225),Output_Data varchar(225) );
```

Conclusion

In conclusion, the "Text Processing Hub" project represents a significant milestone in the realm of natural language processing and artificial intelligence. By amalgamating the power of advanced NLP models with a user-friendly web interface, this project has successfully unified six diverse NLP applications into a single, accessible platform.

Through the integration of “Text Summarization”, “Text generation”, “Emotion detection”, “Sentiment Analysis”, “Spelling Correction”, and “Information Extraction”, the "Text Processing Hub" empowers users to interact with textual data in ways that were previously arduous or inaccessible. The platform's commitment to education ensures that it serves the specific needs of students, particularly those in academic institutions, fostering a dynamic and supportive learning ecosystem.

References

https://www.cambridge.org/core/books/abs/sentiment-analysis/bibliography/F930F55FEF03F6D856579BB796B1B1FF

https://en.wikipedia.org/wiki/Sentiment_analysis

https://link.springer.com/article/10.1007/s10462-022-10144-1

https://www.topcoder.com/thrive/articles/text-summarization-in-nlp

https://medium.com/analytics-vidhya/text-summarization-using-nlp-3e85ad0c6349

https://towardsdatascience.com/text-summarization-with-nlp-textrank-vs-seq2seq-vs-bart-474943efeb09

https://www.analyticsvidhya.com/blog/2018/03/text-generation-using-python-nlp/

https://huggingface.co/tasks/text-generation

ithcode.com/task/text-generation