NEXT WORD PREDICTION

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the

Award of the degree of

BACHELOR OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Submitted By

Khushi Sharma(21EMCAD021)
Shirsty Singhal(21EMCAD033)
Vinod Yadav(21EMCAD059)
Sameer Kumar(21EMCAD052)
Devankit Sahu(21EMCAD012)

Under the Supervision of

Dr. J.R. Arun Kumar
Professor

BIKANER TECHNICAL UNIVERSITY, BIKANER

MODERN INSTITUTE OF TECHNOLOGY & RESEARCH CENTRE

ALWAR

APRIL, 2025

(2024-25)

BIKANER TECHNICAL UNIVERSITY, BIKANER

CERTIFICATE

This is to Certify that the project report entitled **“NEXT WORD PREDICTION”** is the original & genuine work of **“KHUSHI SHARMA, SHIRSTY SINGHAL, VINOD YADAV, SAMEER KUMAR, DEVANKIT SAHU”**, student(s) of B. Tech VIII Semester (Artificial Intelligence & Data Science Branch) who carried out the project work under my supervision & guidance.

SIGNATURE	**SIGNATURE**
Dr. J.R Arun Kumar	Dr. J.R Arun Kumar
HEAD OF THE DEPARTMENT	**PROFESSOR**
Dept. of CSE & AI	Dept. of CSE & AI

ACKNOWLEDGEMENT

Firstly, We express our sincere thanks to **Prof. Dr. S. K. Sharma (Director)** of the Modern Institute of Technology & Research Centre, Alwar, Rajasthan.

We pay our deep sense of gratitude to **Dr. J.R Arun Kumar**, Head of the Computer Science & Engineering Department of Modern Institute of Technology & Research Centre, Alwar for encouraging us to the highest peak and providing us the opportunity to present the Project.

We express our sincere & a deep sense of gratitude towards our guide **Dr. J.R Arun Kumar**, Professor of the Computer Science & Engineering Department of Modern Institute of Technology & Research Centre, Alwar for his invaluable guidance, suggestions, supervision, kind approval of the project, time to time counselling and advices throughout the work.

We would like to thank our **Institute** and **all Faculty Members**. Without them this project would have been a distant reality, we also extend our heartfelt thanks to our families and well-wishers.

Finally, but not least, **our parents** are also an important inspiration for us. With due respect, we also express our gratitude to them.

Khushi Sharma(21EMCAD021)
Shirsty Singhal(21EMCAD033)
Vinod Yadav(21EMCAD059)
Sameer Kumar(21EMCAD052)
Devankit Sahu(21EMCAD012)

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Next-word prediction is a pivotal task in natural language processing (NLP) with applications in predictive text, conversational agents, and intelligent writing systems. This project leverages deep learning techniques to develop a next-word prediction model that generates contextually relevant suggestions based on user input. Using a Long Short-Term Memory (LSTM) network, the model captures sequential dependencies and context from textual data, enabling accurate predictions in diverse linguistic settings. The dataset is preprocessed to include tokenization, padding, and vocabulary creation to prepare it for training. The LSTM-based architecture is designed to learn and generalize patterns in the data, optimizing for loss functions suited to sequence prediction. User interaction is incorporated through a dynamic interface, enabling real-time input and response generation. This project demonstrates the integration of deep learning with NLP for enhanced user experiences, achieving robust and context-aware predictive capabilities, and opens avenues for further enhancements in language modeling tasks.

CHAPTER 1

INTRODUCTION

1.1. What is next word prediction system?

Next-word prediction is a task in natural language processing (NLP) where a model predicts the most probable word or set of words that follow a given sequence of text. This technology forms the basis of many text generation and auto-complete systems. By analyzing the context of preceding words, the model can suggest appropriate continuations, enhancing efficiency and accuracy in text-based communication.

Traditionally, text prediction relied on rule-based approaches or statistical methods like n-grams. However, these methods are limited in their ability to understand long-term dependencies within text, as they only consider a fixed number of preceding words. With the rise of deep learning, more advanced models, such as Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) networks, have emerged as state-of-the-art solutions for sequence prediction tasks. These models excel at capturing context over longer sequences, making them ideal for next-word prediction systems.

Machine learning addresses the question of how to build computers that improve automatically through experience. Experience here can be obtained by training the machines using the model that has been built. By that training, the machine will learn through the patterns. As machine learning uses a new programming paradigm, then its concept is different from traditional programming. In traditional programming, the inputs are rules and data, then the output is the answer. While in machine learning, the inputs are answers and data, then the output is rules. This rule is what will be used later to build the model that can recognize certain types of patterns such as in deep learning.

Deep learning itself is the part of machine learning which uses different layers of neural networks that decide classification and prediction. Some examples of the use are classification include spam detection, cancer prediction, sentiment analysis, and so on. Then another example of it that we use in our daily life, that we may not realize is when we type.

Almost everyday, we use gadgets such as mobile and computer, then for sure we also use it for typing, whether it's just browsing on google or sending messages. During those activities, we may see that it recommends the next word based on what we type. Here is what is called next word prediction, because it predicts the next word that may come after our texts. It helped us to increase writing fluency and save time. Next word prediction (NWP) is an acute problem in the arena of natural language processing. It is also called Language Modelling and here it's about mining the text. Many programmers have used it and so have the researchers.

Some researchers who discussed it had written their findings as a journal and published it. Two of those journals are such as Next Words Prediction Using Recurrent Neural Networks by Saurabh Ambulgekar et al and Pre Training Federated Text Models for Next Word Prediction by Joel Streammel and Arjun Singh. Each researcher used their own models to make the prediction. So the results are different too between one to the other.

But the purpose is the same as to build models. They also focused on getting good accuracy as the greater the accuracy, the better the model will predict the next word. Even so, some models just can't get that good accuracy as there are some obstacles when buildingthe model, such as the limit on the data or the architecture model.

Based on the explanation above, the researcher here would like to make a model to predict the next word using LSTM. LSTM which stands for Long ShortTerm Memory is a variant of the recurrent neural network (RNN) architecture. The reason why the researcher used this model is because we thought it suits this case as it can have a longer memory of what words are important. The aim of creating this model is to predict the next word based on the input which the result should predict correctly.

1.2. Problem statement

Efficient communication systems are becoming increasingly essential in today's digital world, yet traditional text prediction methods often fail to meet modern demands. Rule-based or statistical models typically struggle to handle complex linguistic structures and fail to capture the nuances of context effectively. This lack of contextual understanding leads to inaccurate predictions, which can negatively impact user experience.

Moreover, ambiguity in language adds another layer of complexity, as words often have multiple meanings depending on their context. Real-time applications require predictive systems to not only be accurate but also computationally efficient, so they can function seamlessly across various devices. Additionally, the diversity in language styles, dialects, and domains poses challenges in building a model that can generalize well across different types of text input.

1.3. Problem objective

The objective of this project is to address these challenges by leveraging advanced deep learning techniques, particularly Long Short-Term Memory (LSTM) networks. By training on a large corpus of text, the model aims to provide highly accurate next-word predictions that are both contextually relevant and computationally efficient for real-time usage.

Objectives:

1. **Data Collection:**

   - Collect data from various sources like Kaggle, Wikipedia articles, Open source books, Twitter, WhatsApp, Facebook, etc.
   - Store the collected data in a text file which is referred to as a corpus.

2. **Data preprocessing:**

   - Cleaned the text by:
     - Removing punctuation, numbers, and non-ASCII character using regular expression.
     - Replacing multiple newlines with a single newline.
     - Removing unwanted single characters.
   - Perform tokenization using natural language processing.
     - Example: “I love programming!” to [“I”, “love”, “programming”, “!”]
   - Generated input sequences by:
     - Splitting each sentence into a list of tokens.
     - Creating all possible sub-sequence from each sequence.
   - Padded all sequence to have uniform length.

3. **Model Design:**

- Defined the input (x) as all tokens except the last word, and the output (y) as the last word.
- One-hot encoded the output labels using to_categorical.
- Split the dataset into training and test sets using train_test_split (80/20 split).
- Designed a Sequential LSTM model using TensorFlow/Keras:
    - Input layer with the shape of padded sequences.
    - Embedding layer to map word indices to dense vectors (captures semantic similarity).
    - LSTM layer with 256 units for capturing long-term dependencies.
    - Dense output layer with softmax activation to predict the next word from the vocabulary.

4. **Model Training:**

- Trained the model using the training set.
- Set training for 50 epochs (early stopping could be added optionally to prevent overfitting).

5. **Model Evaluation:**

- Evaluated the model on the test set after training using model.evaluate.
- Predicted the next word from a given seed text using the trained model.

6. **Deployment:**

- Intergrate the trained model into a streamlit application.
- Provide user-friendly interface for real-time next-word predictions.

CHAPTER 2

LITERATURE REVIEW

1. Introduction to Next Word Prediction

Next word prediction is a fundamental task in natural language processing (NLP) and computational linguistics, aiming to predict the most probable word that follows a given sequence of words. This task is essential in various applications such as text auto-completion, chatbot responses, and voice assistants. Next word prediction systems enable enhanced user interaction by suggesting possible word completions, thus improving typing efficiency, communication speed, and user experience.

2. NLP and Deep Learning

Natural language processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and human language. One of the significant advancements in NLP came with the introduction of deep learning techniques, which utilize neural networks to model complex linguistic patterns. Recurrent neural networks (RNNs), long short-term memory networks (LSTMs), and transformers have revolutionized NLP tasks, enabling accurate and context-aware predictions.

2.1. Recurrent Neural Networks (RNNs)

RNNs have been one of the primary models used in sequential data processing tasks, including next word prediction. An RNN processes data sequentially, maintaining hidden states that carry context from previous inputs. However, traditional RNNs struggle with long-range dependencies due to vanishing gradients.

2.2. Long Short-Term Memory (LSTM)

LSTMs, a type of RNN, address the limitations of traditional RNNs by using specialized gates that allow the network to remember and forget information,

making it better suited for long-range dependencies in text. LSTMs have been extensively used for sequence prediction tasks, including next word prediction, due to their ability to capture contextual meaning over extended sequences.

2.3. Transformers and BERT

The transformer architecture, introduced by Vaswani et al. in 2017, further advanced the field by replacing RNNs with self-attention mechanisms, allowing the model to process the entire sequence of words simultaneously. This architecture enables better parallelization and captures long-range dependencies more effectively. Transformer-based models like BERT (Bidirectional Encoder Representations from Transformers) have achieved state-of-the-art results in numerous NLP tasks, including next word prediction.

3. Next Word Prediction Models

Several models and techniques have been developed over the years for next word prediction, evolving with advancements in deep learning. Below are some notable models:

3.1. n-Gram Models

Before the deep learning revolution, n-gram models were the primary approach for next word prediction. These models use the previous "n" words to predict the next word based on statistical probabilities. While n-gram models are simple and efficient, they suffer from the issue of data sparsity and do not capture long-term dependencies in text.

3.2. Word2Vec and GloVe

Word embeddings like Word2Vec and GloVe (Global Vectors for Word Representation) improved the prediction of the next word by mapping words to dense vectors in a continuous vector space. These embeddings capture semantic relationships between words, making it easier to predict the next word by understanding the context and meaning of words in a sentence.

3.3. GPT-2 and GPT-3

OpenAI's Generative Pretrained Transformer (GPT) series represents a leap forward in next word prediction. GPT-2 and GPT-3, based on the transformer architecture, are pretrained on large corpora and fine-tuned for specific tasks. These models are capable of generating human-like text and can predict the next word with high accuracy. GPT-3, with its 175 billion parameters, is currently one of the most powerful language models and is used in a wide range of applications, including text completion and conversation generation.

3.4. BERT and Its Variants

BERT, a bidirectional transformer model, has set new benchmarks for understanding context in both directions (left-to-right and right-to-left). BERT's architecture enables it to capture richer word representations, improving next word prediction in many cases. Variants of BERT, like RoBERTa, and smaller models like DistilBERT, have been developed for more efficient training and deployment.

4. Applications of Next Word Prediction

Next word prediction has a wide range of applications across various domains:

4.1. Text Autocompletion

Autocompletion is one of the most common applications, seen in search engines, messaging apps, and email clients. By predicting the next word or phrase, the system speeds up typing and improves user experience.

4.2. Voice Assistants

Voice assistants like Siri, Alexa, and Google Assistant use next word prediction to generate responses to user queries. These systems often rely on deep learning models to interpret the user's spoken words and predict the most appropriate follow-up response.

4.3. Chatbots

Chatbots and virtual assistants in customer support or services use next word prediction to simulate human-like conversations. These systems need to understand context and provide appropriate responses, making next word prediction a key component.

4.4. Text Generation

Next word prediction is also used in text generation tasks, where models generate coherent and contextually appropriate text. For example, language models can write essays, stories, or articles by predicting subsequent words based on the initial input.

5. Challenges in Next Word Prediction

Despite advancements, several challenges remain in the next word prediction task:

5.1. Ambiguity in Language

Natural language is often ambiguous — the same word or phrase can have different meanings depending on the context in which it is used. For example, the word *"bank"* could refer to a **financial institution** or the **side of a river**. Next word prediction models need to learn contextual representations to accurately disambiguate meanings. Failing to resolve ambiguity can result in incorrect or irrelevant word suggestions that degrade the user experience.

5.2. Data Bias

The data used to train language models often reflects the biases present in society, such as gender, racial, cultural, or political bias. When a model learns from such data, it can unintentionally generate biased, offensive, or stereotypical content.

5.3. Computational Cost

Training deep learning models—particularly LSTMs, RNNs, or large transformer-based architectures like GPT—demands significant computational power (GPUs/TPUs), memory, and time. Even inference (making predictions) can be slow on larger models.

CHAPTER 3

SYSTEM DESIGN

Overview

The system design of the Next Word Prediction model focuses on providing a robust and scalable architecture for predicting the next word in a given text input. The system leverages state-of-the-art deep learning models, specifically LSTM or Transformer-based models, to learn the patterns and structure of language. It utilizes a client-server architecture with a web interface that allows users to interact with the system in real-time. The system is designed to handle a large number of concurrent users and provide quick and accurate predictions.

4.1 UML DIAGRAMS

A UML (Unified Modeling Language) diagram is a standardized graphical representation of a system's design that allows developers to visualize the structure and behavior of a system. UML diagrams are used in software engineering to depict software systems, business processes, and data structures through a set of symbols and notations. They help in planning, visualizing, specifying, constructing, and documenting the artifacts of a software system. UML includes several types of diagrams, such as:

3.1.1. Component Diagram

A component diagram is used to visualize the organization and relationships among software components within a system. It shows how various parts of the system, such as modules, packages, and services, interact to form a complete application. In the context of a next-word prediction system, the component diagram illustrates how different functional units like the frontend user interface, preprocessing module, machine learning engine (e.g., LSTM model), and backend APIs are structured and how they interact. This diagram helps developers and architects understand the modular breakdown of the system, enabling better maintainability and scalability.

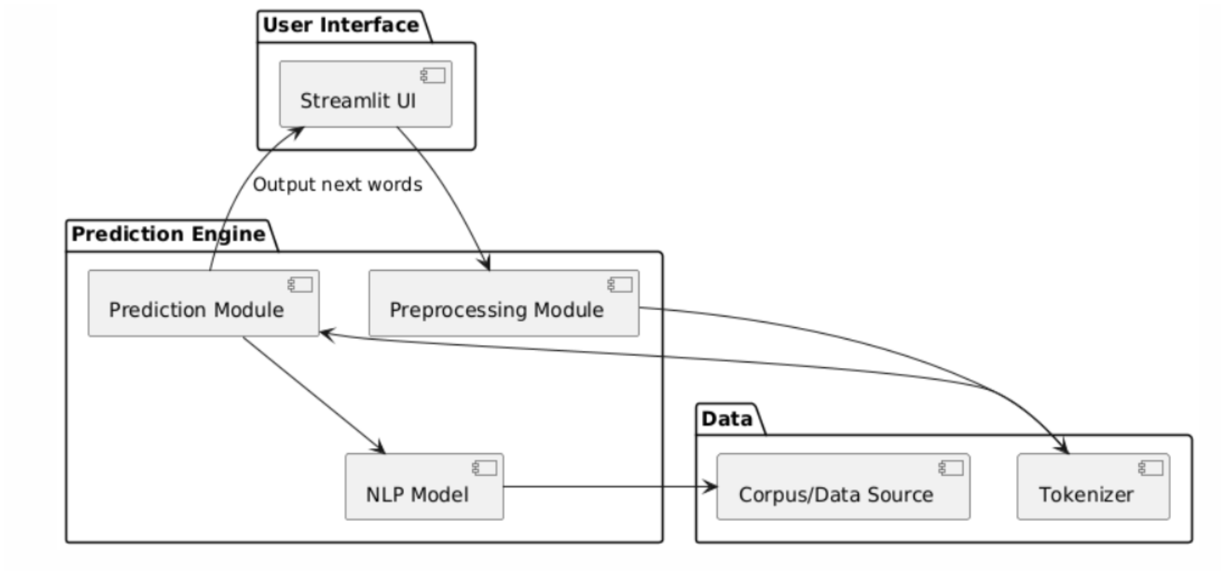


Figure 3.1: Component diagram

3.1.2. Deployment diagram

The deployment diagram describes the physical deployment of artifacts (software components) on nodes (hardware elements). It captures the configuration of runtime processing nodes and the components that live on them. In this project, the deployment diagram shows the placement of different parts of the application on various servers and devices. For example, it may depict how the Streamlit web application runs on a cloud platform like Heroku or AWS, while the LSTM model and preprocessing pipeline are executed on a dedicated backend server. This diagram is essential for understanding how the system is distributed across hardware infrastructure.

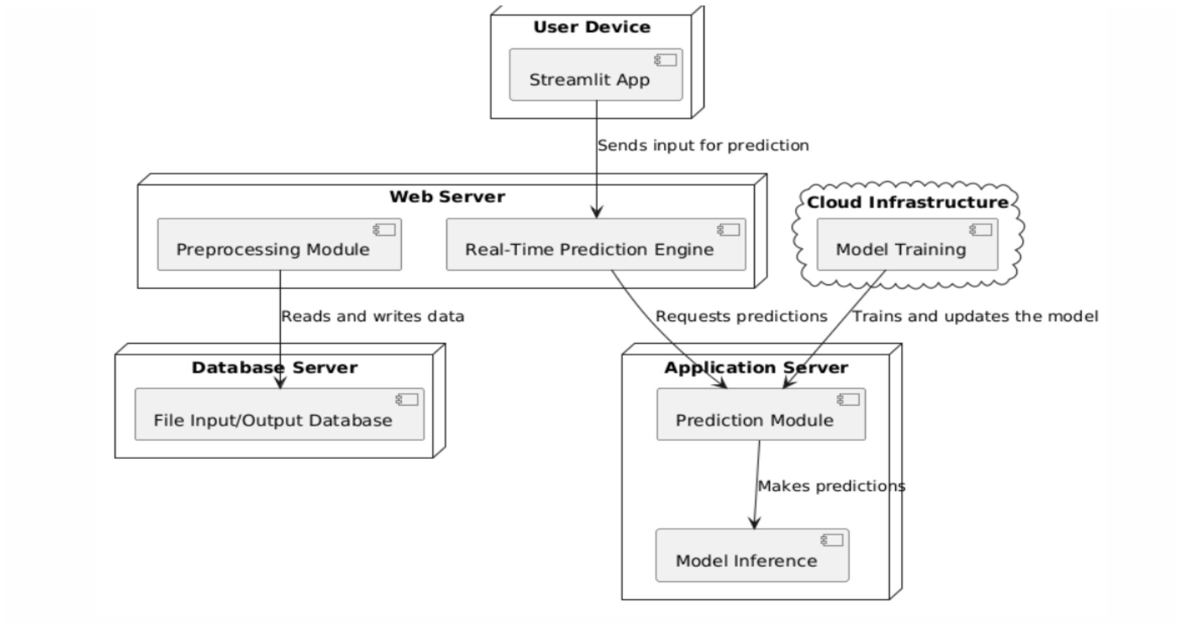


Figure 3.2: Deployment diagram

3.1.3. Object diagram

An object diagram provides a snapshot of the instances of classes in the system at a specific moment in time. It shows concrete instances, their attributes, and the relationships between them, making it useful for visualizing real data structures. In a next-word prediction system, the object diagram may represent instances such as a user input string object, a tokenized sequence object, and a model prediction object. This helps in understanding how the system behaves at runtime and how data flows between objects during execution.

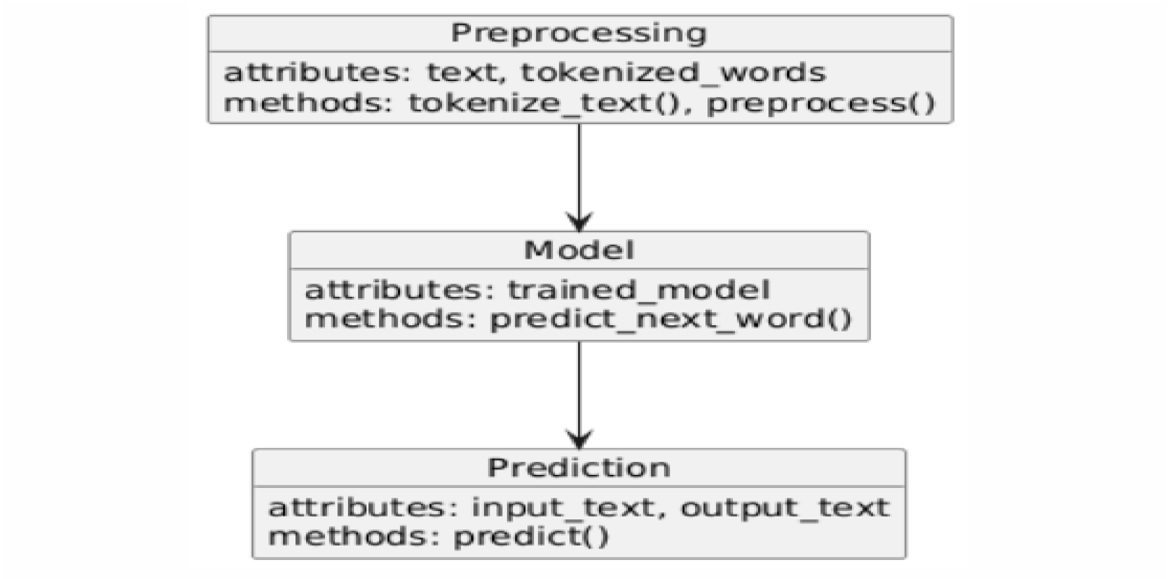

Preprocessing
attributes: text, tokenized_words
methods: tokenize_text(), preprocess()
Model
attributes: trained_model
methods: predict_next_word()
Prediction
attributes: input_text, output_text
methods: predict()

Figure 3.3: Object diagram

3.1.4. Communication diagram

A communication diagram focuses on the interaction between objects and the flow of messages exchanged to perform a specific task. It is particularly useful for visualizing how multiple components or classes communicate to fulfill a request. In this system, the communication diagram shows how the user interacts with the web interface, how the request is sent to the preprocessing module, how the tokenized input is passed to the LSTM model, and how the predicted word is returned and displayed. It emphasizes the sequence and structure of interactions among collaborating objects.

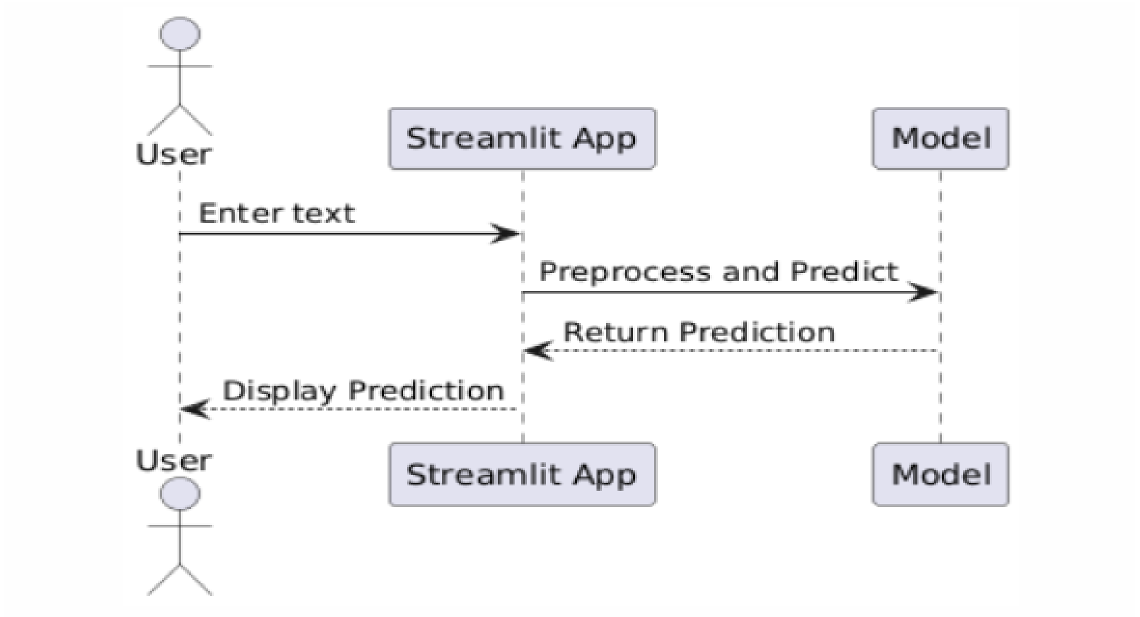


Figure 3.4: Communication diagram

3.1.5. State diagram

A state diagram models the dynamic behavior of a system by showing the different states an object can occupy and the transitions between those states. It is particularly helpful in systems that have well-defined states and conditional transitions. In this project, a state diagram could represent the state of the user input process—from the initial input state to preprocessing, to model prediction, and finally to result display. This provides insight into how the system responds to events and transitions between states during the workflow.

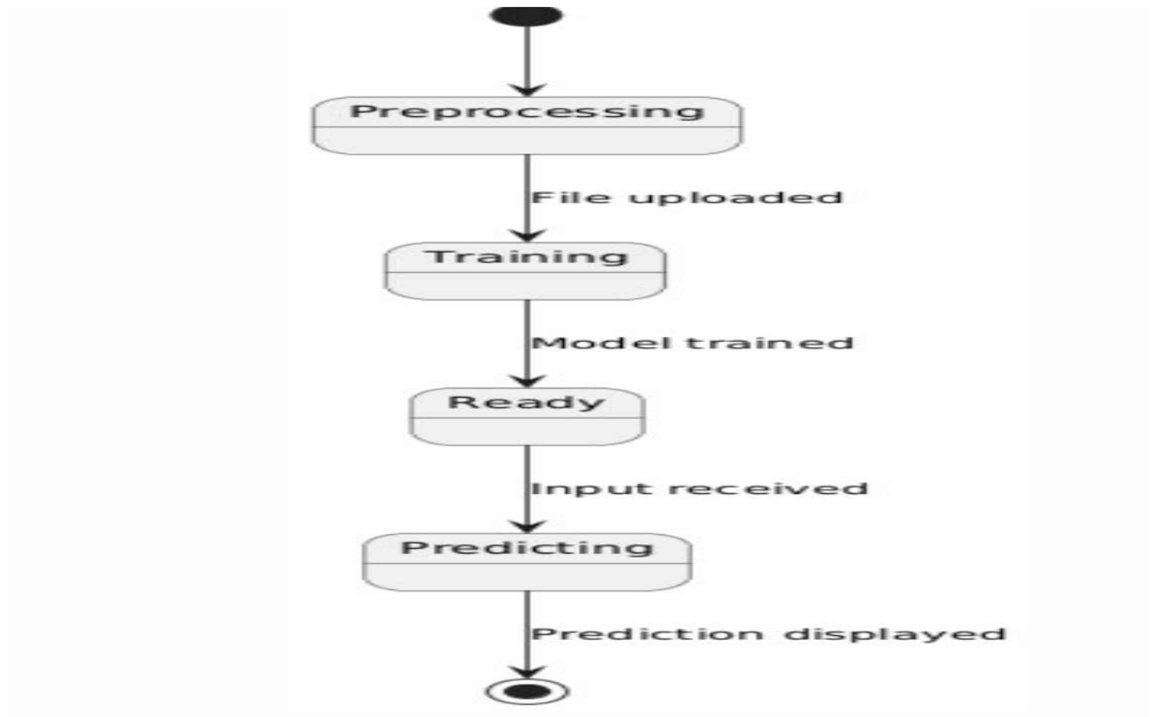


Figure 3.5: State diagram

3.1.6. Use case diagram

A use case diagram is used to capture the functional requirements of the system from the user's perspective. It shows the interactions between actors (users or other systems) and the system's use cases (functionalities). For a next-word prediction system, the use case diagram outlines how users interact with the application, such as entering text, receiving predictions, and possibly customizing the prediction settings. This diagram helps in identifying the system’s functionality and ensures that user requirements are well-understood and implemented.

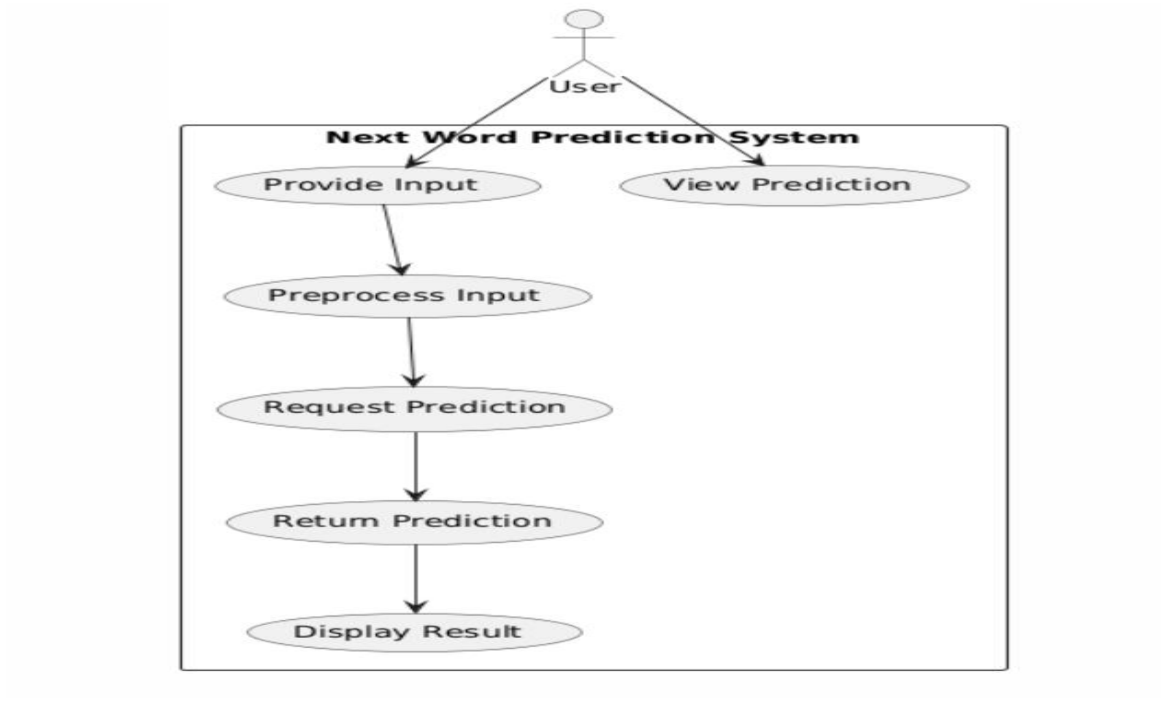


Figure 3.6: Use case diagram

3.1.7. Sequence diagram

A sequence diagram provides a detailed view of the order in which interactions occur between objects in a system over time. It shows the chronological sequence of messages exchanged and the lifelines of objects involved in the interaction. In this project, the sequence diagram illustrates the process beginning with a user's input, followed by text preprocessing, tokenization, LSTM model invocation, and finally the output generation. This diagram is crucial for understanding the temporal flow of data and operations, and for validating the logical sequence of function calls in the system.

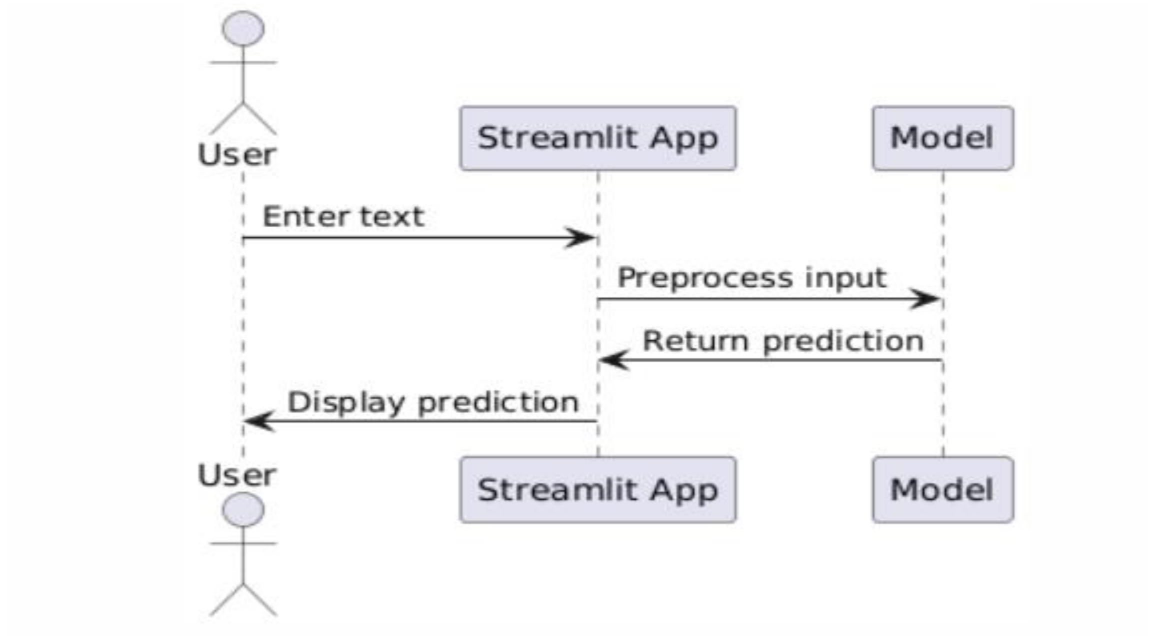


Figure 3.7: Sequence diagram

3.1.8. Activity diagram

An activity diagram represents the flow of control or workflow within a system. It models the sequence of actions and decision points involved in completing a process. For the next-word prediction system, the activity diagram captures the full lifecycle of a prediction request—from user input to preprocessing, model inference, and result display. It may include conditional branches for error handling, such as invalid input or model failure. This diagram is helpful in visualizing the operational logic and ensuring that all possible execution paths are well-defined.

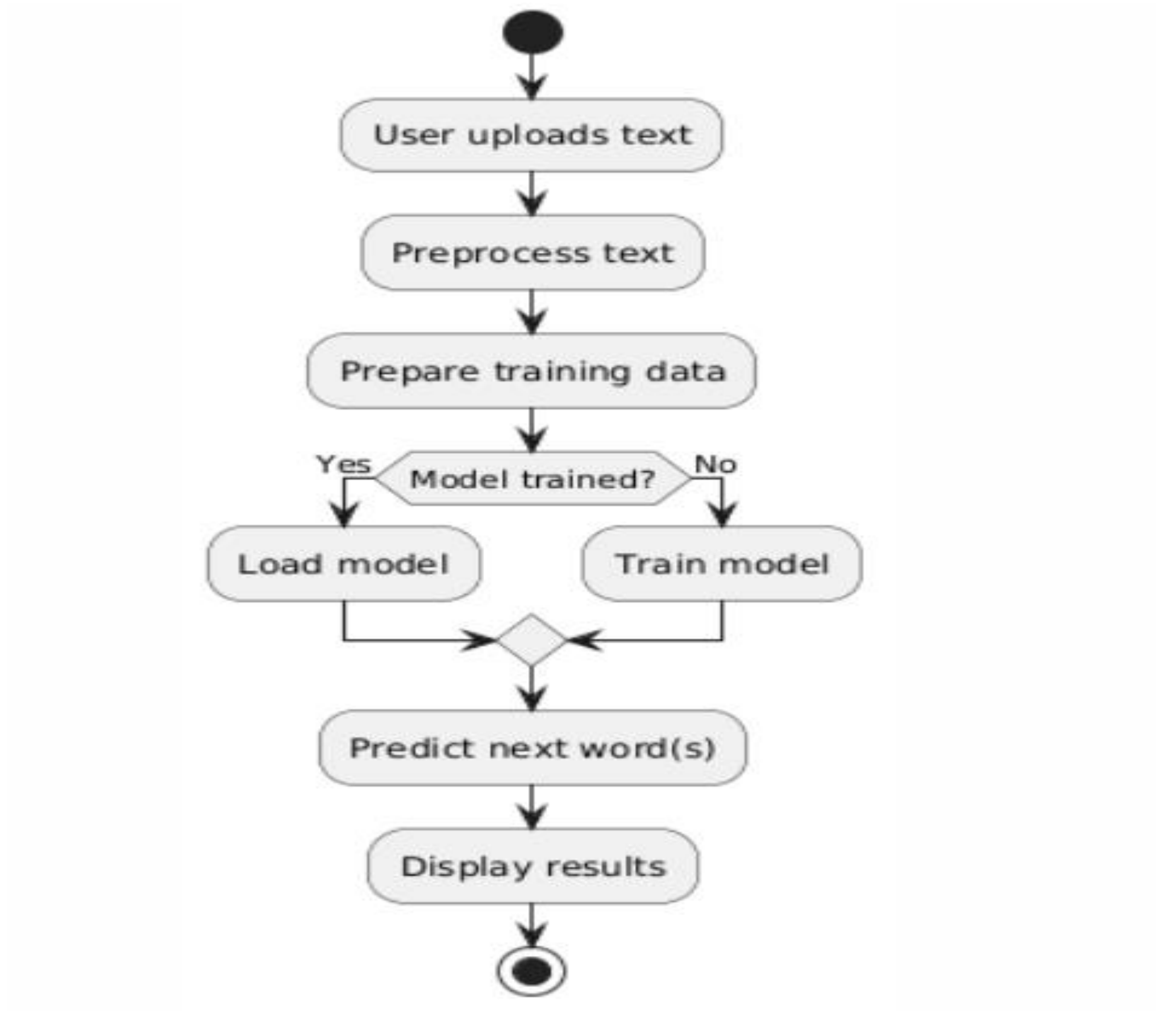


Figure 3.8: Activity diagram

CHAPTER 4

PROPOSED WORK MODEL

4.1 Technology stack

4.1.1 Python

Python is a versatile, high-level programming language known for its simplicity and readability. It is widely used in web development, data analysis, artificial intelligence, scientific computing, and automation. Python's extensive libraries and frameworks make it a preferred choice for developers and data scientists.

- **Role in Project**: All components including data preprocessing, model building, training, and deployment are written in Python.
- Key Features: Easy syntax, cross-platform compatibility, extensive libraries, and an active community.
- Huge collection of third-party packages via PyPI.
- Integration with Jupyter Notebooks for interactive development.
- Ideal for both quick prototyping and full-scale production systems.
- Applications: Machine learning, web applications, scripting, and data visualization.

4.1.2 NLTK

NLTK is a leading library for building Python programs to work with human language data, also known as text processing or Natural Language Processing (NLP). It provides tools for text analysis, tokenization, tagging, parsing, and semantic reasoning.

- **Role in Project**: Used for basic NLP preprocessing like tokenization, and text cleaning (if included).
- Includes a well-documented book, Natural Language Processing with Python, useful for learning and reference.
- Modular design allows selective use of functionalities as needed.
- Contains pre-labeled datasets for tasks like classification and tagging.
- Key Features:
  - Pre-trained datasets and models for NLP tasks.

    - Tools for tokenization, stemming, lemmatization, and sentiment analysis.
    - Supports linguistic research and educational purposes.
- Applications: Sentiment analysis, text classification, language translation, and chatbots.

4.1.4 Keras

Keras is a high-level neural networks API written in Python. It runs on top of TensorFlow, making it easy to build and train deep learning models. Keras is user-friendly, modular, and extensible.

- **Role in Project**: Used to define the LSTM model architecture and handle model training and evaluation.
- Facilitates experimentation with multiple architectures using minimal code.
- Extensive documentation and community support.
- GPU acceleration for faster training using TensorFlow backend.
- Key Features:
    - Simplified model building with sequential and functional APIs.
    - Built-in support for Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and LSTMs.
    - Extensive pre-trained models available in keras applications.
- Applications: Image recognition, language modeling, and AI applications.

4.1.4 Streamlit

Streamlit is an open-source Python library used to build interactive and data-driven web applications with minimal coding. It's particularly popular for machine learning and data science projects.

- **Role in Project**: Used to build a web-based UI that allows users to input text and receive next-word predictions in real time.
- Markdown and LaTeX support for rich text formatting and equation rendering.
- Built-in caching for performance optimization.
- Integration with live data sources, APIs, and databases.
- Key Features:
    - Rapid prototyping for data apps with minimal lines of code.
    - Supports integration with data visualization libraries like Matplotlib, Plotly, and Altair.

    - Real-time interactivity with widgets like sliders, buttons, and input boxes.
- Applications: Creating dashboards, showcasing machine learning models, and interactive reports.

4.1.5 NumPy

NumPy (Numerical Python) is a fundamental library for numerical computing in Python. It provides support for arrays, matrices, and a collection of mathematical functions to perform operations on these data structures efficiently.

- **Role in Project**: Supports preprocessing tasks, especially for reshaping and handling input/output data used in model training and inference.
- Basis for numerical computation in most Python-based machine learning libraries.
- Enables efficient memory management for matrix operations in neural networks.
- Works well with GPU-accelerated libraries like CuPy for even faster processing.
- Key Features:
    - Multidimensional array objects.
    - Mathematical operations for linear algebra, Fourier transforms, and statistics.
    - Highly efficient for large datasets due to vectorization.

4.2 Proposed model/algorithm

4.2.1 Overview

1. Next-word prediction is a task in natural language processing (NLP) where a model predicts the most probable word or set of words that follow a given sequence of text. This technology forms the basis of many text generation and auto-complete systems. By analyzing the context of preceding words, the model can suggest appropriate continuations, enhancing efficiency and accuracy in text-based communication.

4.2.2 Architecture of the model

The architecture of the next-word prediction system is designed to process textual input and predict the most probable next word based on learned language patterns. It combines preprocessing steps with a deep learning model built using LSTM (Long Short-Term Memory) layers. Below is a breakdown of each key component:

1. Input layer

- **Function**: Accepts a partial sentence or phrase from the user.
- **Source**: Text input provided through the Streamlit-based user interface.
- **Purpose**: Serves as the starting point for the entire prediction process, capturing user intent.
- **Example**: If the user types "Artificial Intelligence is", this input is passed to the next component for processing.

2. Text Preprocessing Module

- **Function**: Prepares raw input text for modeling.
- **Operations**:
  - Converts all text to lowercase to maintain uniformity.
  - Removes special characters, punctuation, or unwanted symbols that don't contribute meaningfully.
  - Trims or manages extra whitespaces for cleaner tokenization.
- **Purpose**: Ensures that input data matches the format used during training and avoids errors in prediction.
- **Tools Used**: Python string operations, regular expressions, and optionally NLTK for extra cleaning tasks.

3. Tokenizer

- **Function:** Converts cleaned text into numerical format that the model understands.
- **Process:**
  - Splits the sentence into individual words or tokens.
  - Each word is mapped to an integer index based on the vocabulary learned during training.

- **Type:** Can be a custom tokenizer created using Keras' Tokenizer() or a pre-trained tokenizer if using advanced models.
- **Purpose:** Translates text into sequences suitable for feeding into neural networks.

4. Sequence Padding

- **Function:** Standardizes the length of input sequences.
- **Why It's Needed:** Neural networks, especially LSTMs, require fixed-length inputs.
- **Process:**
  - Pads shorter sequences with zeros at the beginning (or end) to match required length.
  - Truncates longer sequences if they exceed the desired length.
- **Tool Used:** Keras' pad_sequences() method.
- **Purpose:** Ensures uniformity in input shapes and prevents shape mismatch during training or prediction.

5. LSTM Layer

- **Function:** This is the core of the model responsible for learning and predicting text sequences.
- **Why LSTM:**
  - LSTMs are a type of RNN designed to remember long-term dependencies in sequences.
  - They retain context over a series of words, improving language understanding.
- **Structure:**
  - Embedding layer: Transforms word indices into dense vectors capturing semantic meaning.
  - One or more stacked LSTM layers to process the sequence.
  - Dense layer with softmax activation to predict the next most probable word.
- **Purpose:** Based on the context from the input, it outputs a probability distribution over the vocabulary for the next word.
- **Framework Used:** Keras with TensorFlow backend.

6. Output Layer

- **Function:** Takes the output probabilities from the LSTM model and presents the top predictions to the user.
- **Operations:**
    - Extracts top-N words with the highest probability.
    - Converts predicted indices back to human-readable words using the tokenizer's word index dictionary.
- **UI Integration:** Displays the suggestions in real-time on the Streamlit interface.
- **Purpose:** Allows the user to see and optionally select from the most likely next-word options.

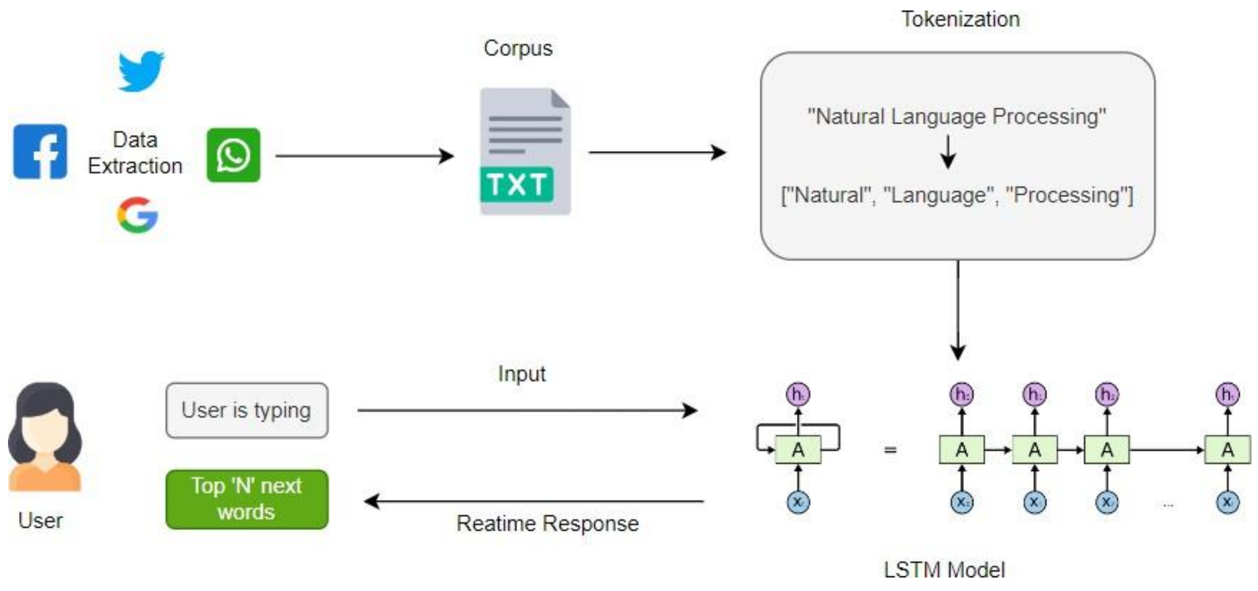


Figure 4.1: Block Diagram

4.2.3 Model Used

The system is built using a **Recurrent Neural Network (RNN)** architecture, specifically a **Long Short-Term Memory (LSTM)** network. LSTM is an advanced form of RNN designed to model **sequential data** and capture **long-range dependencies**, making it ideal for tasks like next-word prediction in natural language processing (NLP).

4.2.3.1 Why LSTM for next word prediction?

Traditional RNNs suffer from issues like **vanishing gradients**, making them ineffective at remembering words or patterns from earlier in a sequence. LSTM networks solve this by introducing a **memory cell** that can retain information over longer time steps, allowing the model to better understand **contextual flow** and **semantic patterns** in a sentence.

4.2.3.2 Advantages of LSTM

1. **Handles Long-Term Dependencies**
   - LSTM is specifically designed to remember information over long sequences, unlike traditional RNNs that struggle with vanishing gradients.
2. **Good for Sequential Data**
   - It’s ideal for time-series, text, or any data where the order of elements matters.
3. **Prevents Vanishing/Exploding Gradient Problems**
   - LSTM’s gating mechanism allows gradients to flow more effectively during backpropagation.
4. **Better Context Understanding**
   - It captures not just local patterns but also long-range relationships between words in a sentence.
5. **Widely Adopted and Well-Supported**
   - It's a well-known architecture supported by libraries like TensorFlow, PyTorch, and Keras, with lots of pre-existing implementations and research.

4.2.3.3 Disadvantages of LSTM

1. **Slower Training and Inference**
   - Due to sequential processing (can’t parallelize like Transformers), training takes longer.
2. **High Memory Usage**
   - LSTMs require more memory compared to simpler RNNs or even some Transformer models, especially with large vocabularies.
3. **Less Scalable than Transformers**
   - For large-scale datasets or complex language tasks, Transformers (like GPT) outperform LSTMs in both speed and accuracy.
4. **Harder to Interpret**

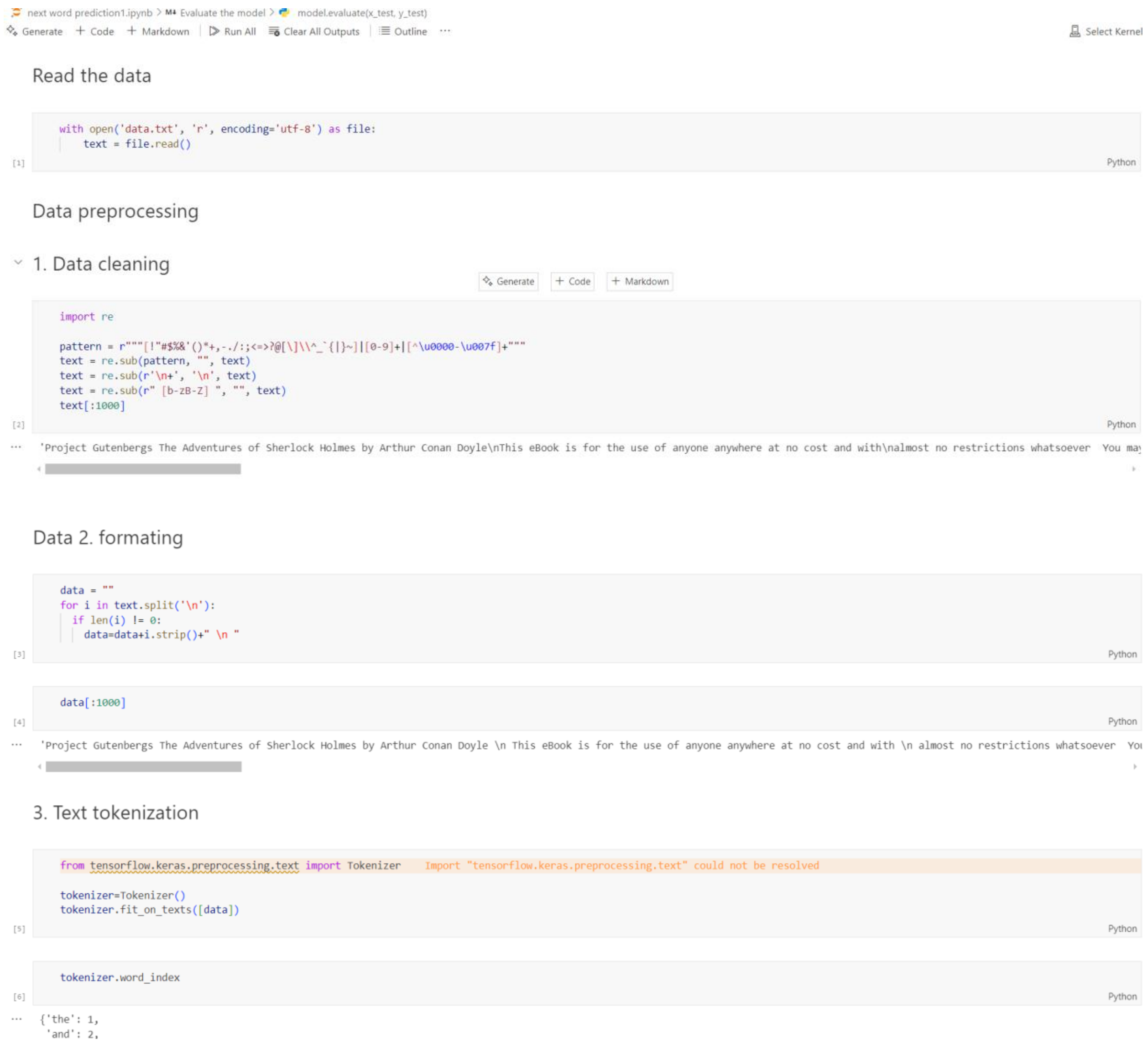
- While LSTMs are better than RNNs, their internal operations (gates and cell states) are still difficult to interpret or visualize.

5. **Needs Careful Tuning**
   - Hyperparameters like number of units, learning rate, sequence length, and regularization techniques need to be tuned properly for best performance.

CHAPTER 5

PROJECT RESULTS AND DISCUSSION

1. Next word prediction notebook


next word prediction1.ipynb > Evaluate the model > model.evaluate(x_test, y_test)
Generate + Code + Markdown | Run All | Clear All Outputs | Outline ···
Select Kernel

Read the data

```
with open('data.txt', 'r', encoding='utf-8') as file:
    text = file.read()
```

[1] Python

Data preprocessing

1. Data cleaning

Generate + Code + Markdown

```
import re

pattern = r"""[!"#$%&'()*+,-./:;<=>?@[\]\\^_`{|}~]|[0-9]+|[^\u0000-\u007f]+"""
text = re.sub(pattern, "", text)
text = re.sub(r'\n+', '\n', text)
text = re.sub(r" [b-zB-Z] ", "", text)
text[:1000]
```

[2] Python

```
'Project Gutenbergs The Adventures of Sherlock Holmes by Arthur Conan Doyle\nThis eBook is for the use of anyone anywhere at no cost and with\nalmost no restrictions whatsoever  You may
```

Data 2. formating

```
data = ""
for i in text.split('\n'):
  if len(i) != 0:
    data=data+i.strip()+" \n "
```

[3] Python

```
data[:1000]
```

[4] Python

```
'Project Gutenbergs The Adventures of Sherlock Holmes by Arthur Conan Doyle \n This eBook is for the use of anyone anywhere at no cost and with \n almost no restrictions whatsoever  You
```

3. Text tokenization

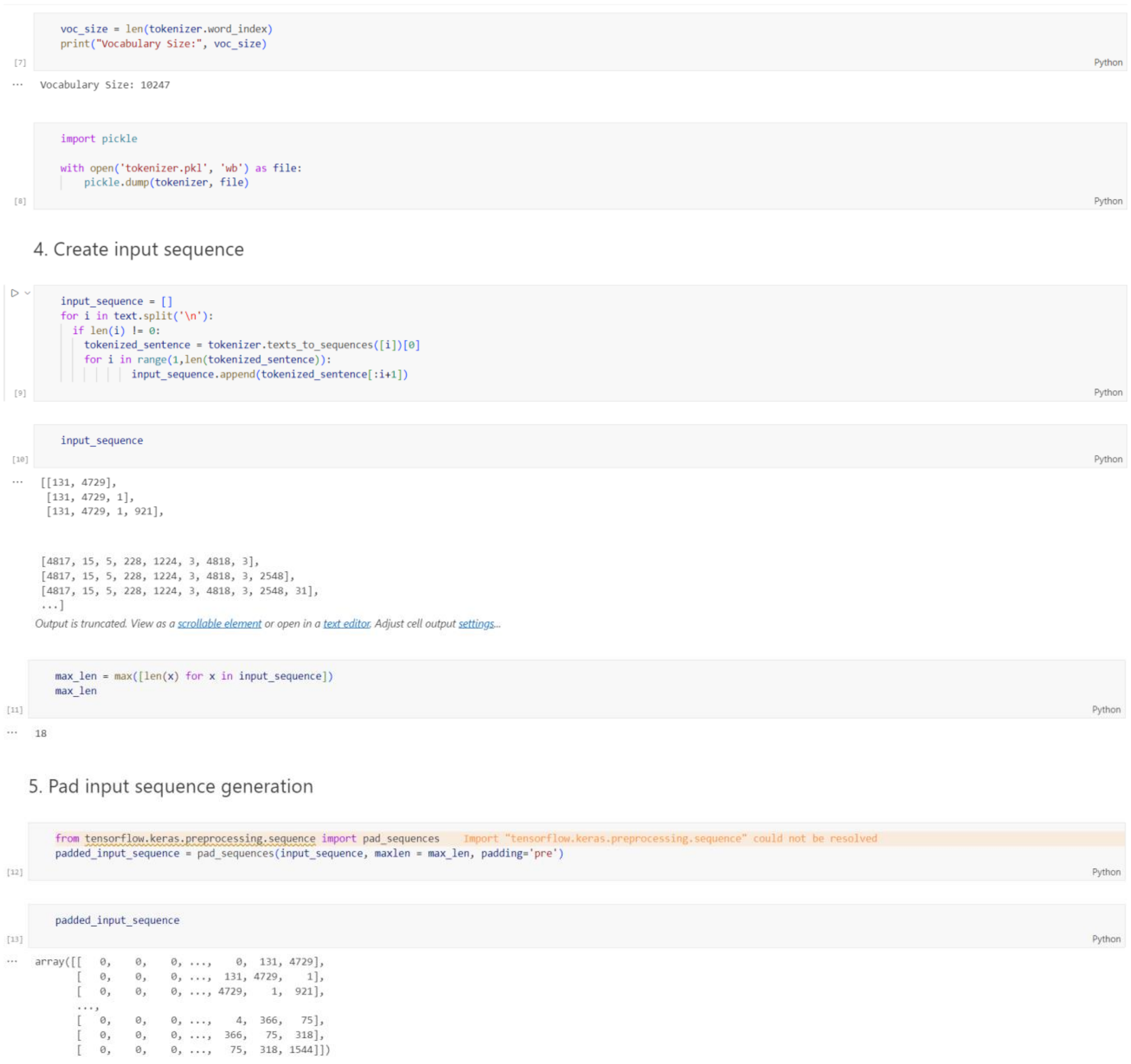
```
from tensorflow.keras.preprocessing.text import Tokenizer    Import "tensorflow.keras.preprocessing.text" could not be resolved

tokenizer=Tokenizer()
tokenizer.fit_on_texts([data])
```

[5] Python

```
tokenizer.word_index
```

[6] Python

```
{'the': 1,
 'and': 2,
```



```python
voc_size = len(tokenizer.word_index)
print("Vocabulary Size:", voc_size)
```
[7] Python

```
Vocabulary Size: 10247
```

```python
import pickle

with open('tokenizer.pkl', 'wb') as file:
    pickle.dump(tokenizer, file)
```
[8] Python

4. Create input sequence

```python
input_sequence = []
for i in text.split('\n'):
  if len(i) != 0:
    tokenized_sentence = tokenizer.texts_to_sequences([i])[0]
    for i in range(1,len(tokenized_sentence)):
        input_sequence.append(tokenized_sentence[:i+1])
```
[9] Python

```python
input_sequence
```
[10] Python

```
[[131, 4729],
 [131, 4729, 1],
 [131, 4729, 1, 921],


[4817, 15, 5, 228, 1224, 3, 4818, 3],
[4817, 15, 5, 228, 1224, 3, 4818, 3, 2548],
[4817, 15, 5, 228, 1224, 3, 4818, 3, 2548, 31],
...]
```
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

```python
max_len = max([len(x) for x in input_sequence])
max_len
```
[11] Python

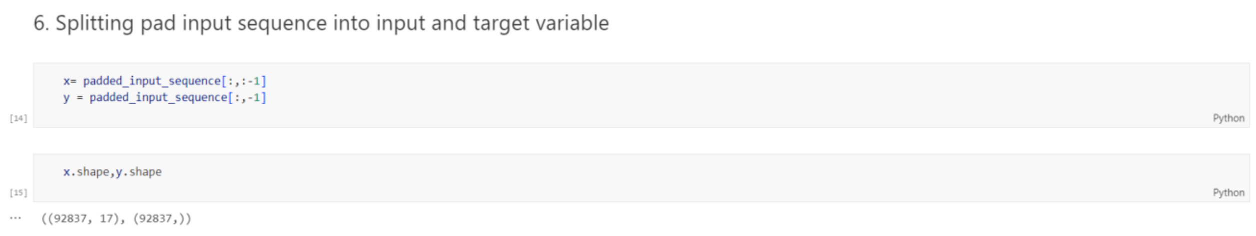
```
18
```

5. Pad input sequence generation

```python
from tensorflow.keras.preprocessing.sequence import pad_sequences    Import "tensorflow.keras.preprocessing.sequence" could not be resolved
padded_input_sequence = pad_sequences(input_sequence, maxlen = max_len, padding='pre')
```
[12] Python

```python
padded_input_sequence
```
[13] Python

```
array([[   0,    0,    0, ...,    0,  131, 4729],
       [   0,    0,    0, ...,  131, 4729,    1],
       [   0,    0,    0, ..., 4729,    1,  921],
       ...,
       [   0,    0,    0, ...,    4,  366,   75],
       [   0,    0,    0, ...,  366,   75,  318],
       [   0,    0,    0, ...,   75,  318, 1544]])
```



6. Splitting pad input sequence into input and target variable

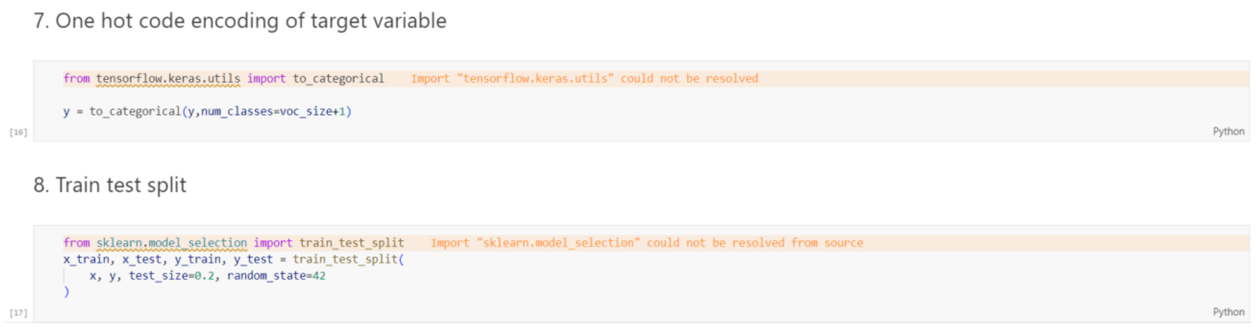
```
x= padded_input_sequence[:,:-1]
y = padded_input_sequence[:,-1]
```

[14] Python

```
x.shape,y.shape
```

[15] Python

```
((92837, 17), (92837,))
```



7. One hot code encoding of target variable

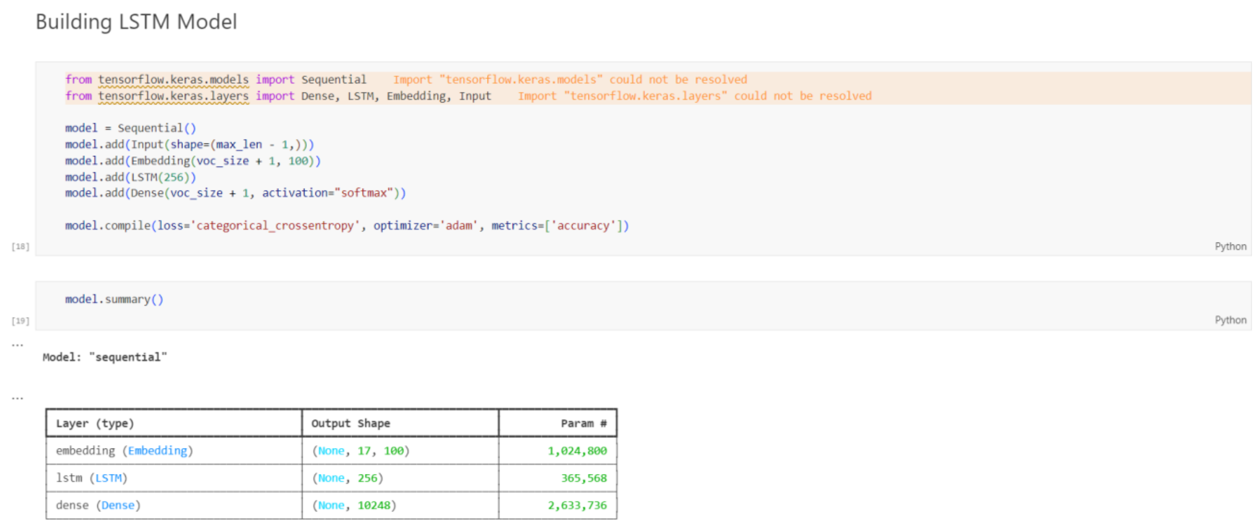
```
from tensorflow.keras.utils import to_categorical    Import "tensorflow.keras.utils" could not be resolved

y = to_categorical(y,num_classes=voc_size+1)
```

[16] Python

8. Train test split

```
from sklearn.model_selection import train_test_split    Import "sklearn.model_selection" could not be resolved from source
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.2, random_state=42
)
```

[17] Python



Building LSTM Model

```
from tensorflow.keras.models import Sequential    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.layers import Dense, LSTM, Embedding, Input    Import "tensorflow.keras.layers" could not be resolved

model = Sequential()
model.add(Input(shape=(max_len - 1,)))
model.add(Embedding(voc_size + 1, 100))
model.add(LSTM(256))
model.add(Dense(voc_size + 1, activation="softmax"))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

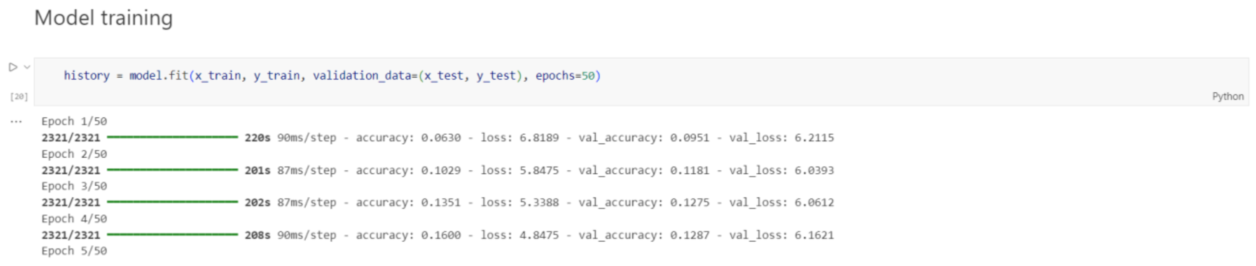
[18] Python

```
model.summary()
```

[19] Python

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 17, 100)	1,024,800
lstm (LSTM)	(None, 256)	365,568
dense (Dense)	(None, 10248)	2,633,736



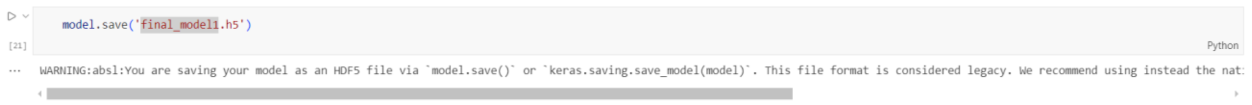
Model training

```
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=50)
```

[20] Python

```
Epoch 1/50
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 220s 90ms/step - accuracy: 0.0630 - loss: 6.8189 - val_accuracy: 0.0951 - val_loss: 6.2115
Epoch 2/50
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 201s 87ms/step - accuracy: 0.1029 - loss: 5.8475 - val_accuracy: 0.1181 - val_loss: 6.0393
Epoch 3/50
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 202s 87ms/step - accuracy: 0.1351 - loss: 5.3388 - val_accuracy: 0.1275 - val_loss: 6.0612
Epoch 4/50
2321/2321 ━━━━━━━━━━━━━━━━━━━━ 208s 90ms/step - accuracy: 0.1600 - loss: 4.8475 - val_accuracy: 0.1287 - val_loss: 6.1621
Epoch 5/50
```

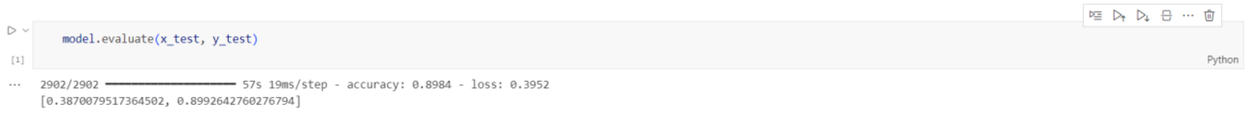
Save the model



```
model.save('final_model1.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the nat
```

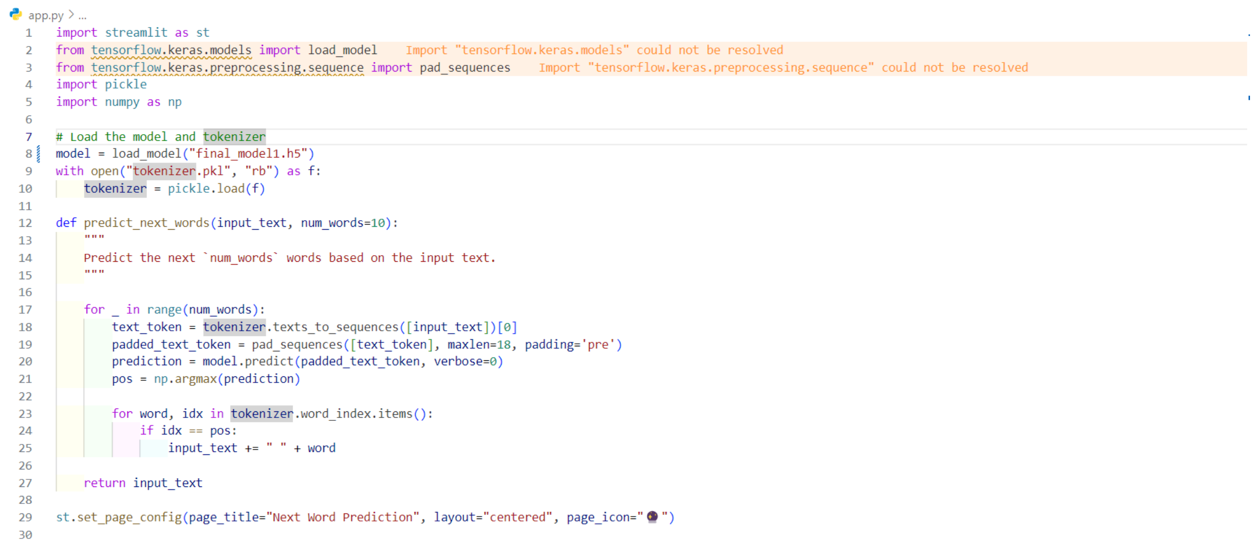
Evaluate the model



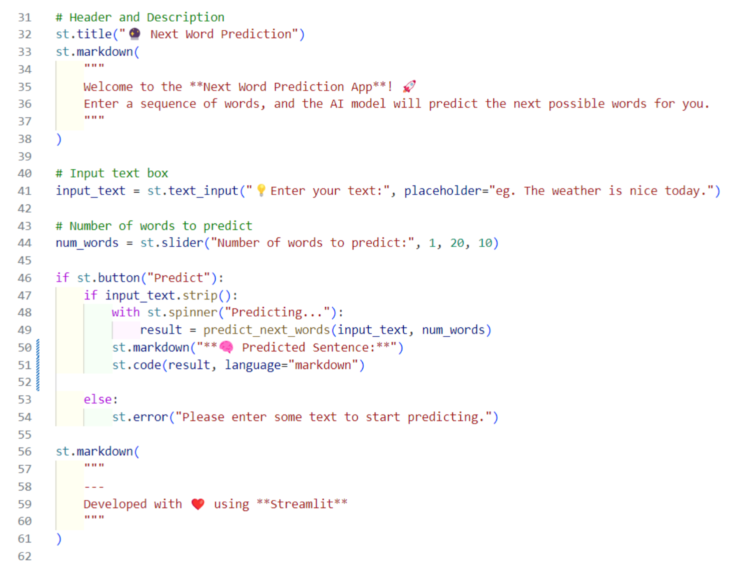
```
model.evaluate(x_test, y_test)
```

```
2902/2902 ━━━━━━━━━━━━━━━━━━━━ 57s 19ms/step - accuracy: 0.8984 - loss: 0.3952
[0.3870079517364502, 0.8992642760276794]
```

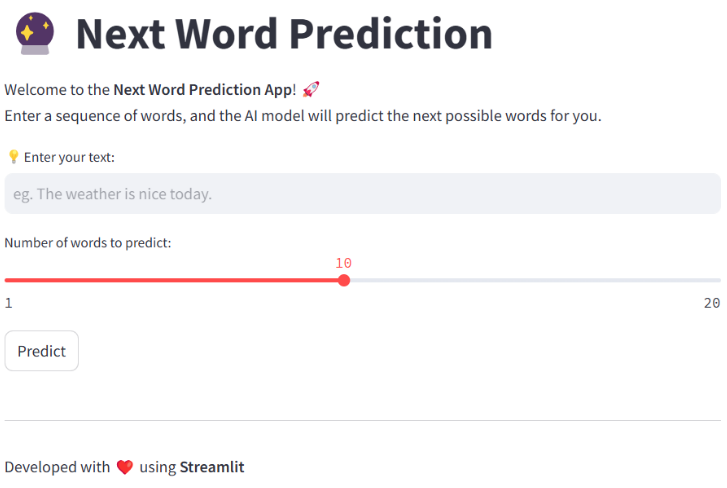
2. app.py file



```python
import streamlit as st
from tensorflow.keras.models import load_model    Import "tensorflow.keras.models" could not be resolved
from tensorflow.keras.preprocessing.sequence import pad_sequences    Import "tensorflow.keras.preprocessing.sequence" could not be resolved
import pickle
import numpy as np

# Load the model and tokenizer
model = load_model("final_model1.h5")
with open("tokenizer.pkl", "rb") as f:
    tokenizer = pickle.load(f)

def predict_next_words(input_text, num_words=10):
    """
    Predict the next `num_words` words based on the input text.
    """

    for _ in range(num_words):
        text_token = tokenizer.texts_to_sequences([input_text])[0]
        padded_text_token = pad_sequences([text_token], maxlen=18, padding='pre')
        prediction = model.predict(padded_text_token, verbose=0)
        pos = np.argmax(prediction)

        for word, idx in tokenizer.word_index.items():
            if idx == pos:
                input_text += " " + word

    return input_text

st.set_page_config(page_title="Next Word Prediction", layout="centered", page_icon="🔮")

```



```python
# Header and Description
st.title("🔮 Next Word Prediction")
st.markdown(
    """
    Welcome to the **Next Word Prediction App**! 🚀
    Enter a sequence of words, and the AI model will predict the next possible words for you.
    """
)

# Input text box
input_text = st.text_input("💡Enter your text:", placeholder="eg. The weather is nice today.")

# Number of words to predict
num_words = st.slider("Number of words to predict:", 1, 20, 10)

if st.button("Predict"):
    if input_text.strip():
        with st.spinner("Predicting..."):
            result = predict_next_words(input_text, num_words)
        st.markdown("**🧠 Predicted Sentence:**")
        st.code(result, language="markdown")

    else:
        st.error("Please enter some text to start predicting.")

st.markdown(
    """
    ---
    Developed with ❤️ using **Streamlit**
    """
)

```


Next Word Prediction

Welcome to the **Next Word Prediction App**!

Enter a sequence of words, and the AI model will predict the next possible words for you.

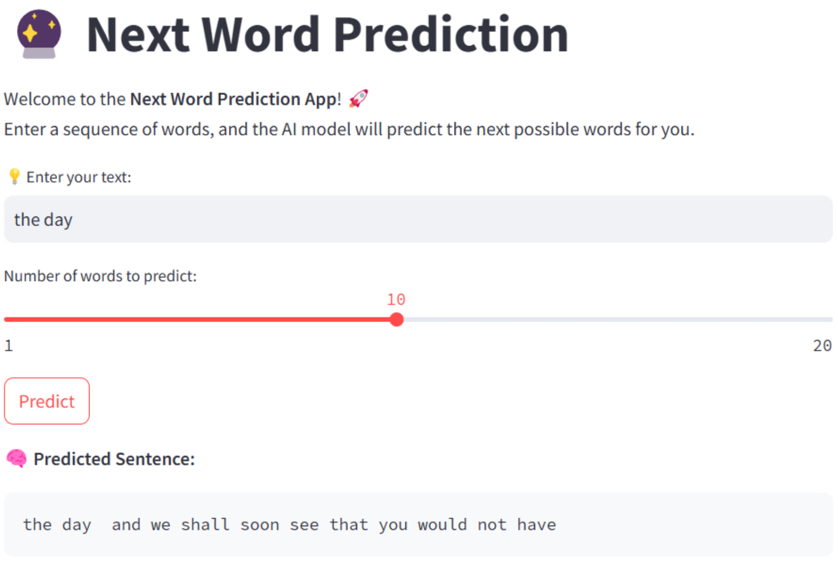
Enter your text:

eg. The weather is nice today.

Number of words to predict: 10 (1–20)

Developed with ❤️ using **Streamlit**

Figure 5.1: Input Screen


Next Word Prediction

Welcome to the **Next Word Prediction App**!

Enter a sequence of words, and the AI model will predict the next possible words for you.

Enter your text:

the day

Number of words to predict: 10 (1–20)

Predicted Sentence:

```
the day  and we shall soon see that you would not have
```

Figure 5.2: Output Screen

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

The next word prediction system proposed and implemented in this project demonstrates the effective use of Natural Language Processing (NLP) and deep learning techniques for intelligent text generation. By taking an incomplete sentence as input and generating contextually relevant next word suggestions, the system showcases its potential for integration into applications such as smart keyboards, chatbots, writing assistants, and educational tools.

The system is built using a modular architecture consisting of preprocessing, tokenization, and a predictive model (such as LSTM or Transformer), all wrapped in a user-friendly Streamlit interface. It successfully processes user input in real-time, delivering accurate and meaningful predictions based on learned language patterns from the training data.

6.2 Future work

6.2.1 Integration into applications

- Build an API or plugin for integration with popular platforms like Microsoft Word, Google Docs, or Slack.
- Use the model in educational tools to help students learn grammar, vocabulary, and writing techniques.
- Integrate into virtual assistants like Siri, Alexa, or Google Assistant for contextual word suggestions.

6.2.2 Handling Complex Language Constructs

- Enhance the model's ability to handle idiomatic expressions, slang, and multi-word phrases.
- Incorporate advanced tokenization methods like subword encoding to manage rare and out-of-vocabulary words.

6.2.3 Dataset Improvements

- Curate larger and more diverse training datasets to improve generalization.
- Use data augmentation techniques to create synthetic data for rare or unseen scenarios.
- Collect real-world user-generated data to fine-tune the model based on practical use cases.

6.2.4 Model Enhancements

- Experiment with advanced architectures like Transformers and GPT models to improve prediction quality.
- Introduce ensemble learning by combining multiple models to enhance robustness and performance.
- Explore lightweight architectures to reduce model size and memory usage for edge deployment.

REFERENCES

1. C. Aliprandi, N. Carmignani, N. Deha, P. Mancarella and M. Rubino, Advances in nlp applied to word prediction, 2008.

2. W. B. Cavnar and J. M. Trenkle, "N-gram-based text categorization", *Ann Arbor MI*, vol. 48113, no. 2, pp. 161-175, 1994.

3. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Upper Saddle River, NJ, USA:Prentice Hall Press, 2009.

4. T. K. Landauer, P. W. Foltz and D. Laham, "An introduction to latent semantic analysis", *Discourse processes*, vol. 25, no. 2–3, pp. 259-284, 1998.

5. Y. Wang, K. Kim, B. Lee and H. Y. Youn, "Word clustering based on pos feature for efficient twitter sentiment analysis", *Human-centric Computing and Information Sciences*, vol. 8, pp. 17, Jun 2018.

6. T. Mikolov, M. Karafiat, L. Burget, J. Cernocky and S. Khudanpur, "Recurrent neural network based language model", *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

7. F. A. Gers, J. Schmidhuber and F. Cummins, Learning to forget: Continual prediction with lstm, 1999.

8. C. Zhou, C. Sun, Z. Liu and F. Lau, "A c-lstm neural network for text classification", *arXiv preprint*, 2015.

9. M. K. Sharma and D. Smanta, "Word prediction system for text entry in hindi", *ACM Trans. Asian Lang. Inform. Process*, June 2014.

10. M. Sharma, D. Goplani and M. Tripathi, "An Approach for Predicting Related Word for the Hindi Language", *International Journal of Computer Applications Fundamental of Computer Science*, vol. {123}, no. {6}, 2015.

11. M. Ghayoomi and E. Daroodi, "A POS-based word prediction system for the Persian language", *Springer International Conference on Natural Language Processing*, pp. 138-147, 2008.

12. S. Sukhbaatar, A. Szlam, J. Weston and R. Fergus, "End-to-end memory networks", *Advances in neural information processing systems*, pp. 2440-2448, 2015.

13. D. Bahdanau, K. Cho and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *arXiv preprint*, 2014.
14. W. Stoop and A. V. D. Bosch, "Using idiolects and sociolects to improve word prediction", *[Sl]: Association for Computational Linguistics*, 2014.
15. Md. M. Haque, Md. T. Habib and Md. M. Rahman, "Automated word prediction in bangla language using stochastic language models", *arXiv preprint*, 2016.