

XCS608 – BIG DATA ANALYTICS LABORATORY

**MINI PROJECT REPORT**

ON

**PREDICTIVE TEXT GENERATION**

Submitted by

C.NATARAJ (121012012756)

M.SANTHOSH(121012012762)

B.MOKESH (1210120127755)

B .Tech CSE ( III Year)

**ABSTRACT :**

The project titled "Natural Language Processing using Deep Neural Networks: Predictive Text Generation" presents an innovative application of deep learning techniques in the field of Natural Language Processing (NLP). The primary objective of this project is to train a predictive text generation model capable of generating coherent and contextually relevant text based on a given input prompt.

The project utilizes the TensorFlow and Keras libraries for developing and training the model. The text data for training is sourced from a text file, specifically focusing on the domain of pizza-related content. This data is preprocessed to extract individual sentences and tokenize the words for further analysis.

The NLP model is a Sequential Neural Network consisting of an Embedding layer, an LSTM (Long Short-Term Memory) layer, and a Dense output layer. The model is trained with a categorical cross-entropy loss function and optimized using the Adam optimizer. Over 500 epochs, the model learns to predict the next word in a sequence given a context of preceding words.

The trained model is saved for future use and can be loaded to generate text based on a seed input. This allows for creative and contextually relevant text generation. The project's report demonstrates the ability to extend and apply the developed model for various text generation applications, which can be beneficial for content creation, chatbots, and more.

The project's results are showcased through an example in which the model generates text based on the input prompt and the model predicts the next word, offering a glimpse into the capabilities of the text generation model.

**INTRODUCTION :**

# The advent of Natural Language Processing (NLP) and deep learning has ushered in a new era of artificial intelligence, enabling machines to understand, interpret, and generate human language. One of the most intriguing and practical applications of NLP is text generation, a field that has witnessed remarkable progress in recent years. In this context, the project, titled "Natural Language Processing using Deep Neural Networks: Predictive Text Generation," explores the development and application of a predictive text generation model.

# Text generation has diverse applications, ranging from creative content generation to chatbots and automated writing assistance. It empowers machines to generate coherent and contextually relevant text, simulating human-like language production. The project leverages the power of deep learning, TensorFlow, and Keras to build a model that can generate text based on a given input prompt.

# 1. Motivation: The motivation behind this project lies in the increasing demand for intelligent systems capable of producing meaningful and coherent text. Whether it's for automating content generation, enhancing chatbot capabilities, or assisting writers in their creative process, text generation models hold significant promise.

# 2. Project Objective: The primary objective of this project is to design, train, and evaluate a predictive text generation model that can generate text with semantic and grammatical coherence..

# 3. Methodology: The project employs deep learning techniques, particularly Recurrent Neural Networks (RNNs), which are known for their ability to model sequential data. The model consists of an Embedding layer, an LSTM (Long Short-Term Memory) layer, and a Dense output layer. The use of these layers and the model's architecture are explained in detail.

# 4. Data Preparation: A crucial aspect of the project involves the acquisition and preprocessing of data. The text data is sourced from a designated file and divided into individual sentences, followed by tokenization to create a vocabulary of words. This processed data serves as the basis for training the text generation model.

# 5. Model Training: The model is trained over a specified number of epochs, fine-tuning its ability to predict the next word in a sequence of words. The training process is elaborated upon, highlighting the use of categorical cross- entropy loss and the Adam optimizer.

# 

# 6. Practical Application: In addition to model training, the report demonstrates the model's practical application by generating text based on a seed input

# 7. Conclusion and Implications: The project's findings and implications are discussed, emphasizing the potential applications of the developed text generation model in various domains, from content generation to natural language understanding.

# This project showcases the exciting possibilities of NLP and deep learning in the realm of text generation. By providing insights into the model's architecture, training process, and practical applications, this report contributes to the expanding field of artificial intelligence and its role in human language simulation

# PROBLEM DEFINITION :

Problem Statement:

The problem addressed by this project revolves around the need for developing a robust and context-aware text generation model within the domain of Natural Language Processing. With the exponential growth of textual data across various platforms, the demand for automated text generation tools that can create coherent and meaningful content has surged. This project aims to tackle the challenge of generating contextually relevant and grammatically accurate text, which is a fundamental problem in NLP. Specifically, it focuses on creating a text generation model that can understand and mimic a specific thematic context, The problem statement encompasses the design, training, and evaluation of a predictive text generation model, with the overarching goal of contributing to the development of intelligent systems capable of creative and context-aware language production

**PROPOSED SYSTEM :**

The proposed system aims to develop a predictive text generation model using Neural Networks for creative and contextually relevant text generation. This system leverages the power of deep learning libraries such as TensorFlow and Keras to train a model capable of generating coherent text based on a given input prompt. The primary focus of this system is to enhance natural language understanding and text generation .

**Objectives:**

The key objectives of the proposed system are as follows:

**Text Generation Model:**

Develop a robust and adaptive text generation model that can generate text with semantic coherence and relevance to the input context.

**Creative Content Generation :**

The system will have the capability to produce creative and engaging text, which can be employed for content creation, writing assistance, and similar applications.

**User Interaction :**

Enable user interaction with the model by allowing users to input a seed text or prompt and generating text based on that input, thereby offering a user-friendly and practical interface.

# HARDWARE SPECIFICATION :

The project is designed to run on any device (Desktop or Laptop) with Windows Latest Version.

# SOFTWARE SPECIFICATION :

* Google Colaboratory
* Windows 10 10.0 • IDE – Pandas
* Language-Python

# METHODOLOGY:

Data Acquisition:

The system sources text data from a designated file. This text data consists of sentences and paragraphs related to the domain of pizza, ensuring the thematic focus of the model.

Data Preprocessing:

Data preprocessing involves cleaning and structuring the text data. It includes tasks such as sentence extraction, tokenization, and vocabulary creation. Tokenization is essential to convert text into numerical values that can be used for model training.

Model Architecture:

The heart of the system is the predictive text generation model. The architecture involves three main layers:

Embedding Layer:

This layer is responsible for converting words into dense vectors, which can be processed by the model. It allows the model to understand the relationships between words.

LSTM (Long Short-Term Memory) Layer:

The LSTM layer is crucial for handling sequential data. It has the ability to capture context and dependencies between words, making it well-suited for text generation.

Dense Output Layer:

The dense layer serves as the output layer, predicting the probability distribution of the next word in a sequence.

Model Training:

The model is trained over a specified number of epochs. Training involves the use of a categorical cross-entropy loss function and the Adam optimizer to minimize the loss and improve the model's ability to generate relevant text.

User Interaction:

After model training, the system allows users to interact with the model. Users can input a seed text or prompt, and the model will generate text based on this input. This interactive feature makes the system practical and user-friendly.

**Implementation of Next word prediction :**

#Corpus to train

#read the corpus

corpus=open("corpus.txt").read()

print(corpus)

#preprocess the corpus import re

corpus=corpus.lower()

clean\_corpus=re.sub('[^a-z0-9]+',' ', corpus)

clean\_corpus

#Data Preparation

#required libraries import nltk

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

from keras.preprocessing.text import Tokenizer from keras.utils import to\_categorical

import numpy as np

#tokenizing the text into words

tokens = word\_tokenize(clean\_corpus) tokens

#length of the sequence to train

train\_len = 3

#converting the data into required sequence

text\_sequences = []

for i in range(train\_len,len(tokens)+1): seq = tokens[i-train\_len:i]

text\_sequences.append(seq)

print(text\_sequences)

#converting the texts into integer sequence

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(text\_sequences)

sequences = tokenizer.texts\_to\_sequences(text\_sequences)

print(sequences)

sequences=np.asarray(sequences)

#vocabulary size

vocabulary\_size = len(tokenizer.word\_counts)+1

print(vocabulary\_size)

#train X

train\_inputs=sequences[:,:-1]

print(train\_inputs)

#input sequence length

seq\_length=train\_inputs.shape[1]

print(seq\_length)

#train Y

train\_targets=sequences[:,-1]

print(train\_targets)

#one hot encoding

train\_targets = to\_categorical(train\_targets, num\_classes=vocabulary\_size)

print(train\_targets)

#Let's build the model!

#required libraries import torch

from torch.optim import Adam import torch.nn as nn

#lstm model

class lstm(nn.Module):

def init (self, vocab\_size, embed\_size, hidden\_size):

super(). init ()

#simple lookup table that stores embeddings of a fixed dictionary and size. self.embed = nn.Embedding(vocab\_size, embed\_size)

#lstm

self.lstm = nn.LSTM(embed\_size, hidden\_size, num\_layers=2, bidirectional=False)

#fully connected layer

self.linear = nn.Linear(hidden\_size\*seq\_length,vocab\_size)

def forward(self, input\_word):

#input sequence to embeddings

embedded = self.embed(input\_word)

#passing the embedding to lstm model output,

hidden = self.lstm(embedded)

#reshaping

output=output.view(output.size(0), -1)

#fully connected layer

output = self.linear(output)

return output

hidden

model=lstm(vocab\_size=vocabulary\_size,embed\_size=128, hidden\_size=256)

model

lstm((embed): Embedding(36, 128)

(lstm): LSTM(128, 256, num\_layers=2)

(linear): Linear(in\_features=512, out\_features=36, bias=True)

)

#Adam optimizer

optimizer= Adam(model.parameters(), lr=0.07)

#loss

criterion = nn.BCEWithLogitsLoss()

#training the model def train(epoch):

#set the model to train model.train()

tr\_loss=0

#clearing the Gradients optimizer.zero\_grad()

#predict the output

y\_pred, (state\_h, state\_c) = model(torch.from\_numpy(train\_inputs))

#compute the loss

loss=criterion(y\_pred,torch.from\_numpy(train\_targets))

losses.append(loss)

#backpropagate loss.backward()

#update the parameters optimizer.step()

tr\_loss = loss.item()

print("Epoch : ",epoch,"loss : ",loss)

#number of epoch no\_epoch=50

losses=[]

for epoch in range(1,no\_epoch+1):

train(epoch)

import torch

import matplotlib.pyplot as plt

# Print the type of the losses variable

print(type(losses))

import numpy as np np.ndarray

numpy.ndarray

print(type(losses))

losses = torch.tensor(losses)

losses = losses.detach()

losses = losses.numpy()

#Prediction

def predict\_next\_word(text):

# Set the model to evaluation mode

model.eval()

# Preprocess the input text

text = text.lower().strip()

# Tokenize the input text

input\_tokens = word\_tokenize(text)

# Convert tokens to integer sequence

sequences = tokenizer.texts\_to\_sequences([input\_tokens])

# Convert sequences to numpy array sequences = np.asarray(sequences)

with torch.no\_grad():

# Convert sequences to tensor and ensure it's of type torch.LongTensor sequences = torch.from\_numpy(sequences).long()

# Predict the output

predict, (hidden, cell) = model(sequences)

# Get the output from the LSTM

output = predict[-1] # Get the output of the last time step

# Apply softmax layer

softmax = torch.exp(output) prob = list(softmax.numpy())

# Get the index of the predicted word predictions = np.argmax(prob)

# Convert the index back to word

next\_word = tokenizer.sequences\_to\_texts([[predictions]])

return next\_word[0] # Return the first prediction

**Example-1**

#we trained our model with sequence length of 2

input\_text=input("Give Input Text :")

print(input\_text)

Give Input Text :Next Word Next Word

print("Possible next word will be:")

predict\_next\_word(input\_text)

**Possible next word will be: 'prediction'**

Example-2

#we trained our model with sequence length of 2

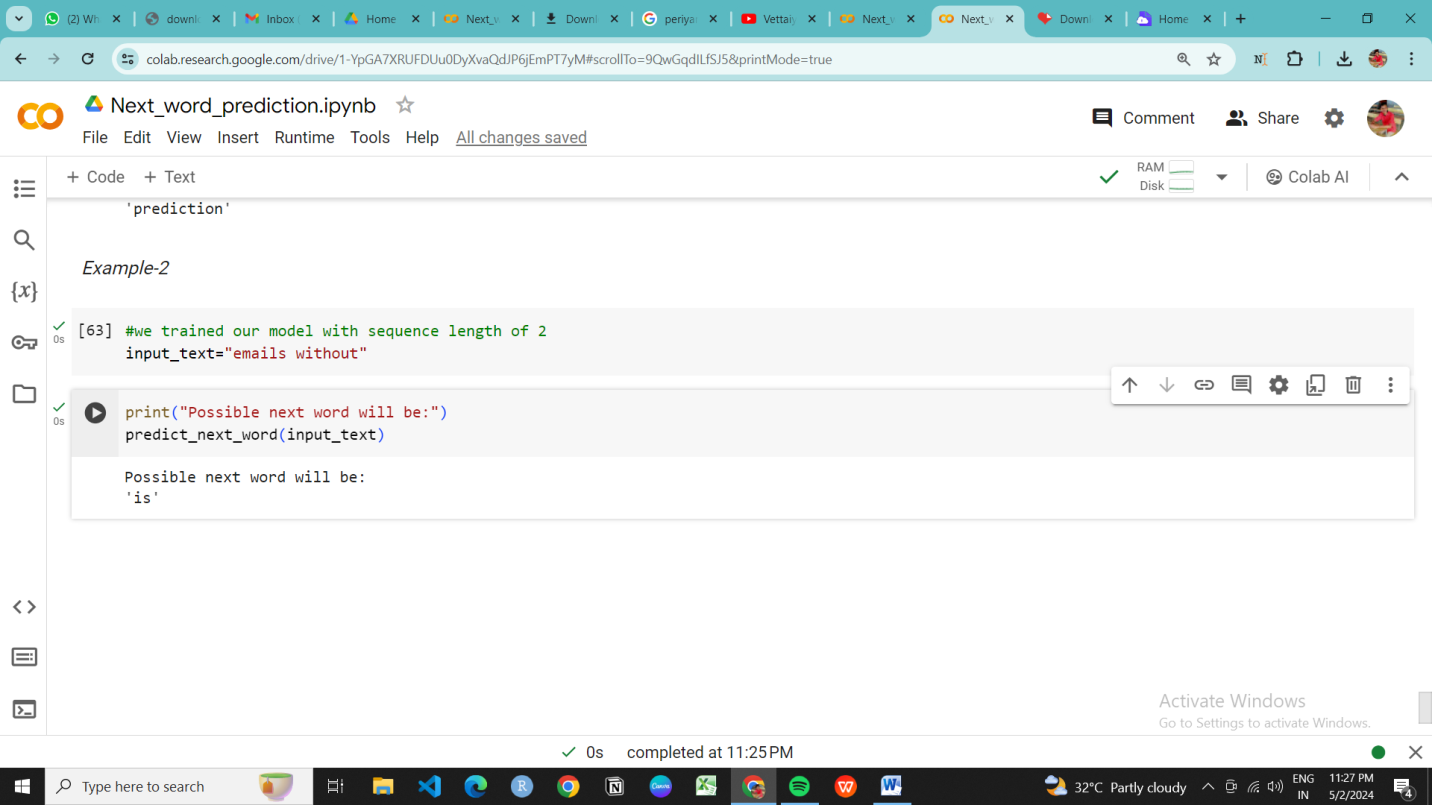
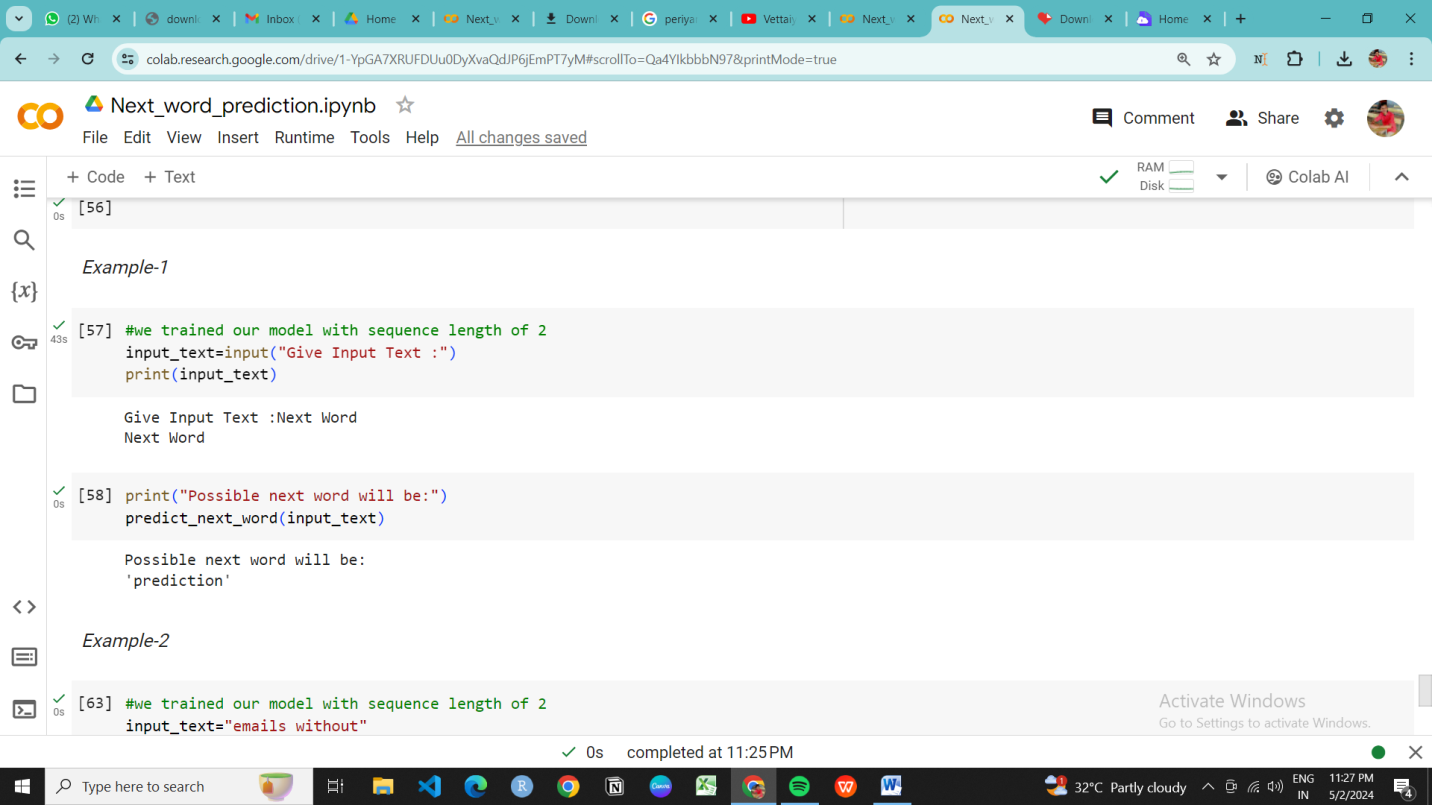
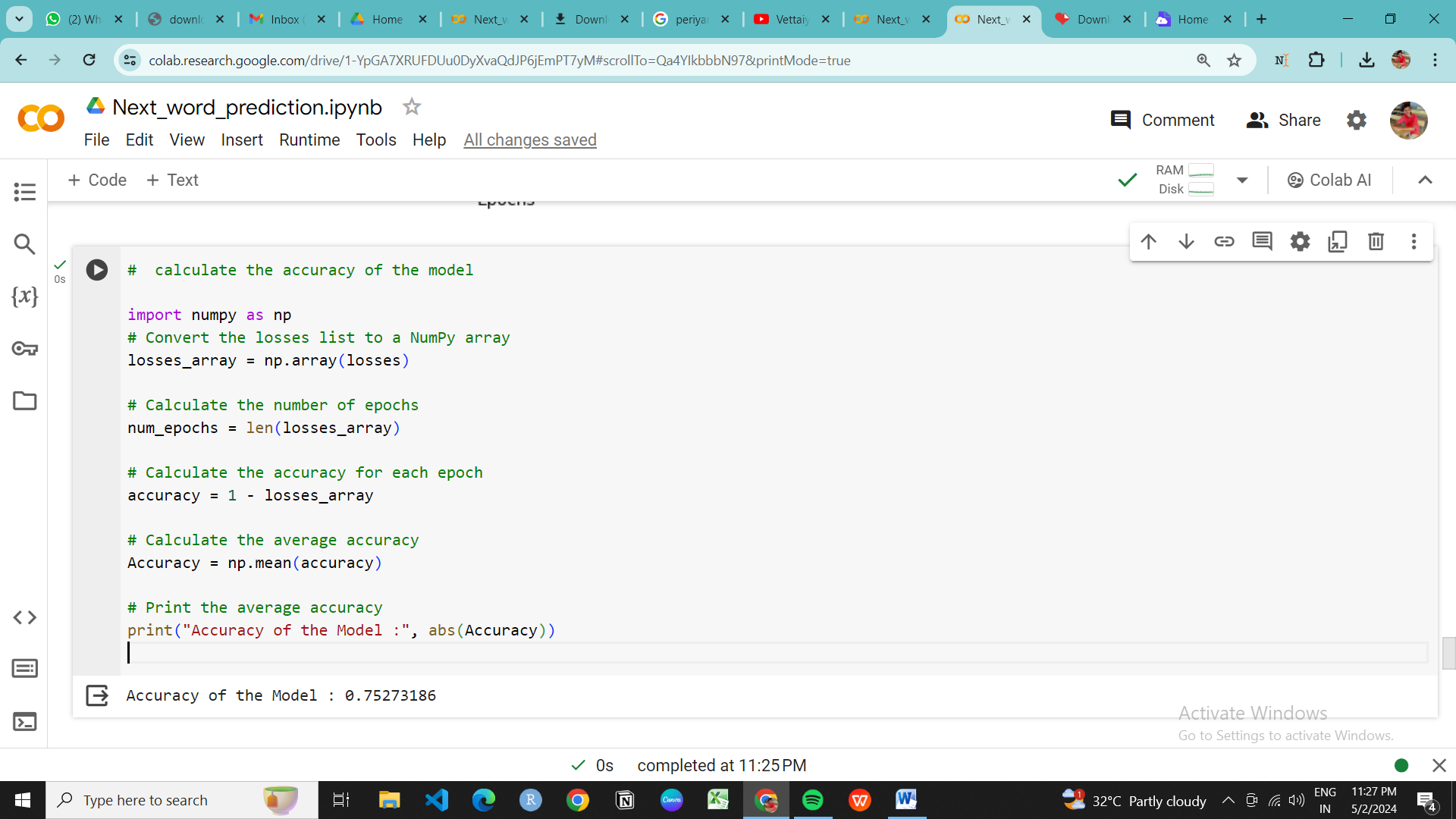
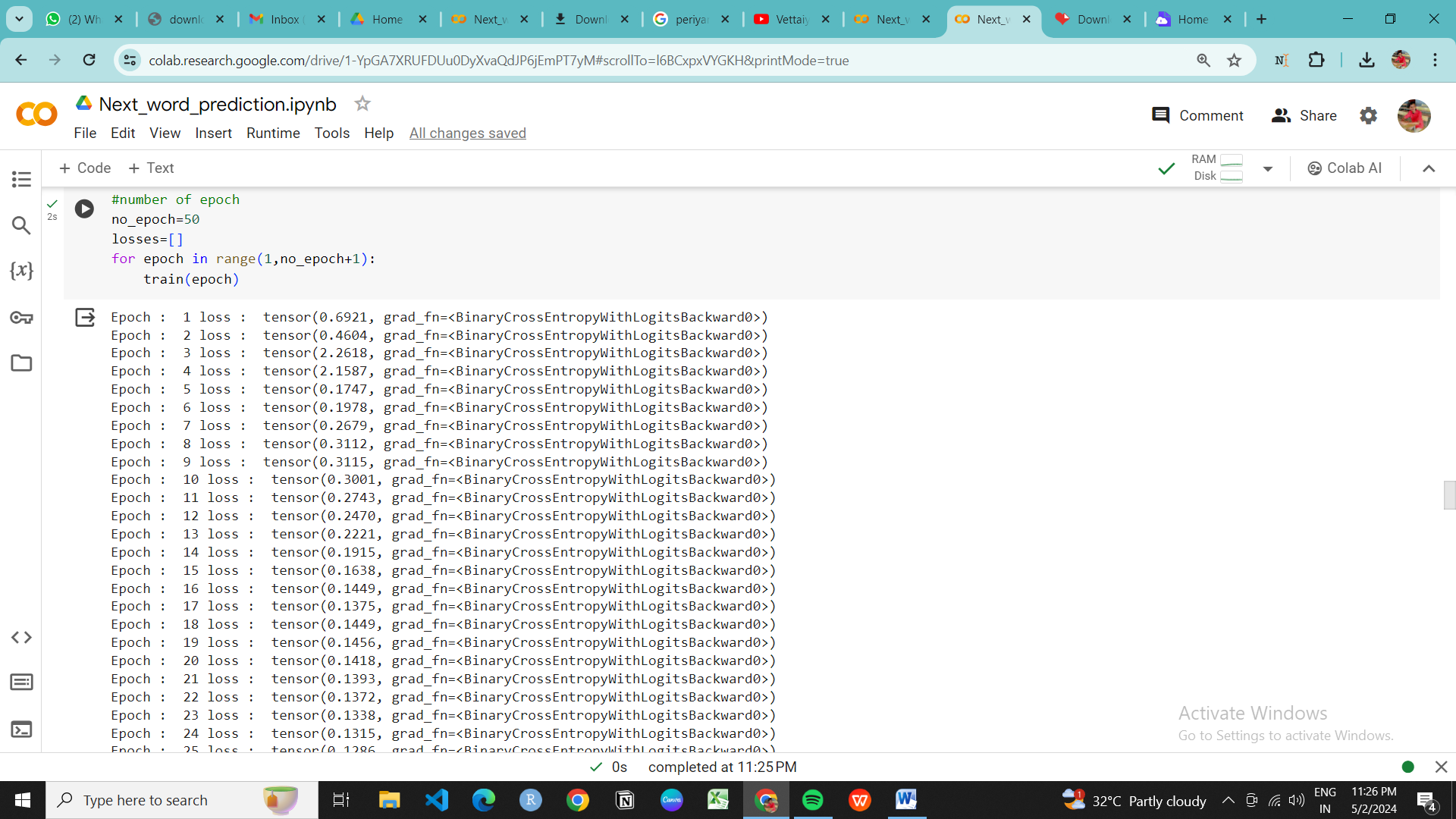
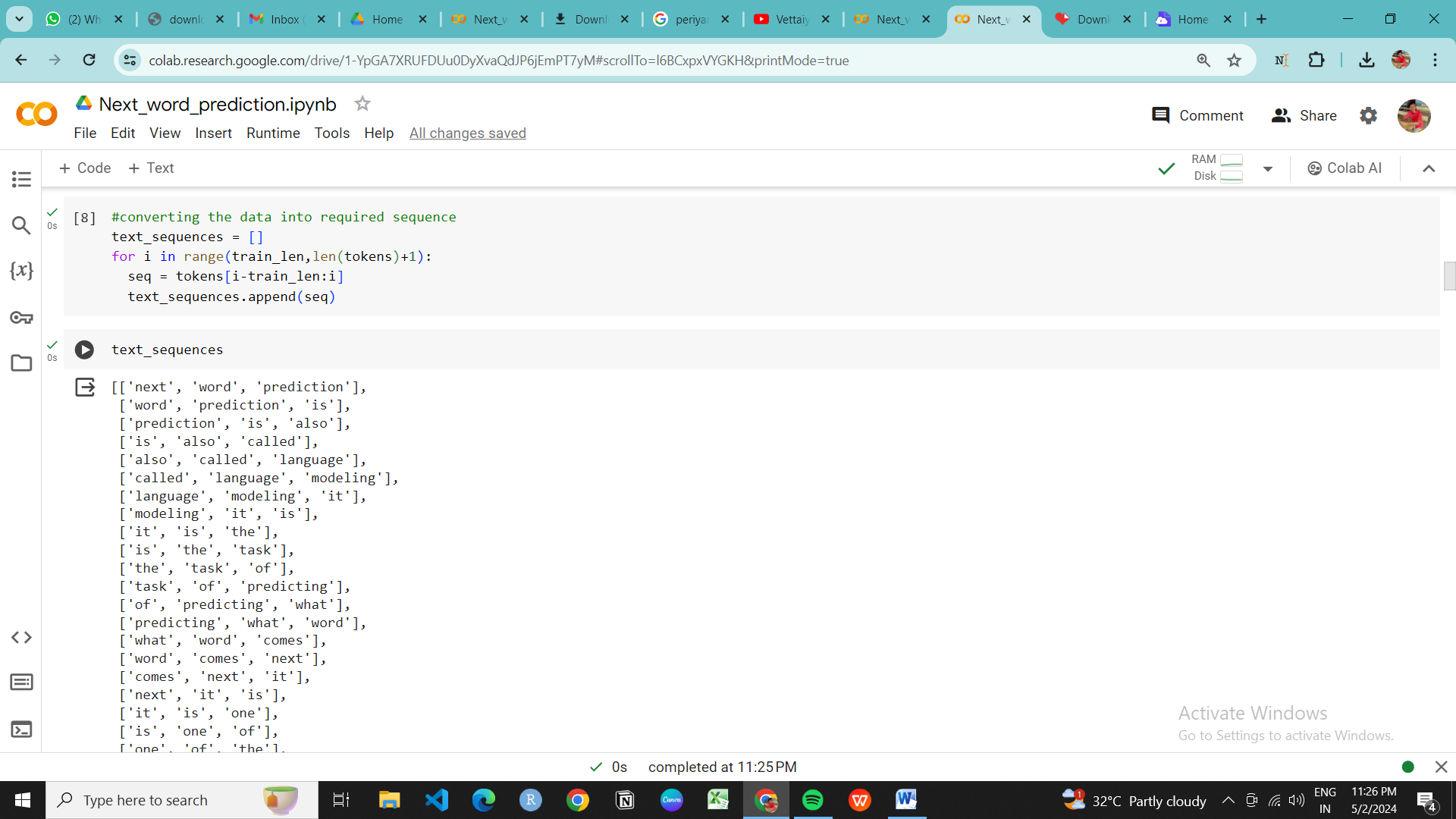
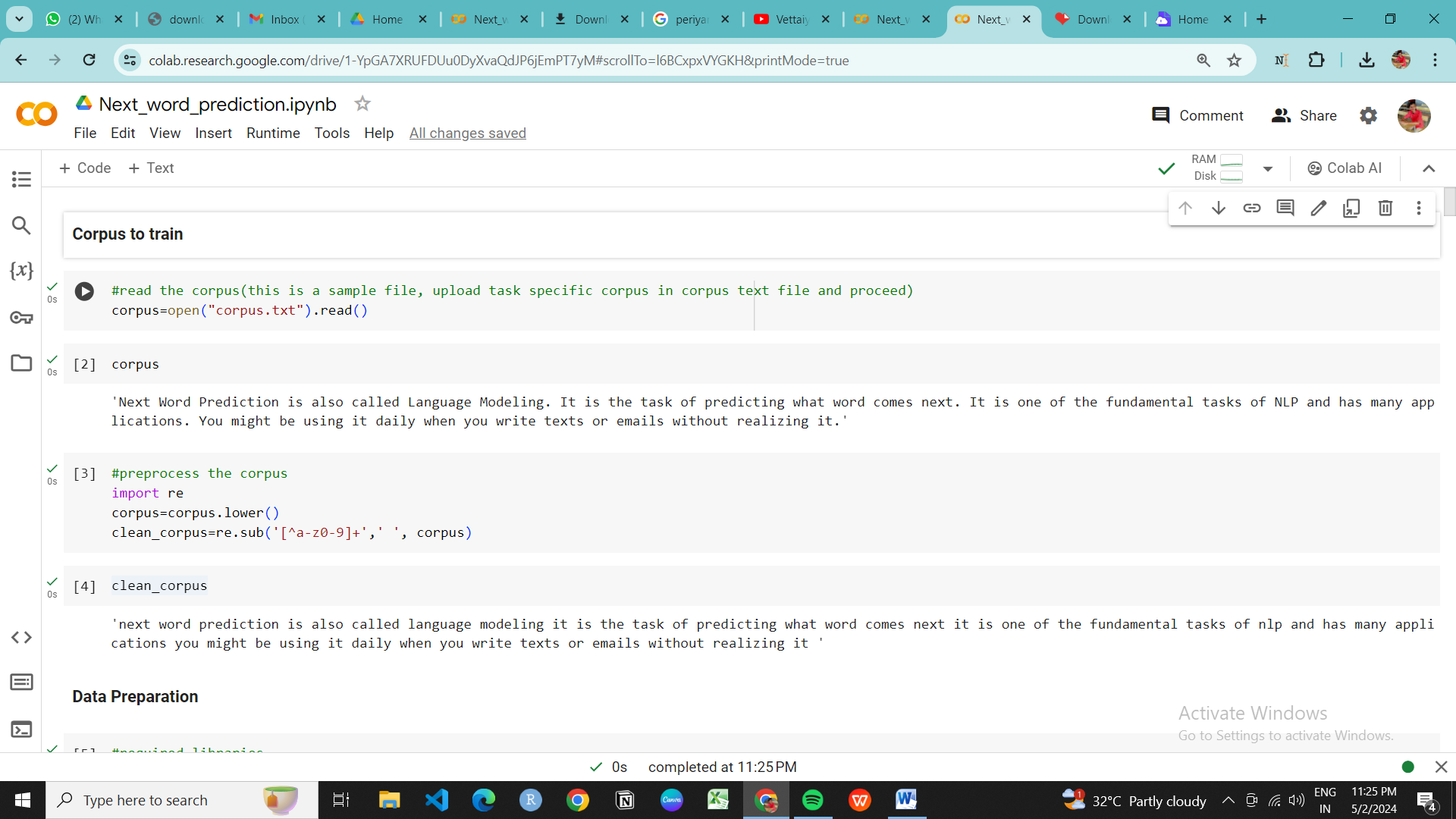
input\_text="predicting what"

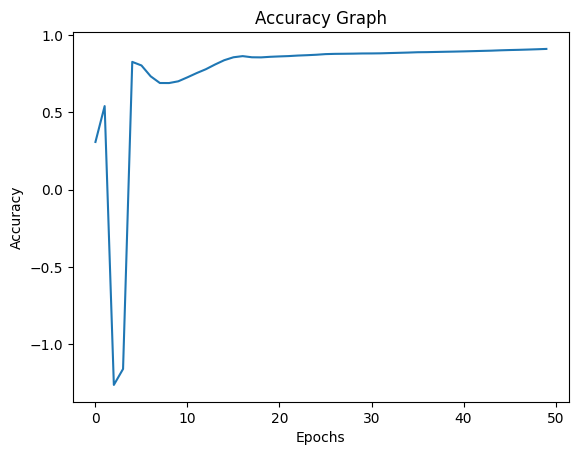
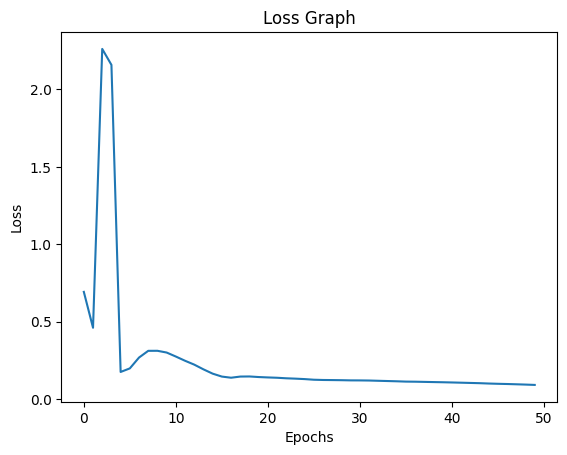
print("Possible next word will be:")

predict\_next\_word(input\_text)

**Possible next word will be: 'is'**

**RESULTS AND DISCUSSION:**

****



**APPLICATIONS:**

The system's applications extend to various domains:

a. Content Creation: Content creators can utilize the system to generate content ideas, create engaging articles, or generate marketing

b. Chatbots: The system can enhance the conversational abilities of chatbots, making them more context-aware and capable of generating meaningful responses.

c. Writing Assistance: Writers can use the system to overcome writer's block, find synonyms and creative alternatives, or generate thematic text to incorporate into their work.

**FUTURE DEVELOPMENT** :

1. Advanced Neural Network Architectures: Explore Transformer-based models like GPT or BERT, which have shown remarkable performance in various natural language processing tasks. These models can capture complex language patterns and dependencies, leading to more accurate predictions.
2. Fine-tuning: Fine-tune pre-trained language models on domain-specific datasets to improve their understanding of specialized language patterns. Fine-tuning can significantly enhance prediction accuracy in specific domains such as medicine, law, or finance.
3. Contextual Understanding: Develop techniques to better understand the context of the input text, such as incorporating larger context windows or utilizing syntactic and semantic analysis. Contextual understanding is crucial for making accurate predictions, especially in ambiguous or context-dependent scenarios.
4. User Personalization: Implement mechanisms to personalize predictions based on individual users' writing styles and preferences. By leveraging user feedback and historical input, the model can adapt its predictions to better suit the needs of specific users.
5. Efficient Inference: Optimize the model for efficient inference on various devices, including mobile and edge devices. Techniques such as model compression and hardware acceleration can reduce computational requirements and latency, making the prediction system more accessible and responsive.

**CONCLUSION :**

In conclusion,Recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks for next word prediction represents a significant advancement in natural language processing. These architectures excel at capturing sequential dependencies and contextual information, making them well-suited for predicting the next word in a sequence of text.

Through the use of RNNs and LSTMs, next word prediction systems can effectively model complex language patterns and dependencies, resulting in more accurate predictions. By processing input sequences iteratively and retaining information over multiple time steps, these networks can learn meaningful representations of text and generate contextually relevant predictions.

It's important to acknowledge some limitations of RNNs and LSTMs, such as difficulties in capturing long-range dependencies and the potential for vanishing or exploding gradients during training. Despite these challenges, ongoing research and advancements in model architectures, training techniques, and optimization algorithms continue to enhance the performance and capabilities of RNNs and LSTMs for next word prediction.

The utilization of RNNs and LSTMs in next word prediction projects represents a powerful approach for modeling language sequences and providing accurate, contextually relevant predictions

Top of Form

**REFERENCES :**

1. Mikolov, T., Karafiát, M., Burget, L., et al. (2010). "Recurrent Neural Network Based Language Model." Proceedings of the 11th Annual Conference of the International Speech Communication Association.
2. Hochreiter, S., & Schmidhuber, J. (1997). "Long Short-Term Memory." Neural Computation. DOI: 10.1162/neco.1997.9.8.1735
3. Ilya Sutskever, Oriol Vinyals, Quoc V. Le. "Sequence to Sequence Learning with Neural Networks" (2014)
4. Christopher Olah. "Understanding LSTM Networks" - Provides in-depth insights into LSTM operation.