UNIVERSITY OF ELOUED

University of El Oued
Faculty of Exact Sciences
Computer Science Department

ACADEMIC LICENCE

End of Year project report

Predicting the Next Word with Machine Learning model

Presented by:

Hala Mohamed Islam
Hibi Adam
Kaddour Rabeh

Supervised by:

Dr.Guia Sana Sahar

Academic Year: 2024/2025

Thanks to Our Supervisor Guia Sana Sahar

We would like to express our sincere gratitude to our esteemed supervisor, Guia Sana Sahar, for her patience, continuous support, and valuable guidance, which had a significant impact on the completion of this work. Her consistent assistance and insightful advice greatly helped us clarify key concepts and overcome the challenges we faced throughout this project. We deeply appreciate her dedication and sincere efforts.

abstract

Next Word Prediction is a task within Natural Language Processing (NLP) that involves predicting the next word in a given sequence based on the preceding words. This process is crucial for a wide range of applications, including text completion, machine translation, and intelligent writing assistants. By analyzing large amounts of textual data, machine learning models can identify patterns in language use and generate contextually relevant predictions.

The goal of this report is to explore the implementation of next-word prediction using deep learning models, particularly Recurrent Neural Networks (RNNs) and Transformers, which have shown significant improvements in prediction accuracy over traditional methods.

Keywords: Next Word Prediction, Natural Language Processing, Deep Learning, Recurrent Neural Networks, Transformers

الملخّص

يُعدّ توقّع الكلمة التالية مهمةً ضمن مجال معالجة اللغة الطبيعية ،(NLP) حيث يتمثّل في التنبؤ بالكلمة التالية في تسلسل معين استنادًا إلى الكلمات السابقة. وتُعدّ هذه العملية ضرورية لمجموعة واسعة من التطبيقات، بما في ذلك إكمال النصوص، والترجمة الآلية، والمساعدات الذكية في الكتابة. من خلال تحليل كميات كبيرة من البيانات النصية، يمكن لنماذج التعلم الآلي التعرّف على أنماط استخدام اللغة وتوليد توقّعات ذات صلة وسياق مناسب.

يهدف هذا التقرير إلى استكشاف تطبيق تقنيات التنبؤ بالكلمة التالية باستخدام نماذج التعلم العميق، لا سيما الشبكات العصبية المتكررة (RNNs) ونماذج المحوّلات ،(Transformers) والتي أظهرت تحسّنًا كبيرًا في دقة التنبؤ مقارنةً بالطرق التقليدية.

الكلمات المفتاحية: توقّع الكلمة التالية، معالجة اللغة الطبيعية، التعلم العميق، الشبكات العصبية المتكررة، المحوّلات

Contents

List of Figures

List of Equations

List of Tables

Introduction

The task of **Next Word Prediction** is a critical application within the broader field of **Natural Language Processing (NLP)**, which focuses on enabling machines to understand and generate human language. By leveraging advanced models, such as **Transformers** and **Recurrent Neural Networks (RNNs)**, next-word prediction systems can accurately predict the next word in a sequence, enhancing applications like **email autocompletion**, **code suggestion tools**, and **real-time language translation**. It is a technology designed to speed up typing and reduce errors.

This report investigates the implementation of next-word prediction using deep learning models, with a focus on Recurrent Neural Networks (RNNs) and Transformer-based architectures. These models have demonstrated substantial improvements in prediction accuracy compared to traditional methods. Specifically, we employ a Bidirectional LSTM model and leverage transfer learning with DistilBERT, a lightweight variant of BERT, to enhance performance while maintaining computational efficiency.

The core objective of this research is to explore and improve the predictive capabilities of these models to offer more accurate and contextually relevant word suggestions.

This report is organized into three chapters:

- **Chapter 01**: This chapter introduces **Natural Language Processing (NLP)** and outlines the foundational principles of language modeling, discussing various techniques from traditional models like **N-grams** to advanced **Transformer-based architectures**. It provides a comprehensive foundation for understanding how language models operate and their applications.
- **Chapter 02**: This chapter focuses on the **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks, examining how they address challenges like the **vanishing gradient problem** and enable sequential data modeling. The chapter also explores how these models lay the groundwork for next-word prediction.
- **Chapter 03**:In this chapter, we presented the overall **design** and **implementation** of our **next-word prediction model**. We detailed the **dataset**, **model architecture**, **training process**, and concluded with the **evaluation** and **prediction results** of the final model.

CHAPTER I

1 Next Word Prediction and Language Modeling Task

In this chapter, we will explore the tasks of Next Word Prediction and Language Modeling, both of which are fundamental in the field of Natural Language Processing (NLP). We begin by defining NLP and discussing the core principles that enable machines to understand, interpret, and generate human language. NLP leverages techniques from computational linguistics, machine learning, and deep learning to process large volumes of text and speech. We will then examine the wide range of NLP applications, such as machine translation, sentiment analysis, and virtual assistants, that are increasingly integrated into daily life. Following this, we will focus on Language Modeling, which involves training models to predict the structure and sequence of words in a given context. A key application of language modeling is Next Word Prediction, where models predict the next word in a sequence based on the preceding context. Finally, we will discuss the challenges currently facing NLP and explore future directions, including the latest advancements in AI and machine learning technologies that are shaping the future of NLP.

1.1 Natural Language Processing Definition and Principles :

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on enabling machines to understand, interpret, and generate human language. It combines techniques from computational linguistics, machine learning, and deep learning to process large volumes of text and speech data. Human language is inherently ambiguous, context-dependent, and constantly evolving, making NLP both a challenging and crucial area of AI research. Through advanced NLP systems, researchers aim to bridge the gap between human communication and computational understanding, allowing for more natural and efficient interactions between humans and machines.

NLP is an interdisciplinary field that draws from computer science, linguistics, and artificial intelligence. The primary goal of NLP is to build systems that can process and generate human language in a way that is both meaningful and contextually relevant. These systems are used in a wide range of applications, such as sentiment analysis, machine translation, text summarization, and conversational agents, all relying on sophisticated algorithms and large-scale language models to understand and generate human language accurately.

1.1.1 Principles of Natural Language Processing:

NLP operates based on several core principles, each playing a crucial role in language comprehension and generation:

1. **Speech and Text Processing**:
- Spoken language is transformed into text through speech recognition technologies.
- Written language is processed using computational techniques to extract meaningful patterns and insights.

2. **Tokenization**:
- Text is broken down into smaller units known as tokens, which can represent words, phrases, or symbols, allowing for structured linguistic analysis.

3. **Normalization**:
- Text is standardized by converting it to lowercase, removing punctuation, and handling special characters to ensure consistency in processing.

4. **Syntax Analysis**:
- NLP models analyze sentence structures to determine grammatical roles such as subjects, verbs, and objects, facilitating more accurate interpretation.

5. **Semantic Analysis**:
- The meaning of words and sentences is determined by examining relationships between concepts, synonyms, and contextual clues.

6. **Contextual Analysis**:
- Text is analyzed within a broader context, considering cultural, temporal, and situational factors to improve comprehension.

7. **Machine Learning and AI**:
- NLP systems utilize deep learning techniques and neural networks to improve language modeling, enabling better predictive capabilities and nuanced understanding.

1.1.2 Challenges in Natural Language Processing:

Despite advancements in NLP, several challenges remain:
- **Ambiguity**: The same phrase or word can have multiple meanings depending on context, requiring sophisticated disambiguation methods.
- **Linguistic Variability**: Differences in grammar, syntax, and vocabulary across languages and dialects complicate NLP development.
- **Cultural and Contextual Sensitivity**: Language carries cultural nuances that affect meaning and interpretation, making it difficult for models to generalize across different populations.
- **Data Limitations**: Training robust NLP models requires vast amounts of high-quality data, which is often unavailable for low-resource languages or specialized fields.

1.1.3 The Application of NLP:

NLP is used in a wide range of applications, each addressing specific challenges and opportunities. Below are some of the most prominent examples:

1.**Sentiment Analysis:**

This involves determining the emotional tone behind a piece of text, whether it's positive, negative, or neutral. This is used to understand customer feedback, monitor brand reputation, and gauge public opinion.

2. **Classification:**

This is the process of categorizing text into predefined groups or categories. Examples include spam detection, topic classification, and intent detection.

3.**Chatbots Virtual Assistants:**

These use NLP to understand and respond to user queries in a conversational manner, providing information, completing tasks, and offering support.

4.**Text Extraction:**

This involves automatically identifying and extracting specific pieces of information from text, such as names, dates, locations, or key phrases.

5.**Machine Translation:**

This is the use of computers to automatically translate text or speech from one language to another, breaking down communication barriers.

6.**Market Intelligence:**

NLP is used to analyze market data, such as news articles, social media posts, and customer reviews, to identify trends, understand competitor activities, and gain insights into customer preferences.

7.**Speech Recognition:**

This technology converts spoken language into written text, enabling voice commands, transcription services, and voice assistants.

8.**Hiring Recruitment:**

NLP can be used to analyze resumes, job descriptions, and social media profiles to identify the best candidates for a job, automate screening processes, and reduce bias.

9.**Email Filters:**

These use NLP to automatically categorize and filter incoming emails, separating spam from important messages and routing emails to the appropriate recipients.

10.**Customer Support:**

NLP powers AI-driven customer support tools that can understand customer inquiries, provide relevant solutions, and offer personalized assistance.

1.1.4 Impact of NLP on Industries

The impact of Natural Language Processing (NLP) on various industries is profound, revolutionizing operations, enhancing decision-making, and improving customer experiences. Below is an overview of how NLP is transforming key sectors, supported by scholarly references:

- Healthcare:

NLP has become an indispensable tool in healthcare, aiding in the processing and analysis of vast amounts of unstructured data from electronic medical records (EMRs). This facilitates improved diagnosis, treatment planning, and patient care. For instance, NLP techniques are used to extract valuable information from clinical notes, enabling better clinical decision support and research outcomes [6] .

- Finance:

the financial sector, Natural Language Processing (NLP) has become instrumental in analyzing unstructured data sources such as financial reports, news articles, and customer feedback. This analysis aids in extracting insights into market trends and public sentiment, which are crucial for informed investment decisions and assessing market movements. For instance, a study by Wang et al. (2024) demonstrates the effectiveness of NLP-based models in identifying and predicting potential risks in financial documents and communications, thereby enhancing risk management strategies [23].

- Legal:

Legal professionals utilize NLP to automate the analysis of legal documents, contracts, and case law, reducing manual workloads and increasing efficiency. This automation aids in tasks such as contract review and legal research, allowing lawyers to focus on more complex legal analysis. For example, NLP techniques can extract pertinent information from legal texts, facilitating quicker and more accurate legal processes [10].

- Manufacturing:

NLP is applied in manufacturing for predictive maintenance and quality control. By analyzing textual data from maintenance logs and sensor data, NLP helps predict machine failures before they occur, ensuring better operational efficiency. This proactive approach minimizes downtime and enhances productivity [5].

- Business Operations:

NLP is transforming business operations by enabling the automation of customer service interactions through chatbots and virtual assistants. These technologies comprehend human language and respond to customer inquiries, providing instant support and improving customer satisfaction. For instance, NLP-powered chatbots can handle routine customer queries, allowing human agents to focus on more complex issues [13].

This table showcases the different ways companies are currently implementing NLP solutions across various departments.

Percentage	**Usage**
38%	Customer care
36%	Security
32%	Business development
30%	Sales
29%	Marketing
28%	Human resources or employee services
26%	Finance
25%	Supply chain or procurement
23%	Market research
20%	Corporate governance or ESG
18%	Legal or compliance

Tabel 1: How companies are using NLP solutions [8]

1.1.5 The 10 Biggest Issues for NLP :

If we're going to keep progressing in terms of the potential applications and overall capabilities of NLP, these are some of the most important issues we need to resolve:

1. Language differences

In the United States, most people speak English, but if you're thinking of reaching an international and/or multicultural audience, you'll need to provide support for multiple languages.

Different languages have not only vastly different sets of vocabulary, but also different types of phrasing, different modes of inflection, and different cultural expectations. You can resolve this issue with the help of "universal" models that can transfer at least some learning to other languages. However, you'll still need to spend time retraining your NLP system for each language.

2. Training data

At its core, NLP is all about analyzing language to better understand it. A human being must be immersed in a language constantly for a period of years to become fluent in it; even the best AI must also spend a significant amount of time reading, listening to, and utilizing a language. The abilities of an NLP system depend on the training data provided to it. If you feed the system bad or questionable data, it's going to learn the wrong things, or learn in an inefficient way.

3. Development time

Along similar lines, you also need to think about the development time for an NLP system.

To be sufficiently trained, an AI must typically review millions of data points. Processing all those data can take lifetimes if you're using an insufficiently powered PC. However, with a distributed deep learning model and multiple GPUs working in coordination, you can trim down that training time to just a few hours. Of course, you'll also need to factor in time to develop the product from scratch—unless you're using NLP tools that already exist.

This bar chart provides a comparison of the training time in hours for different NLP models, specifically BERT and GPT-3, when trained on varying numbers of GPUs and CPUs.

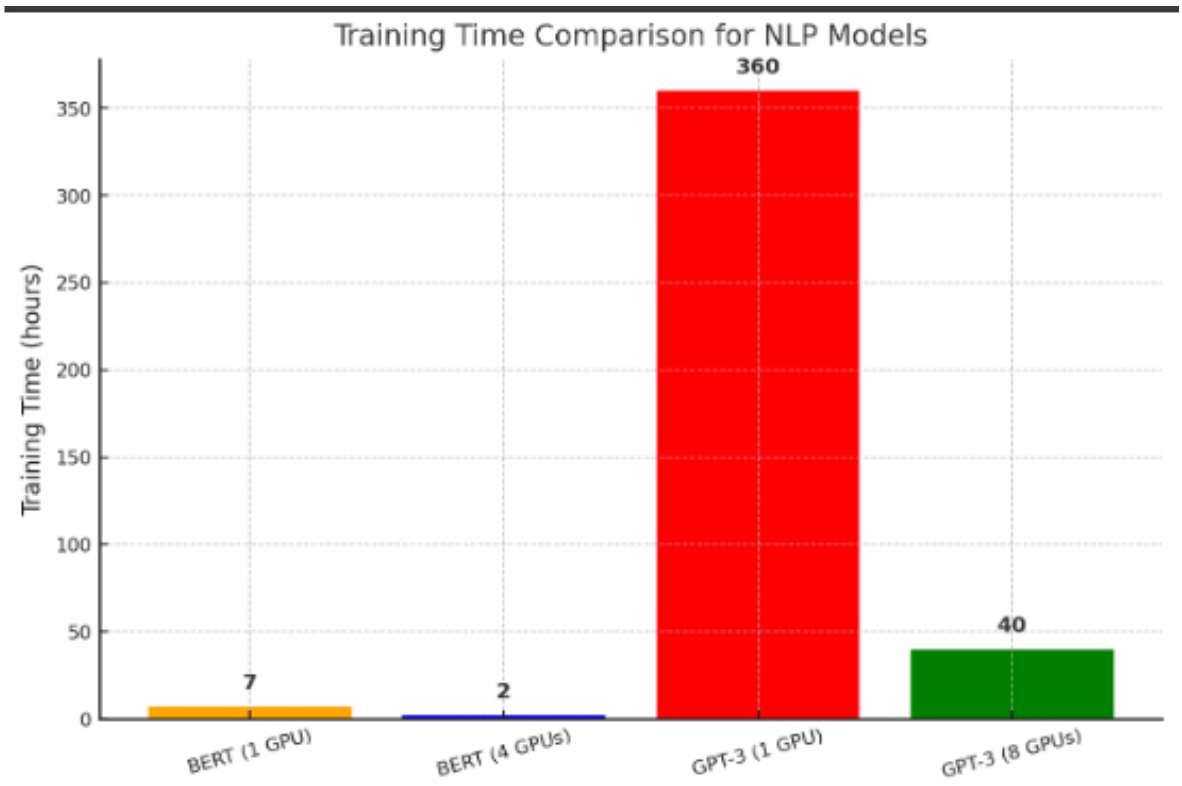

Figure 1 : Training time comparison for NLP models on different GPUs (Source: NVIDIA).

4. Phrasing ambiguities

Sometimes it's hard even for another human being to parse out what someone means when they say

something ambiguous. There may not be a clear concise meaning to be found in a strict analysis of their words. In order to resolve this, an NLP system must be able to seek context to help it understand the phrasing. It may also need to ask the user for clarity.

5. Misspellings

Misspellings are a simple problem for human beings. We can easily associate a misspelled word with its properly spelled counterpart, and seamlessly understand the rest of the sentence in which it's used. But for a machine, misspellings can be harder to identify. You'll need to use an NLP tool with capabilities to recognize common misspellings of words, and move beyond them.

6. Innate biases

In some cases, NLP tools can carry the biases of their programmers, as well as biases within the data sets used to train them. Depending on the application, an NLP could exploit and/or reinforce certain societal biases, or may provide a better experience to certain types of users over others. It's challenging to make a system that works equally well in all situations, with all people.

7. Words with multiple meanings

No language is perfect, and most languages have words that have multiple meanings. For example, a user who asks, "how are you" has a totally different goal than a user who asks something like "how do I add a new credit card?" Good NLP tools should be able to differentiate between these phrases with the help of context.

8. Phrases with multiple intentions

Some phrases and questions actually have multiple intentions, so your NLP system can't oversimplify the situation by interpreting only one of those intentions. For example, a user may prompt your chatbot with something like, "I need to cancel my previous order and update my card on file." Your AI needs to be able to distinguish these intentions separately.

9. False positives and uncertainty

A false positive occurs when an NLP notices a phrase that should be understandable and/or addressable, but cannot be sufficiently answered. The solution here is to develop an NLP system that can recognize its own limitations, and use questions or prompts to clear up the ambiguity.

10. Keeping a conversation moving

Many modern NLP applications are built on dialogue between a human and a machine. Accordingly, your NLP AI needs to be able to keep the conversation moving, providing additional questions to collect more information and always pointing toward a solution.[19]

1.1.6 Future Directions :

The field of NLP is constantly evolving, with new techniques and applications emerging regularly. Future advancements are expected to focus on improving the accuracy and efficiency of NLP systems, reducing biases, and expanding their capabilities to handle more complex tasks. As NLP continues to advance, its applications will become even more integrated into our daily lives, transforming the way we interact with technology.

1.2 Language Modeling

Language models (LMs) are statistical models that predict the probability of a sequence of words. They are a fundamental part of many NLP tasks, such as machine translation, text generation, speech recognition, and more. By understanding the structure of language, these models can predict what comes next in a sentence or sequence of words, making them crucial for creating applications that involve human language [12].

1.2.1 Types of Language Models:

1. **Statistical Language Models:**
These models rely on statistical methods to calculate the probability of a word or phrase appearing in a given context. The most basic form is the N-gram model, which predicts the probability of the next word based on the previous N words. While simple and interpretable, statistical models often struggle with long-range dependencies in language [20].

2. **Neural Network-based Language Models:**
Neural networks, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, helped improve language modeling by learning complex patterns over time. They are more adept at handling long-range dependencies than statistical models but are computationally intensive [2].

3. **Transformer-based Models:**
Transformer models, such as GPT (Generative Pre-trained Transformer), BERT (Bidirectional Encoder Representations from Transformers), and T5 (Text-to-Text Transfer Transformer), have significantly advanced the field of language modeling. These models use self-attention mechanisms to handle long-range dependencies and parallelize processing, making them highly efficient and effective for tasks like text generation, translation, and sentiment analysis [24].

1.2.2 Applications of Language Models:

1. **Machine Translation:**
LMs are essential for translating text from one language to another. Models like Google Translate rely on neural and transformer models to understand and translate context accurately [12].

2. **Text Generation:**
LMs power text generators like GPT, enabling them to create human-like content. These applications are used in writing assistants, content generation, and chatbots [20].

3. **Speech Recognition:**
Speech-to-text systems use language models to accurately transcribe spoken language into text, improving user experience in virtual assistants and voice-controlled devices [2].

4. **Question Answering Systems:**
NLP models help machines answer questions by analyzing and interpreting text, often used in virtual assistants and customer service bots [24].

1.2.3 Recent Advances in Language Modeling:

Recent breakthroughs in language modeling have been driven by deep learning, particularly with the advent of transformer models. BERT, GPT-3, and newer models such as T5 and GPT-4 are capable of generating highly coherent and contextually accurate text, pushing the boundaries of NLP applications. These models are trained on massive datasets and are fine-tuned for specific tasks to improve their performance. Additionally, techniques like zero-shot learning and few-shot learning are becoming increasingly important. These techniques allow models to perform tasks with little to no task-specific training data, making them more versatile and scalable [12].

1.2.4 Future Directions in Language Modeling:

The future of language modeling lies in improving the scalability, efficiency, and ethical considerations of these models. Researchers are focused on reducing the computational cost of training large-scale models,

improving model interpretability, and ensuring that language models do not perpetuate biases. Additionally, there is an increasing interest in multimodal models, which can understand both text and visual data, creating a more comprehensive understanding of the world. Moreover, models that can adapt to user-specific contexts and generate more personalized responses are expected to be a significant focus, enabling more efficient and user-friendly AI systems [20].

1.3 Next Word Prediction

Next word prediction is a fundamental task in natural language processing (NLP) that focuses on predicting the most likely word to follow a given sequence of words. This task is central to many applications, such as autocomplete systems, virtual assistants, and text generation tools. By accurately predicting the next word, these systems enhance user experience, improve efficiency, and enable more natural interactions between humans and machines.

Next word prediction involves estimating the probability of the next word in a sequence based on the preceding words. For example, given the sequence "The cat sat on the," a next word prediction model might suggest "mat" or "chair" as the most likely continuation. This task relies on understanding the context and structure of the sentence to make accurate predictions.

1.3.1 Approaches to Next Word Prediction:

Several approaches are used to tackle next word prediction, ranging from simple statistical methods to advanced neural network architectures:

1. **N-gram Models:**

- N-gram models predict the next word based on the frequency of previous word sequences in a dataset. For example, a trigram model uses the previous two words to predict the next one.
- While simple and effective for small datasets, N-gram models struggle with long sequences and rare word combinations.

This diagram figure 2 illustrates the concept of N-grams (Uni-gram, Bi-gram, and Tri-gram) as they relate to understanding word sequences for next word prediction models.

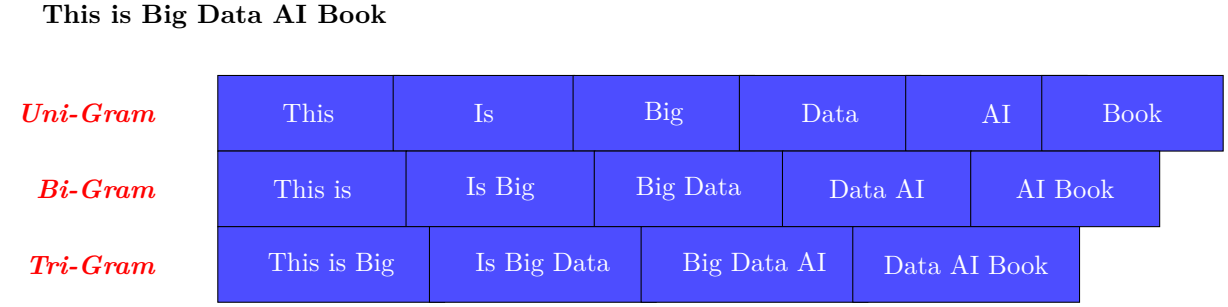

Figure 2 : Understanding N-grams in Next Word Prediction

‘

2. **Neural Network Models:**

- Neural networks, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, are widely used for next word prediction. These models capture complex patterns in sequences by maintaining a hidden state that encodes information about previous words.
- They are particularly effective for handling long-range dependencies and large datasets.

3.**Transformer-Based Models**:

Transformer-based models, such as the Generative Pre-trained Transformer (GPT), have revolutionized next-word prediction tasks in natural language processing. These models employ self-attention mechanisms, allowing them to assess the significance of different words within a sequence, thereby generating contextually accurate predictions [15].

This self-attention capability enables the model to focus on various parts of the input sequence, efficiently capturing long-range dependencies. Furthermore, transformers are adept at handling large-scale datasets, producing outputs that are both contextually relevant and coherent. Their architecture facilitates parallel

processing, making them more efficient than traditional sequential models like recurrent neural networks (RNNs) [17].

1.3.2 Applications of Next Word Prediction:

Next word prediction is used in a variety of real-world applications, including:
- **Autocomplete Systems**: Suggesting the next word as a user types, improving typing speed and reducing errors.
- **Virtual Assistants**: Enhancing conversational agents by predicting the most likely response or query.
- **Text Generation**: Assisting in creating coherent and contextually relevant text for content creation or storytelling.

This graphic demonstrates how an AI model utilizes the context of a sentence ("It rains a lot in the") to predict potential next words, showcasing the importance of context in intelligent text generation.

Figure 3 : Understanding Context for Smarter Text Generation

- **Speech Recognition**: Improving the accuracy of speech-to-text systems by predicting the most likely sequences.

This illustration depicts the process of speech recognition, where spoken words are converted into a digital format by a mobile device and can then be used to interact with various applications.

1.3.3 Challenges in Next Word Prediction:

Despite its widespread use, next word prediction faces several challenges:
- **Contextual Ambiguity**: Sentences can have multiple valid continuations depending on the context, making it difficult to predict the most appropriate word.
- **Handling Rare Words**: Predicting words in rare or unseen sequences remains a challenge, especially for models trained on limited datasets.
- **Computational Complexity**: Advanced models, such as transformers, require significant computational resources for training and inference.
- **Bias in Predictions**: Models can inherit biases from the training data, leading to unfair or inappropriate predictions.

1.3.4 Recent Advances in Next Word Prediction:

Recent advancements in next word prediction have focused on improving accuracy, efficiency, and scalability:
- **Large-Scale Pretraining**: Models like GPT-3 are pretrained on massive datasets, enabling them to generate highly accurate predictions across diverse contexts.

Figure 4 : Speech Recognition

- **Few-Shot Learning**: Modern models can make accurate predictions with minimal task-specific training data, thanks to their ability to generalize from pretrained knowledge.
- **Efficiency Improvements**: Techniques like model pruning and quantization have made it possible to deploy advanced models on resource-constrained devices.

1.3.5 Future Directions:

The field of next word prediction continues to evolve, with ongoing research focusing on:
- **Contextual Understanding**: Developing models that better understand the broader context of sentences to improve prediction accuracy.
- **Multimodal Integration**: Combining text with other modalities, such as images or audio, to create more versatile prediction systems.
- **Ethical AI**: Addressing issues related to bias, fairness, and the responsible use of the next word prediction models.

Next word prediction remains a critical area of research in NLP, with the potential to transform how we interact with technology and process information.

1.4 Conclusion

This study explored Natural Language Processing (NLP), focusing on language modeling and next-word prediction. We examined the transition from traditional methods like N-grams to modern transformer architectures and discussed the challenges of computational complexity.

CHAPTER II

2 Recurrent Neural Networks and Their Applications

Natural Language Processing (NLP) has become a key area of artificial intelligence, particularly in tasks involving *sequential data* like sentences and speech, where understanding the order of elements is crucial.

In Chapter 1, we covered next-word prediction using traditional models like N-grams and the rise of Transformers. Before attention-based models, **Recurrent Neural Networks (RNNs)** were essential for handling temporal dependencies by allowing information to persist across time steps.

However, RNNs suffer from the *vanishing gradient problem*, hindering their ability to learn long-term dependencies. The *Long Short-Term Memory (LSTM)* network addresses this by incorporating gating mechanisms to improve learning.

This chapter provides an overview of machine learning and deep learning, explores the structure and challenges of RNNs, and examines their real-world applications, particularly in comparison with Transformers.

2.1 Overview of Machine Learning

Machine learning is a subset of artificial intelligence that focuses on the development of algorithms that enable computers to learn from data and improve their performance over time without being explicitly programmed. Rather than relying on predetermined rules, machine learning systems identify patterns in data and use these patterns to make predictions or decisions.

The fundamental concept behind machine learning is the ability to generalize from the input data, enabling models to apply learned insights to unseen data. This makes machine learning particularly powerful in applications across various fields such as healthcare, finance, autonomous driving, and marketing, where it is used for tasks such as prediction, classification, and decision-making.[18]

2.1.1 Types of Machine Learning

Machine learning can be categorized into several types based on the nature of the data and the learning approach:

- **Supervised Learning:** The algorithm learns from labeled data, where each input is paired with a correct output. It is widely used in applications such as spam detection and price prediction.
- **Unsupervised Learning:** The model is trained on unlabeled data and tasked with discovering hidden structures or patterns, such as grouping customers by purchasing behavior (clustering).
- **Semi-supervised Learning:** This method combines a small amount of labeled data with a large amount of unlabeled data, offering a balance between supervised and unsupervised learning.
- **Reinforcement Learning:** The model learns by interacting with an environment and receiving rewards or penalties. It is commonly used in robotics, game playing, and autonomous systems.

2.2 Deep Learning Fundamentals

Traditional machine learning relies on manual feature engineering followed by the application of algorithms for classification or prediction. In contrast, deep learning goes a step further by utilizing neural networks that automatically learn representations from raw data. This ability makes deep learning particularly effective for complex data such as images, text, and audio.

2.2.1 What is a Neural Network?

A neural network is a computational model inspired by the structure and function of the human brain. It consists of units, or neurons, organized into layers. Each neuron receives a set of inputs, applies a mathematical operation, and passes the result to neurons in the next layer. Through this process, neural networks can learn complex patterns and representations from data.

2.2.2 Components of a Neural Network

- **Neurons:** Basic units that receive input, apply weights and an activation function, then pass the result forward.
- **Layers:**
  - **Input Layer:** Takes the raw data.
  - **Hidden Layers:** Extract features and patterns.
  - **Output Layer:** Gives the final result.
- **Activation Function:** Adds non-linearity, helping the model learn complex patterns. Examples include ReLU and Sigmoid.

2.2.3 Importance of Deep Layers

When a neural network contains more than one hidden layer, it is referred to as a *deep neural network*. The presence of multiple layers enables the model to learn hierarchical representations. For example, in image recognition: the first layer may detect edges, the second shapes, and the third entire objects.

2.2.4 Neural Network Architecture

The architecture of a neural network forms the foundation of its functionality and performance. At its simplest form, the network is composed of three primary types of layers:

- **Input Layer:** This is the first layer of a neural network that receives the raw input data. Depending on the task, the input can be images (as pixel values), text (converted into numerical vectors using tokenization), or numerical data. It doesn't perform any computation—its job is to pass the data to the next layer.
- **Hidden Layers:** These are intermediate layers between the input and output layers. They perform a series of mathematical computations through neurons and activation functions to extract meaningful patterns, features, or representations from the input data. A deep network typically has multiple hidden layers, which allow it to model complex relationships.
- **Output Layer:** This is the final layer that produces the result of the neural network's processing. For classification tasks, it often uses functions like softmax to provide class probabilities. For regression tasks, it may output a single value or a vector of continuous values. The output layer's format depends on the specific type of prediction required.

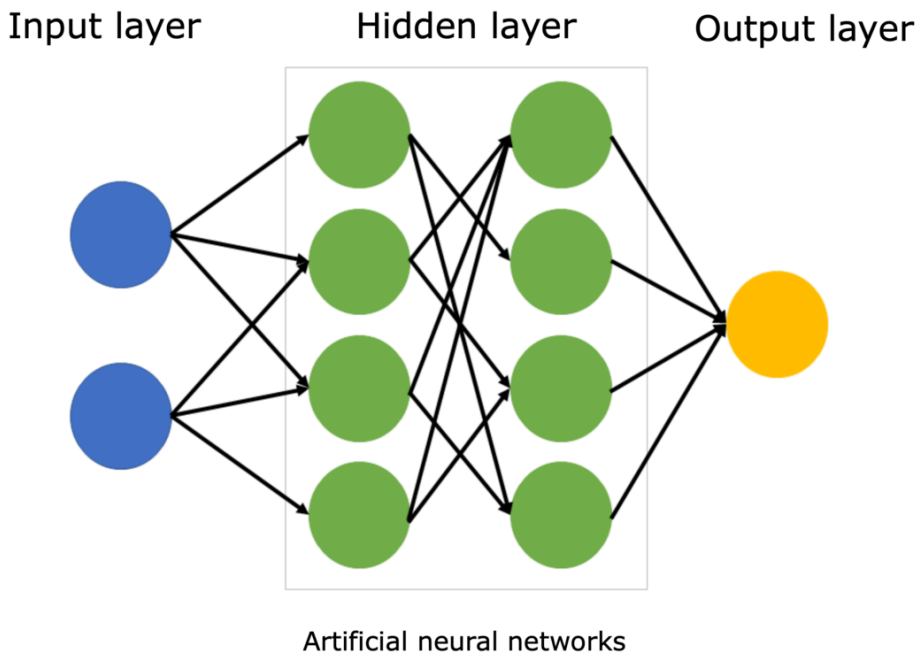

Figure 5 : A simple neural network with three layers.[21]

This diagram illustrates a simple artificial neural network with three layers: an input layer, a hidden layer, and an output layer, showcasing the basic structure of interconnected nodes that process information.

2.2.5 Perceptron

The perceptron is the simplest type of neural network and acts as a building block for larger models. It consists of a single neuron that computes a weighted sum of the input, adds a bias term, and applies an activation function to determine the output.

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1}$$

Where:

- x_i: Input features
- w_i: Weights
- b: Bias
- f: Activation function

2.2.6 Backpropagation

Backpropagation is one of the most important techniques used to train neural networks. It allows the network to update its weights by minimizing the error between the predicted and actual output. The process involves:

- Calculating the error at the output layer.
- Propagating this error backwards through the network.
- Updating the weights using gradients to reduce the loss in the next iteration.

2.2.7 Advantages and Challenges

2.3.7.1 Advantages of Deep Learning

Deep learning offers several benefits that distinguish it from traditional machine learning:

- **Automatic Feature Extraction:** Removes the need for manual feature engineering by learning features directly from raw data.
- **High Performance on Complex Tasks:** Excels in tasks like image recognition, natural language processing, and speech recognition.
- **Scalability with Data:** The performance often improves with more data, making deep learning suitable for big data applications.
- **Flexible and Scalable Architectures:** Supports a variety of model structures and sizes, from simple to very deep networks.

2.3.7.2 Challenges of Deep Learning

Despite its advantages, deep learning also comes with notable challenges:

- **Large Data Requirements:** Requires vast amounts of labeled data for effective training.
- **High Computational Cost:** Demands significant computing resources, especially during training, often requiring GPUs or TPUs.
- **Interpretability Issues (Black Box):** It is often difficult to understand or explain how decisions are made by the network.
- **Risk of Overfitting:** Particularly when using deep architectures with small datasets or poor regularization.

2.3 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data. They are particularly useful for tasks involving time-dependent or contextual data, such as text, speech, or time-series data.

2.3.1 Difference Between Traditional Neural Networks and RNNs

In traditional Feedforward Neural Networks, there is no interaction between layers or even within the same layer across different timesteps, meaning that they are unable to store information from previous inputs.

In contrast, Recurrent Neural Networks (RNNs) feature a feedback loop where each unit is connected to itself at the next timestep, allowing the network to "remember" past inputs and use them to make predictions for future data points.

Unlike feedforward neural networks, which process inputs in a single direction without retaining information from previous inputs, recurrent neural networks incorporate loops within their architecture. These loops allow RNNs to maintain a memory of past inputs, enabling them to model sequential data and capture temporal dependencies effectively [14].

2.3.2 Applications of RNNs

- **Language Modeling:** Predicting the next word in a sentence or text.
- **Speech Recognition:** Converting audio signals into text.
- **Time-Series Forecasting:** Predicting stock prices, weather conditions, etc.

2.3.3 Mathematical Representation

Recurrent Neural Networks (RNNs) are mathematically represented by equations that describe how the network processes data over time. Each neuron in the network depends not only on the current input but also on previous hidden states, which allows the network to capture temporal dependencies.

The basic equation representing the relationship between inputs and the hidden state is as follows:

$$h_t = f(Wx_t + Uh_{t-1} + b) \tag{2}$$

Where:

- h_t is the hidden state at time t.
- x_t is the input at time t.
- W is the weight matrix associated with the input.
- U is the weight matrix associated with the previous hidden state.
- b is the bias term.
- f is the activation function, such as tanh or ReLU.

2.3.4 Types of RNN Architectures

There are several types of RNN architectures, each suited to different kinds of tasks. The choice of architecture depends on the problem being solved. Below are the main types:

1. One-to-One

This is the simplest type, where each input corresponds to one output, similar to traditional feedforward networks.

2. One-to-Many

In this architecture, the network takes one input and generates multiple outputs. An example application is converting text to speech.

3. Many-to-One

This architecture is used for tasks where multiple inputs are transformed into a single output, such as text classification.

4. Many-to-Many

This architecture is used in tasks that require multiple inputs and multiple outputs, such as machine translation.

5. Bidirectional RNN

Bidirectional Recurrent Neural Networks (BRNNs) are an extension of traditional RNNs that process data sequences in both forward and backward directions. This architecture allows the network to have information from both past and future contexts, which can be particularly useful for tasks where context from both directions is crucial, such as language processing, speech recognition, and time-series analysis

This figure illustrates three common Recurrent Neural Network (RNN) architectures: (a) Many-to-one, where a sequence of inputs maps to a single output; (b) One-to-many, where a single input generates a sequence of outputs; and (c) One-to-one, representing a standard feedforward network without the sequential processing capabilities of RNNs.

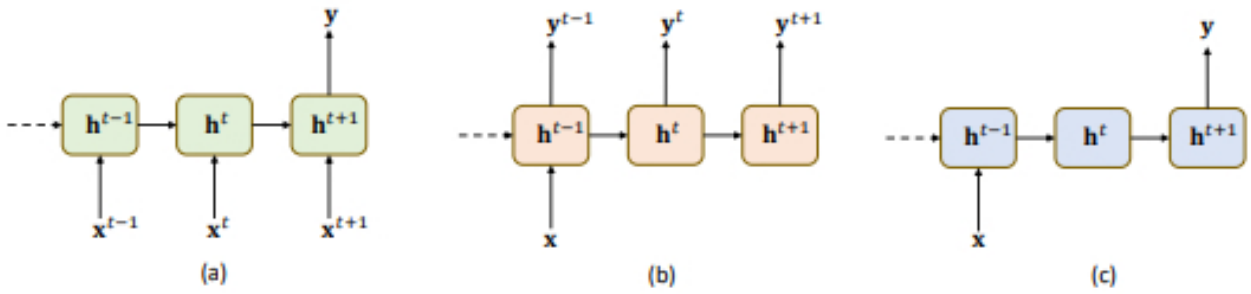

Figure 6: RNN architectures. (a) Many-to-one, (b) One-to-many, and (c) One-to-one [9]

2.3.5 Limitations of Standard RNNs

Despite their ability to handle sequential data, Recurrent Neural Networks (RNNs) have some limitations that affect their performance in certain applications:

1. Vanishing Gradient Problem

This issue arises when errors are propagated through many layers in the network, causing the gradients to gradually shrink, which prevents the network from effectively learning over long time sequences.

2. Difficulty in Remembering Long-Term Context

RNNs struggle to retain long-term dependencies in sequences, which limits their ability to capture relationships between distant inputs in tasks that require such memory.

3. Difficulty in Training

Due to their recurrent nature, training RNNs can be much more difficult and time-consuming compared to traditional feedforward networks. Additionally, tuning the model parameters effectively is challenging.

2.4 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Networks (RNNs) designed to address the vanishing gradient problem, which hampers the learning of long-term dependencies in standard RNNs. Introduced by Hochreiter and Schmidhuber, LSTMs incorporate gating mechanisms—specifically input, forget, and output gates—that regulate the flow of information, enabling the network to retain relevant information over extended sequences. This architecture allows LSTMs to effectively model temporal data where context from earlier inputs is crucial for accurate predictions [7].

2.4.1 Why Did LSTM Emerge?

Long Short-Term Memory (LSTM) networks were introduced to address the limitations of traditional Recurrent Neural Networks (RNNs) in learning long-term dependencies. Standard RNNs struggle with the vanishing gradient problem, which hampers their ability to retain information over extended sequences. LSTMs mitigate this issue through a unique architecture comprising memory cells and gating mechanisms—input, forget, and output gates—that regulate the flow of information. This design enables LSTMs to effectively capture and maintain long-range dependencies, making them well-suited for tasks such as machine translation and text generation [7].

2.4.2 How Does LSTM Address the Vanishing Gradient Problem?

LSTM relies on a unique cell structure that includes gates (such as the input, forget, and output gates) to regulate the flow of information. This structure allows the network to ”forget” irrelevant information and ”remember” important details, thus improving the network’s ability to handle long sequences of data.

2.4.3 LSTM Cell Architecture

The LSTM cell is the fundamental building block of an LSTM network. It is distinguished from traditional neural network cells by its use of several gates that control the flow of information over time. These gates allow the cell to ”forget” or ”remember” information depending on the temporal sequence of data.

Components of the LSTM Cell

- **Cell State:** The cell state is the primary path that carries information through time. It represents the long-term memory of the network.
- **Forget Gate:** The forget gate decides what portion of the information in the cell state will be discarded. It uses a sigmoid activation function, where values range from 0 to 1. A value of 0 means ”forget” and 1 means ”retain”.
- **Input Gate:** This gate controls which new information will be added to the cell state. It is computed using both a sigmoid function and a tanh activation function to determine the update values.
- **Output Gate:** The output gate determines whether the cell state will contribute to the output. It depends on the current input and the state of the cell.

This diagram figure 7 details the architecture of a Long Short-Term Memory (LSTM) cell. It illustrates the flow of information through the cell state, hidden state, input data, and the various gates (Forget Gate, Input Gate, Output Gate) that regulate the information to prevent the vanishing gradient problem and enable the learning of long-range dependencies.

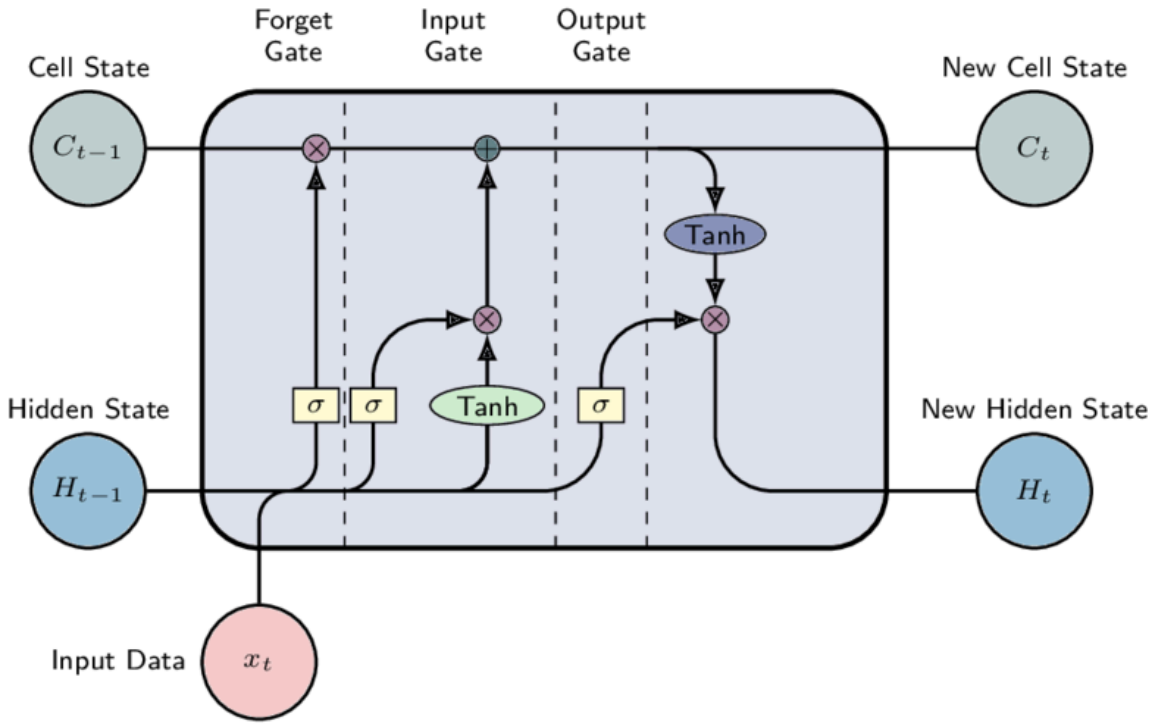

Figure 7 : LSTM cell architecture.[3]

2.4.4 Mathematical Formulation

The LSTM cell consists of several mathematical equations that govern the flow of information through the cells over time. The following equations describe how each component of the LSTM cell is computed.

Basic Equations:

– **Forget Gate:**

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3}$$

where:

* f_t is the forget gate.
* W_f is the weight associated with the forget gate.
* h_{t-1} is the output from the previous time step.
* x_t is the current input.
* b_f is the bias.

– **Input Gate:**

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

where:

* i_t is the input gate.
* W_i is the weight associated with the input gate.

  - b_i is the bias.
- **Cell State Update:**

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

  where:
  - $\tilde{C}_t$ is the new cell state.
  - W_C is the weight associated with the cell state.
  - b_C is the bias.
- **Cell State Update Equation:**

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{6}$$

  where:
  - C_t is the new cell state.
  - C_{t-1} is the previous cell state.
- **Output Gate:**

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

  where:
  - o_t is the output gate.
  - W_o is the weight associated with the output gate.
  - b_o is the bias.
- **Final Output:**

$$h_t = o_t * \tanh(C_t) \tag{8}$$

  where:
  - h_t is the final output.

2.4.5 Variants of LSTM

There are several variants of the standard LSTM architecture that have been developed to address specific challenges or to improve performance in different tasks. Some of the most notable variants include:

- **Gated Recurrent Unit (GRU):** The GRU is a simplified version of the LSTM, combining the forget and input gates into a single update gate. It is computationally less expensive than the standard LSTM while still maintaining comparable performance in many tasks.
- **Peephole LSTM:** In this variant, the LSTM's gates are augmented with connections that allow them to "see" the cell state. This enables more precise control over the information flow and can lead to improved performance on certain tasks.
- **Bidirectional LSTM (BiLSTM):** This variant processes the input data in both forward and backward directions, using two separate LSTM layers. This approach is beneficial for tasks where the context of both past and future information is important, such as in speech recognition or machine translation.
- **Attention-LSTM:** The Attention-LSTM combines the LSTM architecture with attention mechanisms. The attention mechanism helps the model focus on specific parts of the input sequence, improving its ability to handle long sequences and identify relevant context.

2.5 Comparison with Transformer Models

In recent years, Transformer models have gained popularity in sequence modeling tasks, often outperforming traditional recurrent neural networks. This section compares RNNs, LSTM, and Transformers across various aspects.

2.5.1 Fundamental Differences

- **RNNs and LSTMs** process input data sequentially. Each output depends on the previous hidden state, making them slower to train and harder to parallelize.
- **Transformers**, on the other hand, use self-attention mechanisms to process the entire input sequence at once. This allows better handling of long-range dependencies and faster training on modern hardware.

This table provides a comparative overview of three key neural network architectures used in sequence modeling: Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformer models. It contrasts them across several aspects, including input processing, context handling, training speed, gradient issues, interpretability, suitability for long sequences, and hardware utilization.

Aspect	**RNN**	**LSTM**	**Transformer**
Input Processing	Sequential	Sequential	Parallel
Context Handling	Short-term	Long-term (relatively)	Very long-term via self-attention
Training Speed	Slow	Moderate	Fast (on GPU)
Gradient Issues	Vanishing/Exploding	Handled via gates	No such issue
Interpretability	Low	Medium	Medium to High (via attention maps)
Suitability for Long Sequences	Weak	Better	Best
Hardware Utilization	Poor (CPU friendly)	Moderate	Excellent (GPU/TPU optimized)

Table 2 : Comparison between RNN, LSTM, and Transformer models

2.5.2 When to Use Which Model?

RNN / LSTM are suitable when:

- The sequence length is short.
- Computational resources are limited.
- Real-time inference with low latency is required.

Transformer is preferred when:

- Working with long and complex sequences.
- Accuracy is more important than speed.
- Training on high-performance GPUs or TPUs is available.

2.5.3 Is RNN Obsolete?

Although Transformer models have become the state-of-the-art in many tasks, RNNs and LSTMs are still widely used in industry applications, particularly where simplicity and resource efficiency are key requirements.

2.6 Conclusion

In this chapter, we explored the evolution of Recurrent Neural Networks (RNNs), their challenges, and the advancements brought by Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). We discussed how these models addressed issues like the vanishing gradient problem and the emergence of attention mechanisms, particularly in Transformer models, which have revolutionized sequence processing.

- **RNNs and LSTMs** remain valuable for sequence-based tasks, especially with temporal data.
- **Transformers** excel in handling long-range dependencies and are preferred for large-scale applications.
- While newer models dominate many tasks, RNNs and LSTMs still serve essential roles in specific use cases, especially in resource-constrained settings.

Understanding these models is key to grasping the development of AI techniques and their practical applications. The future may see hybrid models that combine the strengths of RNNs and Transformers, improving efficiency and handling complex dependencies.

CHAPTER III

3 Design and implementation

In this chapter, we present the design and implementation of a model for next word prediction. The aim is to build a system that can accurately predict the following word based on a given sequence of words. We describe the required hardware and software, the architecture of the model, and the steps taken to implement and test it.

3.1 Software Requirements

3.1.1 Python Programming Language

Python is a fundamental component in the "AI: The Next World" project due to its simplicity and flexibility. It is a high-level programming language developed by Guido van Rossum in the late 1990s and has become widely used in fields such as artificial intelligence and machine learning. The project relies on Python because of its strong support for libraries like TensorFlow and PyTorch, which facilitate the efficient development of intelligent models and data processing. Its clean syntax and extensive ecosystem make it the ideal choice for implementing the technical aspects of this project.

Figure 8 : Python Programming Language official logo [22]

3.1.2 Computing Resources

In the development of the "AI: The Next World" project, Google Colaboratory (Colab) has played a vital role by providing a cloud-based environment ideal for training and testing deep learning models. Colab offers free access to powerful computing resources, including CPUs, GPUs, and even TPUs, which significantly accelerate the training process. Its collaborative features allow team members to share and edit notebooks in real time, supporting efficient teamwork. Additionally, Colab comes pre-installed with essential Python libraries such as TensorFlow, PyTorch, and Keras, enabling seamless integration and faster development of AI models.

Figure 9 :Google Colaboratory Official Logo [22]

3.1.3 Numpy

NumPy is essential in the "AI: The Next World" project, as it introduces a powerful data structure called ndarray (n-dimensional array), designed for efficient storage and manipulation of numerical data. Arrays can be created using various NumPy functions or by converting existing Python lists. NumPy also provides a wide array of mathematical functions for tasks like element-wise operations, linear algebra, Fourier transforms, and random number generation. These functions are implemented in optimized C and Fortran code, making NumPy fast and efficient even for large datasets, a crucial feature for handling complex AI models.

Figure 10 :NumPy official logo [22]

3.1.4 PyTorch Library

is a key framework in the "AI: The Next World" project, prized for its dynamic computation graph and intuitive API. It allows developers to construct and modify neural network architectures on-the-fly, facilitating rapid prototyping and debugging. PyTorch provides seamless integration with Python's data ecosystem (including NumPy) and features modules like torch.nn for building layers,

torch.optim for optimization algorithms, and torch.utils.data for data loading and preprocessing. Its strong community support and extensive ecosystem (e.g., torchvision, torchaudio) make it ideal for research as well as production deployments.

Figure 11 : PyTorch Library official logo[22]

3.2 Hardware Requirements

The project "AI: The Next Word" can run on a mid-range device with minimum specifications such as an Intel Core i5 (6th generation) or equivalent processor, 8GB of RAM, an integrated GPU or GTX 1050 at minimum, and at least 256GB of storage. These specs are sufficient for basic development and testing, but may be limiting when training AI models or handling large datasets.

we relied on a set of tools and frameworks that require reliable hardware for smooth performance during development and AI model training. Below are the specifications of the machine we used:

- **Processor:** AMD Ryzen 5 5600X.
- **Graphics Card:** NVIDIA RTX 3060 Ti.
- **RAM:** 16GB DDR4.
- **Storage:** 512GB SSD.

3.3 Hybrid Next-Word Prediction Model: DistilBERT + Bidirectional LSTM

This project combines **DistilBERT**'s contextual understanding with **BiLSTM**'s sequential processing to create a high-accuracy, efficient next-word prediction system.

3.3.1 Bidirectional LSTM

Traditional RNNs suffer from vanishing gradients, making them ineffective for long sequences. LSTMs

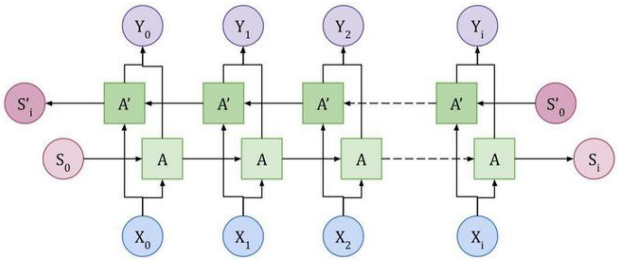

Figure 12 : Bidirectional LSTM layer Architecture

mitigate this with gating mechanisms, while Bidirectional LSTMs capture context from both past and future words, improving prediction accuracy.

3.3.2 DistilBERT

DistilBERT is a smaller, faster, and lighter version of BERT (Bidirectional Encoder Representations from Transformers), retaining 95% of its performance while being 40% smaller and 60% faster. It's ideal for next-word prediction when computational efficiency matters [16].

Transfer learning with DistilBERT leverages large-scale pre-trained knowledge to achieve high accuracy in next-word prediction tasks with minimal fine-tuning data and training time.

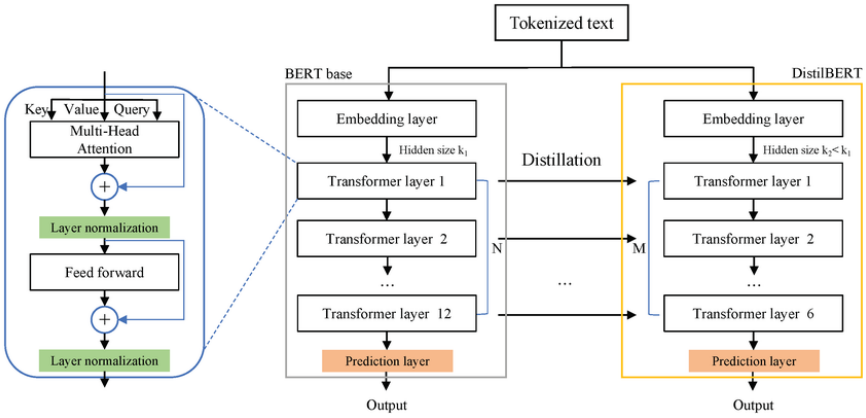

Figure 13 : The DistilBERT model architecture and components[1]

3.3.3 Methodology

The design process includes the following steps:

1. **Data Collection**: We train two specialized hybrid models (DistilBERT + BiLSTM) on distinct Kaggle datasets to address domain-specific linguistic patterns :
   - Dataset of Medium Articles CSV [11] containing more than 100k article titles, subtitles, and additional metadata.
   - Books CSV dataset [4] containing 50K book titles and authors (fiction, academic)

   These are used as input sequences for the model to learn contextual patterns in natural language.
2. **Data Preprocessing**:
   - Text is cleaned and standardized (e.g., lowercasing and punctuation removal).
   - A **tokenizer compatible with DistilBERT** is used to convert the input text into token IDs and attention masks.
   - Sequences are truncated or padded to a fixed length for batch processing.
3. **Model Architecture**:
   - The **DistilBERT encoder** processes the input tokens and generates contextualized embeddings for each token.
   - These embeddings are passed to a **BiLSTM layer**, which learns sequential dependencies by processing the sequence in both forward and backward directions.
   - The BiLSTM output is passed to a **Linear layer with a Softmax activation** to generate a probability distribution over the vocabulary.
   - The final predicted word corresponds to the token with the highest probability.
4. **Training**:
   - The model is trained end-to-end using *categorical cross-entropy loss*.
   - The *Adam optimizer* is used for efficient weight updates.
   - Dropout regularization is applied to prevent overfitting.
5. **Evaluation**:
   - The model is evaluated using metrics such as *accuracy* and *loss*.
   - A **confusion matrix** is also used to visualize prediction performance.
6. **Prediction**:
   - Once trained, the model can predict the most probable next word given an incomplete input sentence.
   - The input sentence is tokenized using the same DistilBERT tokenizer, passed through the trained model, and the output word is selected based on the highest probability score.

This architecture, combining **DistilBERT** and **BiLSTM**, allows the model to capture both deep semantic meaning and temporal word dependencies, making it well-suited for the task of next-word prediction.

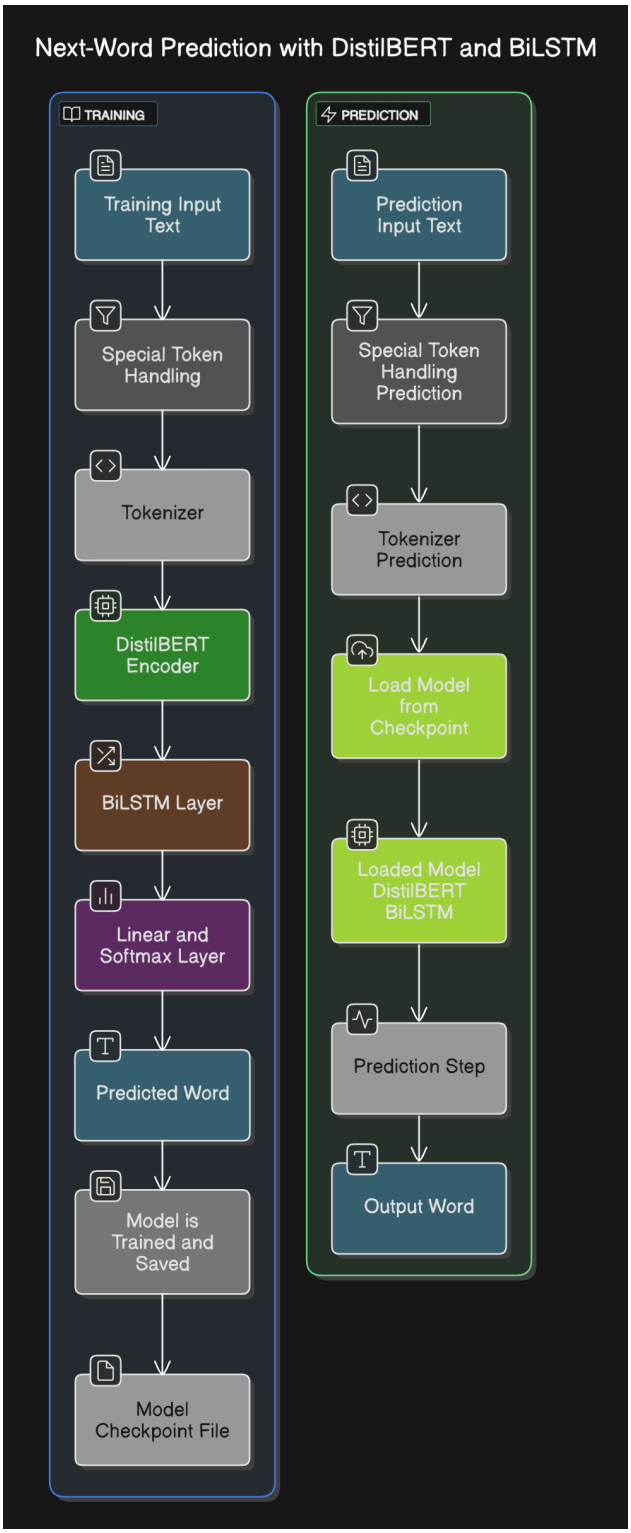

Figure 14 : Next-Word Prediction Model Architecture: Training and Prediction Flow.

3.4 Implementation of the Next-Word Prediction Model

3.4.1 Data Collection

The dataset used in this project was obtained from **Kaggle**, a popular online platform for data science and machine learning competitions. Kaggle also serves as a rich repository of datasets contributed by users across various domains.

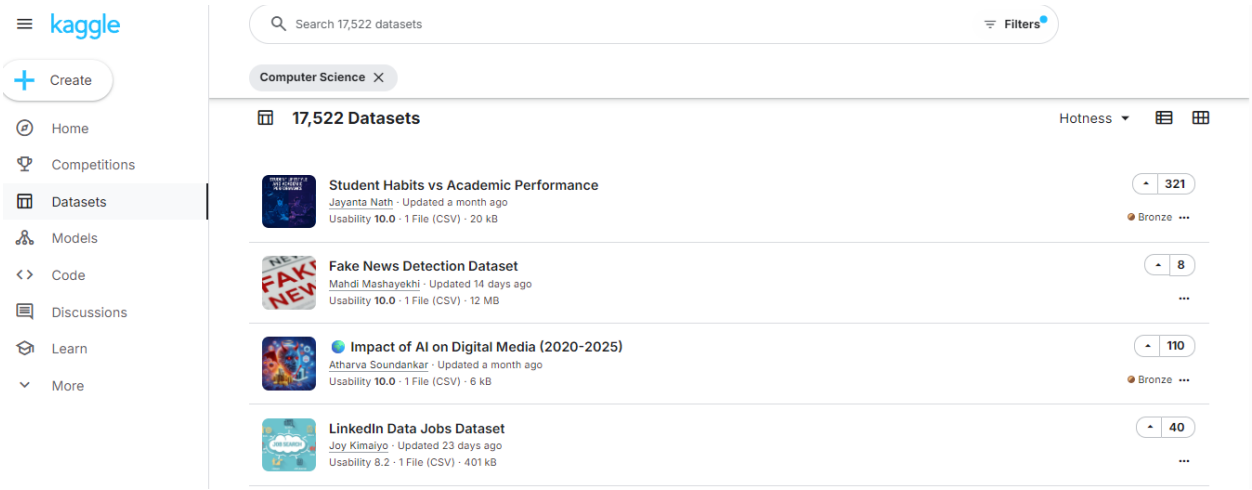

Figure 15 :Kaggle datasets

In this work, we downloaded a dataset containing metadata of articles published on the Medium platform, specifically focusing on the `title` field, which includes short, meaningful headlines.

The titles were used as the primary training data for our next-word prediction model. Since these titles are often concise yet semantically meaningful, they are well-suited for sequential language modeling tasks. Before feeding them into the model, we performed a data cleaning process to remove special characters such as non-breaking spaces (`\xa0`) and thin spaces (`\u200a`), ensuring compatibility with the tokenizer.

Throughout this project, the titles will be tokenized using the DistilBERT tokenizer and then passed into a transformer-based model combined with a bidirectional LSTM. The aim is to train a system capable of predicting the next word in a given sequence, simulating a form of language understanding based on real-world data.

Below is a sample from the dataset for the first model(next word prediction) :

- 1 https://towardsdatascience.com/a-beginners-guide-to-word-embedding-with-gensim-word2vec-model-5970fa56cc92 A Beginner's Guide to Word Embedding with Gensim Word2Vec Model 1.png 850 8 8 Towards Data Science 2019-05-30

- 2 https://towardsdatascience.com/hands-on-graph-neural-networks-with-pytorch-pytorch-geometric-359487e221a8 Hands-on Graph Neural Networks with PyTorch & PyTorch Geometric 2.png 1100 11 9 Towards Data Science 2019-05-30

- 3 https://towardsdatascience.com/how-to-use-ggplot2-in-python-74ab8adec129 How to Use ggplot2 in Python A Grammar of Graphics for Python 3.png 767 1 5 Towards Data Science 2019-05-30

Below is a sample from the dataset for the second model (Books titles):

```
ISBN,Book-Title,Book-Author,Year-Of-Publication,Publisher
0195153448,Classical Mythology,Mark P. O. Morford,2002,Oxford University Press
0002005018,Clara Callan,Richard Bruce Wright,2001,HarperFlamingo Canada
0060973129,Decision in Normandy,Carlo D'Este,1991,HarperPerennial
0374157065,Flu: The Story of the Great Influenza Pandemic of 1918 and the Search for the Virus That
0393045218,The Mummies of Urumchi,E. J. W. Barber,1999,W. W. Norton & Company
0399135782,The Kitchen God's Wife,Amy Tan,1991,Putnam Pub Group
```

We extracted and cleaned the book titles from this dataset and used them to train the second language model. The goal of this model was to guess or predict book names based on partial input. Since book titles are often short but contextually meaningful, they provide a unique linguistic structure that challenges the model to learn precise word associations.

3.4.2 Importing Libraries

The following Python libraries are imported:

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import DistilBertModel, DistilBertTokenizerFast
import pandas as pd
import numpy as np
from sklearn.model\_selection import train\_test\_split
import matplotlib.pyplot as plt
from sklearn.metrics import confusion\_matrix
import pickle
```

- `torch`, `torch.nn`: Used to build and train neural networks.
- `Dataset`, `DataLoader`: Handle data batching and loading during training.
- `DistilBertModel`, `DistilBertTokenizerFast`: Provide a pre-trained transformer (DistilBERT) and its tokenizer for converting text into numerical representations.
- `pandas`, `numpy`: Used for loading and manipulating structured data.
- `train_test_split`: Splits the dataset into training and testing subsets.
- `matplotlib.pyplot`: Used to plot evaluation results (e.g., loss curves).
- `confusion_matrix`: Helps evaluate model performance by comparing predictions to actual labels.
- `pickle`: Saves the trained model to a file for later use.

3.4.3 Loading Data

```
from google.colab import drive
drive.mount('/content/drive')

medium\_data = pd.read\_csv('/content/drive/MyDrive/medium\_data.csv')
medium\_data\['title'] = medium\_data\['title'].str.replace(u'\xa0', u' ').str.replace('\
    u200a', ' ')
```

- `drive.mount('/content/drive')`: Mounts the user's Google Drive to access files directly from Colab.
- `pd.read_csv('...')`: Loads the dataset `medium_data.csv` stored in Google Drive into a Pandas DataFrame.
- `medium_data['title'] = ...`: Cleans the `title` column by removing non-breaking spaces (`\xa0`) and narrow spaces (`\u200a`) which can interfere with text preprocessing.

3.4.4 Initializing Tokenizer and Transformer

```
tokenizer = DistilBertTokenizerFast.from\_pretrained("distilbert-base-uncased")
transformer = DistilBertModel.from\_pretrained("distilbert-base-uncased")
```

- `DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")`: Loads a fast version of the tokenizer for the pre-trained DistilBERT model. It converts raw text into token IDs suitable for model input.
- `DistilBertModel.from_pretrained("distilbert-base-uncased")`: Loads the DistilBERT transformer model with pre-trained weights. It generates contextualized embeddings for each token in the input sequence.

3.4.5 Dataset Class

```
class NextWordPredictordataset(Dataset):
def **init**(self, texts, tokenizer, max\_len):
self.encodings = tokenizer(texts.tolist(), padding='max\_length', truncation=True, max\
    _length=max\_len, return\_tensors='pt')

```
def __len__(self):
    return self.encodings['input_ids'].shape[0]

def __getitem__(self, idx):
    inputs = self.encodings['input_ids'][idx, :-1]
    masks = self.encodings['attention_mask'][idx, :-1]
    labels = self.encodings['input_ids'][idx, 1:]
    return inputs, masks, labels
```
```

- **Purpose:** This class prepares input sequences for next-word prediction from a list of text titles.
- `__init__(self, texts, tokenizer, max_len)`: Uses the tokenizer to convert text data into token IDs, with padding and truncation up to `max_len`. It stores the encoded tensors (input IDs and attention masks).
- `__len__()`: Returns the total number of samples in the dataset.
- `__getitem__(idx)`: For a given index:
  - `inputs`: All tokens except the last one.
  - `masks`: Corresponding attention mask, also excluding the last token.
  - `labels`: Target tokens, which are the input tokens shifted one position to the left (i.e., the "next word").

  This setup allows the model to learn to predict the next word based on the previous context.

3.4.6 Splitting Dataset

```
dataset = NextWordPredictionDataset(medium\_data\['title'], tokenizer, max\_seq\_len)
train\_size = int(0.8 \* len(dataset))
val\_size = len(dataset) - train\_size
train\_dataset, val\_dataset = torch.utils.data.random\_split(dataset, \[train\_size, val\
    _size])
train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)
val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size)
```

- `dataset = NextWordPredictionDataset(...)`: Creates a dataset instance using the titles from the Medium dataset and the tokenizer.

- `train_size`, `val_size`: Calculate the number of samples for training (80%) and validation (20%).
- `random_split(...)`: Randomly splits the full dataset into training and validation subsets.
- `DataLoader(...)`: Wraps the datasets into PyTorch DataLoaders that:
  - Shuffle the training data for better learning.
  - Divide the data into batches of size `batch_size`, which speeds up training and reduces memory usage.

3.4.7 Model Definition

```
class NextWordPredictor(nn.Module):
def **init**(self, transformer, hidden\_size, vocab\_size):
super().**init**()
self.transformer = transformer
self.bi\_lstm = nn.LSTM(input\_size=transformer.config.hidden\_size, hidden\_size=hidden\
    _size, num\_layers=1, bidirectional=True, batch\_first=True, dropout=0.3)
self.fc = nn.Linear(hidden\_size \* 2, vocab\_size)

```
def forward(self, input_ids, attention_mask):
    outputs = self.transformer(input_ids=input_ids, attention_mask=attention_mask)
    hidden_states = outputs.last_hidden_state
    lstm_out, _ = self.bi_lstm(hidden_states)
    logits = self.fc(lstm_out)
    return logits
```
```

- **Class: `NextWordPredictor`**: A PyTorch model that combines a pre-trained transformer with a bidirectional LSTM and a linear layer for next-word prediction.
- `transformer`: The DistilBERT model extracts contextual embeddings for each input token.
- `bi_lstm`: A bidirectional LSTM layer that processes the transformer outputs in both forward and backward directions, enhancing context understanding. It has:
  - `input_size`: Equal to the hidden size of DistilBERT.
  - `hidden_size`: User-defined number of LSTM units.
  - `bidirectional=True`: Enables learning from both past and future contexts.
  - `dropout=0.3`: Helps prevent overfitting.
- `fc`: A fully connected (linear) layer that maps the LSTM output to vocabulary-sized logits, representing the model's prediction for the next token.
- **Forward Pass**:
  - Input IDs and attention masks are passed to the transformer.
  - The last hidden state is sent to the BiLSTM.
  - The output from the LSTM goes through the linear layer to produce the final predictions.

3.4.8 Training Loop

We define loss, optimizer, and run training for 30 epochs. Accuracy and loss are recorded.

```
# Initialize model
model = NextWordPredictor(transformer, hidden_size=128, vocab_size=total_words)
model.to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5, weight_decay=1e-4)

# Lists to store metrics
```

```
train_losses = []
val_losses = []
val_accuracies = []
train_accuracies = []

# Training loop
for epoch in range(30):
    model.train()
    train_loss = 0
    correct_train = 0
    total_train = 0
    train_preds, train_labels = [], []

    for inputs, masks, labels in train_loader:
        inputs, masks, labels = inputs.to(device), masks.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs, masks)

        loss = criterion(outputs.view(-1, total_words), labels.view(-1))
        loss.backward()
        optimizer.step()
        train_loss += loss.item()

        _, predicted_train = torch.max(outputs, 2)
        correct_train += (predicted_train == labels).sum().item()
        total_train += labels.numel()

        train_preds.extend(predicted_train.cpu().numpy().flatten())
        train_labels.extend(labels.cpu().numpy().flatten())

    train_accuracy = 100 * correct_train / total_train

    model.eval()
    val_loss = 0
    correct = 0
    total = 0
    val_preds, val_labels = [], []

    with torch.no_grad():
        for inputs, masks, labels in val_loader:
            inputs, masks, labels = inputs.to(device), masks.to(device), labels.to(
                device)
            outputs = model(inputs, masks)

            loss = criterion(outputs.view(-1, total_words), labels.view(-1))
            val_loss += loss.item()

            _, predicted = torch.max(outputs, 2)
            correct += (predicted == labels).sum().item()
            total += labels.numel()

            val_preds.extend(predicted.cpu().numpy().flatten())
            val_labels.extend(labels.cpu().numpy().flatten())

    avg_train_loss = train_loss / len(train_loader)
    avg_val_loss = val_loss / len(val_loader)
    val_accuracy = 100 * correct / total

    train_losses.append(avg_train_loss)
    val_losses.append(avg_val_loss)
    val_accuracies.append(val_accuracy)
    train_accuracies.append(train_accuracy)

    print(f"Epoch {epoch+1}")
    print(f"Train Loss: {avg_train_loss:.4f}")
    print(f"Train Accuracy: {train_accuracy:.2f}%")
    print(f"Val Loss: {avg_val_loss:.4f}")
```

```
    print(f"Val Accuracy: {val_accuracy:.2f}%")
```

* **Model Initialization:**
  - The model `NextWordPredictor` is instantiated with the pre-trained transformer, a hidden size of 128, and the vocabulary size `total_words`.
  - The model is then moved to the appropriate device (GPU or CPU).
* **Loss Function and Optimizer:**
  - `CrossEntropyLoss`: Used as the loss function for multi-class classification (predicting the next word).
  - `AdamW`: A variant of the Adam optimizer with weight decay for regularization.
* **Training Loop:** The model is trained for 30 epochs, with the following steps in each epoch:
  - `model.train()`: Sets the model in training mode.
  - For each batch in the training dataset:
  - The inputs, attention masks, and labels are moved to the device.
  - The optimizer is reset with `optimizer.zero_grad()`.
  - The model makes predictions on the inputs.
  - The loss is calculated by comparing the model's outputs to the true labels.
  - The gradients are calculated using `loss.backward()` and the optimizer is updated with `optimizer.step()`.
  - The training loss and accuracy are computed by comparing predicted values to actual values.
* **Validation Loop:** The model is evaluated on the validation set after each epoch:
  - `model.eval()`: Switches the model to evaluation mode (disables dropout and other training-specific behaviors).
  - No gradients are calculated during validation using `torch.no_grad()` to save memory.
  - The loss and accuracy are computed for the validation set in the same way as for training, but without updating the model weights.
* **Metrics:**
  - The average training and validation losses, as well as the accuracy, are computed and stored for each epoch.
  - The results are printed at the end of each epoch to monitor the training progress.

Plotting Losses, Accuracies, and Confusion Matrices

```
# Plot losses and accuracies
plt.figure(figsize=(15, 4))

plt.subplot(1, 4, 1)
plt.plot(train_losses, label="Train Loss")
plt.title("Train Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.legend()

plt.subplot(1, 4, 2)
plt.plot(val_losses, label="Validation Loss", color="orange")
plt.title("Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.legend()

plt.subplot(1, 4, 3)
plt.plot(val_accuracies, label="Validation Accuracy", color="green")
plt.title("Validation Accuracy")
```

```
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.grid(True)
plt.legend()

plt.subplot(1, 4, 4)
plt.plot(train_accuracies, label="Train Accuracy", color="purple")
plt.title("Train Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy (%)")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()

# Confusion Matrix
train_cm = confusion_matrix(train_labels, train_preds)
val_cm = confusion_matrix(val_labels, val_preds)

plt.figure(figsize=(10, 7))
plt.subplot(1, 2, 1)
plt.imshow(train_cm, cmap='Blues', interpolation='nearest')
plt.title('Train Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('True')

plt.subplot(1, 2, 2)
plt.imshow(val_cm, cmap='Blues', interpolation='nearest')
plt.title('Validation Confusion Matrix')
plt.colorbar()
plt.xlabel('Predicted')
plt.ylabel('True')

plt.tight_layout()
plt.show()
```

* **Train Loss Plot:** This plot shows the loss calculated on the training dataset after each epoch. A decrease in the train loss suggests that the model is improving its performance on the training data.
* **Validation Loss Plot:** This plot shows the loss on the validation dataset. A rising validation loss might indicate that the model is overfitting to the training data.
* **Validation Accuracy Plot:** This plot tracks the percentage of correct predictions on the validation set for each epoch. An increasing trend in validation accuracy indicates better generalization.
* **Train Accuracy Plot:** Similar to the validation accuracy, this plot shows the model's performance on the training data. A high and steady increase suggests that the model is learning effectively.

These plots provide insights into how the model is learning and whether it's overfitting or underfitting.

Confusion Matrices Confusion matrices are also plotted for both training and validation sets. A confusion matrix is a useful tool for understanding how well a classification model is performing, as it shows the number of correct and incorrect predictions across different classes.

* **Train Confusion Matrix:** This matrix illustrates the true vs. predicted labels for the training data. It helps identify whether the model is misclassifying any particular class.
* **Validation Confusion Matrix:** Similar to the train confusion matrix, this one helps visualize the performance of the model on unseen data. It provides a detailed breakdown of how well the model generalizes to new examples.

Both confusion matrices are color-coded using the `Blues` colormap, where darker shades represent higher counts.

3.4.9 Saving Model and Tokenizer

```
#  Save model correctly using torch.save
torch.save(model.state_dict(), '/content/drive/MyDrive/next_word_predictor_model.pth')

#  Save tokenizer using pickle (this is okay)
with open('/content/drive/MyDrive/next_word_predictor_tokenizer.pkl', 'wb') as
    tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)
```

Once the model has been trained and evaluated, it is important to save it along with the tokenizer to avoid retraining and to enable easy deployment for future use. In this section, we demonstrate how to correctly save the trained model and tokenizer.

Saving the Model To save the model, we use the PyTorch method `torch.save`. This function allows us to store the model's learned parameters (weights) in a file. The following code saves the model's state dictionary (which contains all the learned parameters) to a file:

```
torch.save(model.state_dict(), '/content/drive/MyDrive/next_word_predictor_model.pth')
```

Here, `model.state_dict()` retrieves the weights of the trained model, and the file `next_word_predictor_model.pth` is stored in the specified directory. This saved model can later be loaded and used for inference.

Saving the Tokenizer The tokenizer is a crucial component for transforming text data into a format that the model can understand. Since tokenizers can vary depending on the architecture (e.g., BERT or DistilBERT), it is important to save it as well.

We use the `pickle` module to save the tokenizer. The `pickle.dump` function serializes the tokenizer and writes it to a binary file, allowing it to be loaded later for text preprocessing:

```
with open('/content/drive/MyDrive/next_word_predictor_tokenizer.pkl', 'wb') as tokenizer_file:
    pickle.dump(tokenizer, tokenizer_file)
```

This code saves the `tokenizer` object to the file `next_word_predictor_tokenizer.pkl`, which can be later used to tokenize new input data.

3.4.10 Second Model Implementation

After building and training the first model using Medium article data, we proceeded to build another model using a different dataset — a collection of book titles. The goal was to test the model's ability to adapt to short, context-rich phrases instead of full sentences or paragraphs.

In this second version, we applied several improvements over the first one, such as:

* **Early stopping:** to prevent overfitting and stop training once validation loss stopped improving.
* **More structured token masking:** we masked approximately 15% of tokens dynamically during training.
* **Evaluation after each epoch:** with accuracy and loss tracking for both training and validation sets.

The purpose of this model, unlike the first, is to guess the full book title based on partial input using masked language modeling. By training on book titles, the model learns to handle short, meaningful sequences and predict missing words within the context of actual book names.

```
model = DistilBertForMaskedLM.from_pretrained('distilbert-base-uncased')
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

optimizer = AdamW(model.parameters(), lr=5e-5, weight_decay=0.01)
```

Figure 16 : This figure shows the code used to define and load the language model (DistilBERT) for the second training setup.

3.5 Training Results

the first model
This graph give insight into how the model's error decreases during training, showing its learning progress.

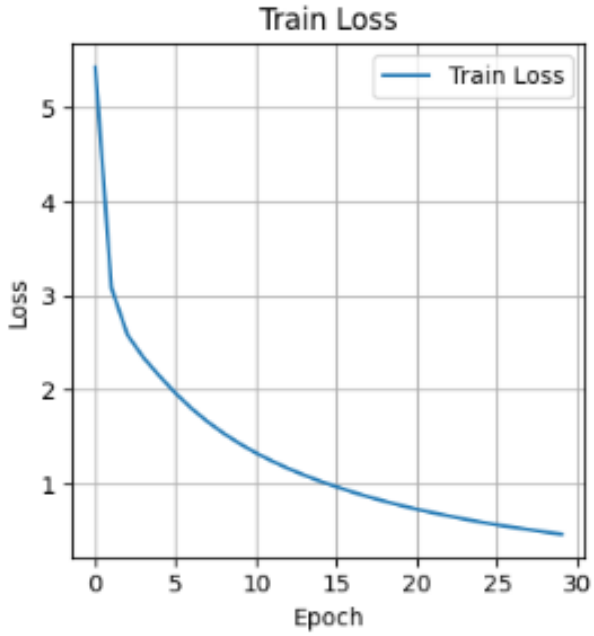

Figure 17 :Train Loss graph

This show how well the model generalizes to unseen data, giving an idea of how well it performs on the validation set.

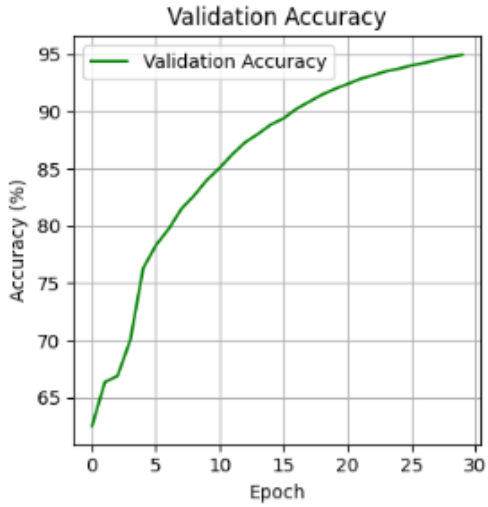

Figure 18 :Validation Accuracy graph

This graph shows the validation loss across training epochs. It demonstrates how the model improves during the initial stages, then plateaus, indicating convergence. Early stopping was used to halt training when no further improvement was observed.

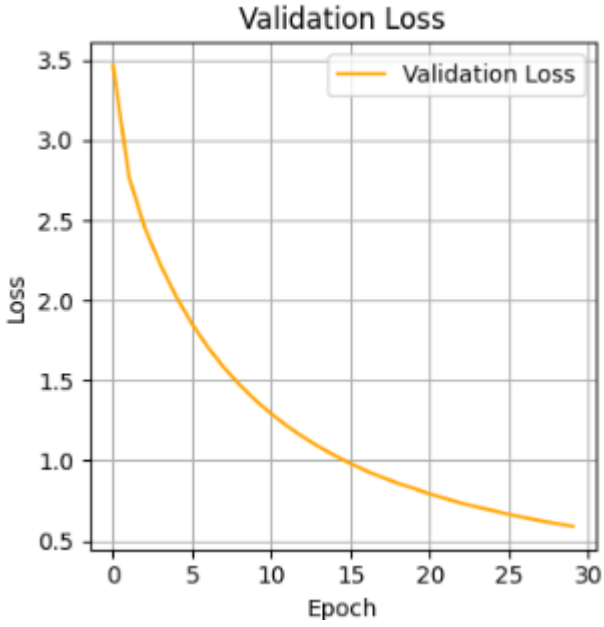

Figure 19 :validation loss graph

This graph shows the training accuracy over the course of epochs. It reflects the model's learning progress, with accuracy increasing steadily as the model becomes better at predicting masked tokens.

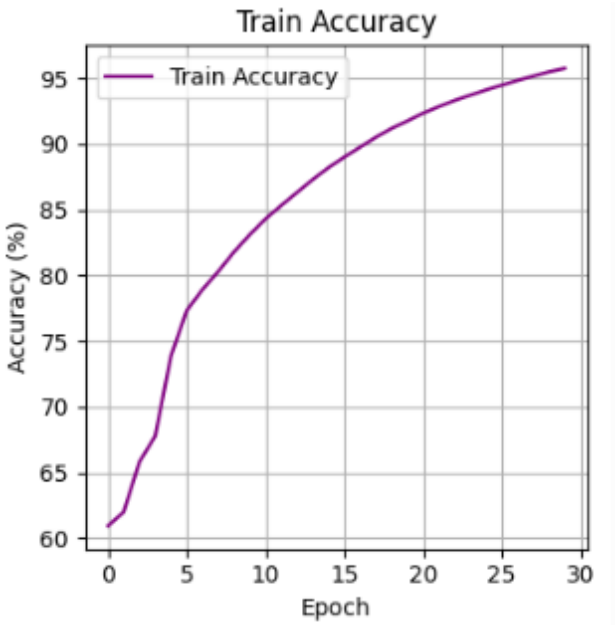

Figure 20 : graph training accuracy

Epoch 30 Performance Report

At this stage of training, the model achieved good and acceptable performance, reaching a **training accuracy of 95.56%** with a loss of **0.4658**, indicating effective learning on the training data.

On the validation set, the model attained an accuracy of **94.96%** with a loss of **0.5892**, reflecting good generalization to unseen data, despite a slight gap between training and validation loss.

Overall, the current performance is considered successful and satisfactory. We have completed the work and adopted this model for practical use. However, there remains potential for further improvements by continuing training or experimenting with new techniques in the future, but the current model meets the requirements well.

the second model

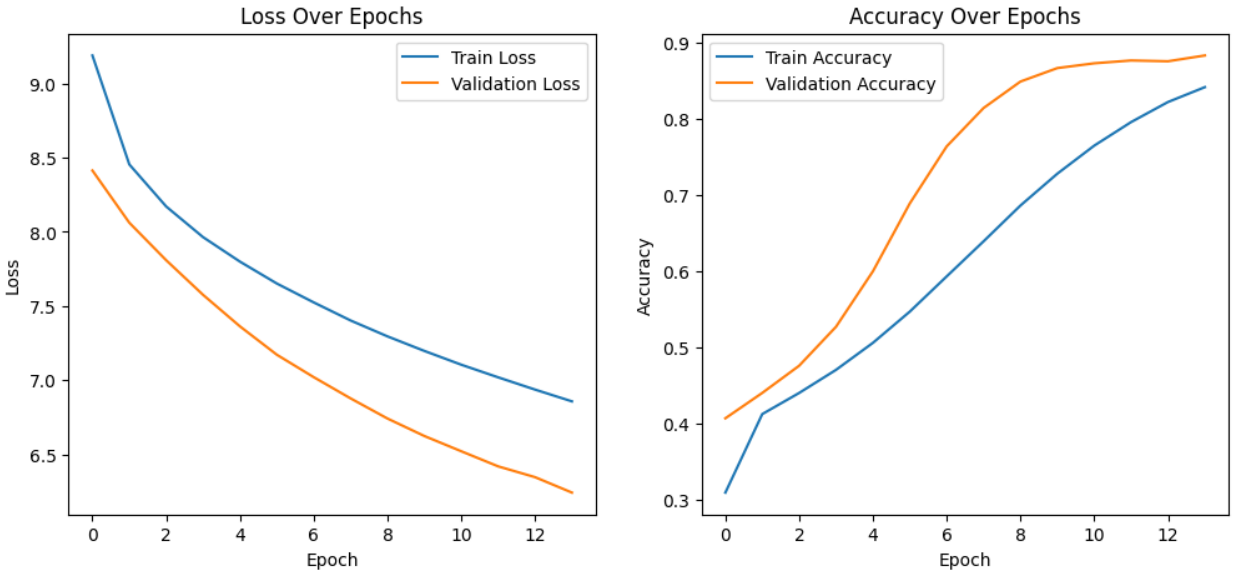

Figure 21: the second model training graphs

Epoch 14 Performance Report

At epoch 14, the model achieved solid performance, indicating effective training progress. The **training accuracy reached 84.21%**, with a **training loss of 6.8574**, suggesting the model has learned the patterns in the training data reasonably well.

On the **validation set**, the model attained an **accuracy of 88.37%** and a **validation loss of 6.2426**. The validation performance not only tracks well with the training metrics but even slightly outperforms it in accuracy, indicating good generalization and no immediate signs of overfitting.

Overall, the model's performance at this stage is strong and acceptable for deployment. While further training may yield marginal improvements, the current model meets the expected standards and is suitable for practical use. Future enhancements could explore deeper architectures or optimization strategies for refinement.

3.6 The Prediction Stage

In the final stage of our project, we focused on demonstrating the practical use of the next-word prediction model. While such models can be integrated into a variety of real-world systems — such as virtual assistants, smart text editors, or learning tools — we built a simple local web application to showcase its functionality.

We developed a minimal interface using HTML, CSS, and JavaScript. The trained model was deployed locally using a Python backend (e.g., Flask or FastAPI), running on localhost. When a user enters a sentence into the web interface, the frontend sends a request to the backend, which loads the model, performs the prediction, and returns the next-word suggestion to be displayed in the browser.

Although this deployment was limited to a local environment for testing purposes, it demonstrates how the model could be used in real-time applications. It also opens possibilities for integrating such predictive models into larger, cloud-based systems or production-ready platforms.

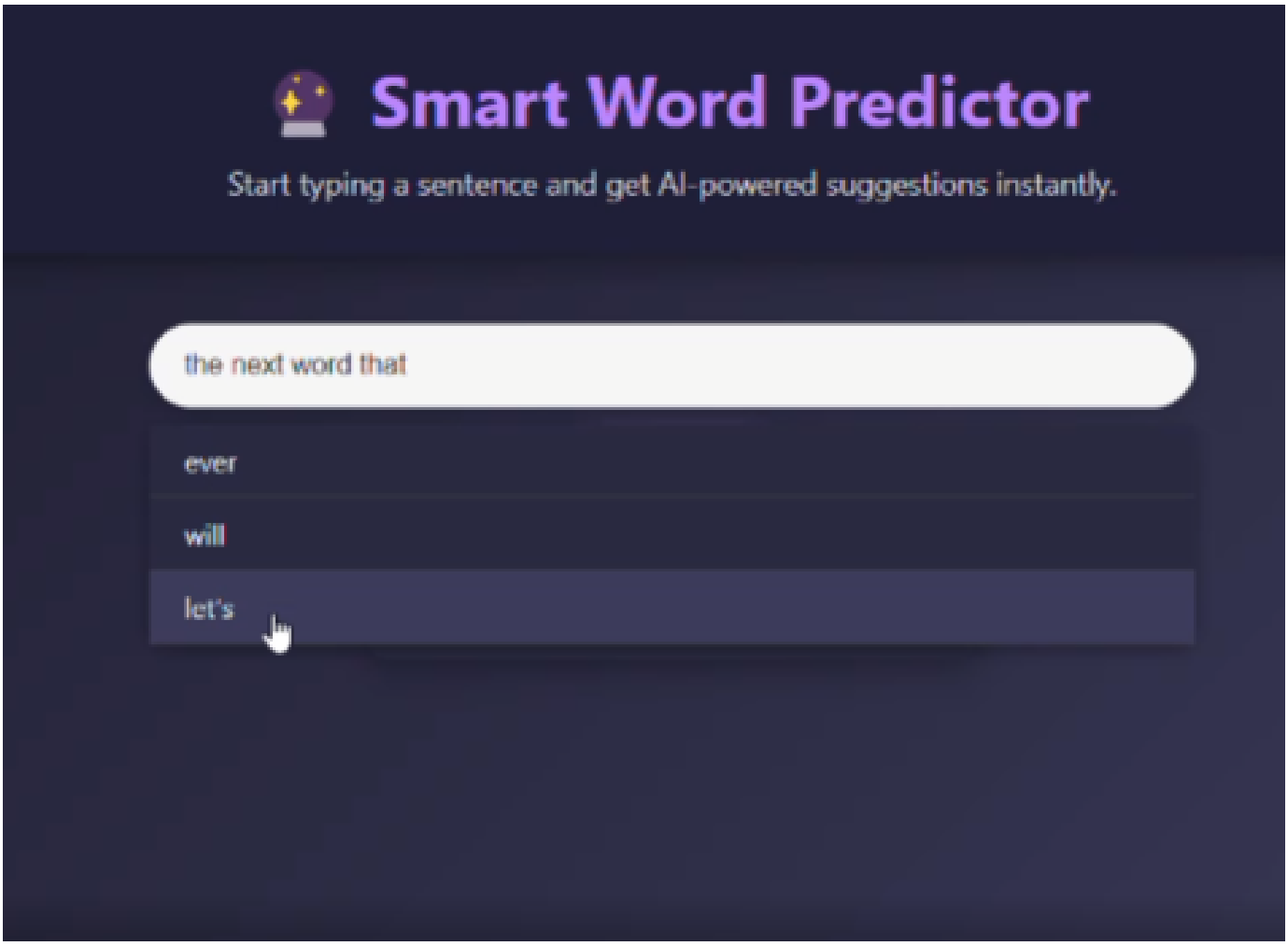

Figure 22: The first model predictions

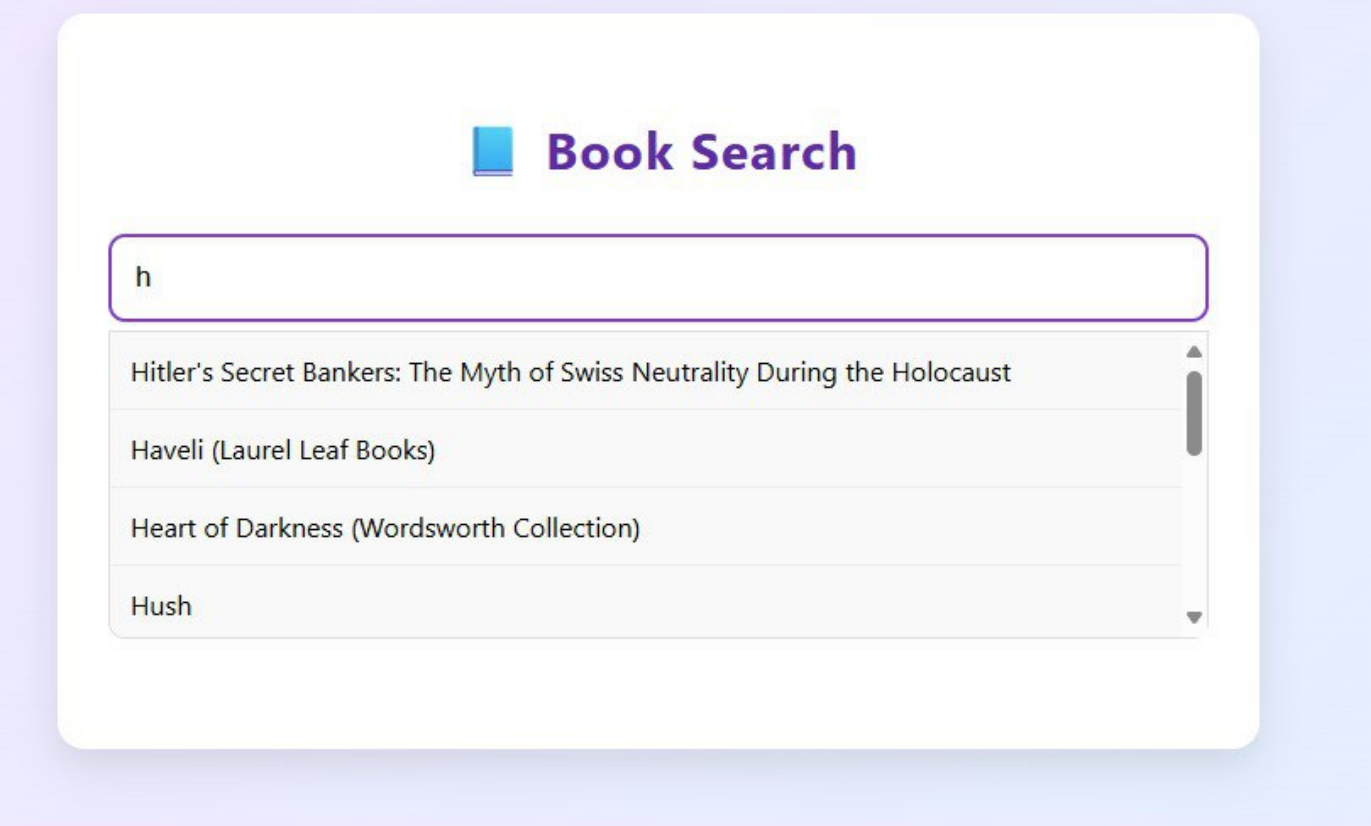

Figure 23: The second model predictions

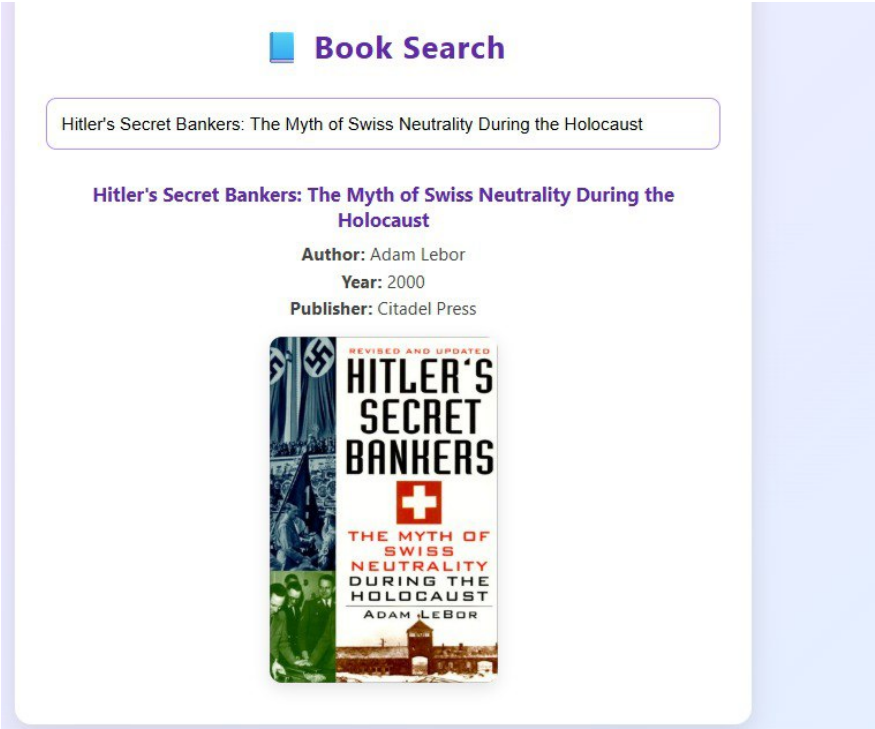

figure 24: The second model predictions

Those images show the web interfaces developed for two distinct language models: **next-word prediction** and **book title search**.

In the *next-word prediction* interface, users can enter a partial sentence, and the system predicts the most probable next word based on the trained model.

In the *book title search* interface, users can input keywords or short phrases to retrieve related book titles using semantic matching.

Both interfaces were built using **HTML, CSS, and JavaScript**, and are designed to run locally in a web browser, providing one of the best ways to test out our models.

As part of enhancing the practical application of our next-word prediction model, we developed a simple chatbot. This chatbot leverages the model's ability to predict the next word, enabling it to engage in basic conversations with users. By integrating the prediction model within the chatbot's backend, the system can generate contextually relevant responses in real time. This addition demonstrates how the model can be extended beyond standalone prediction to interactive applications, providing a foundation for more advanced conversational AI systems.

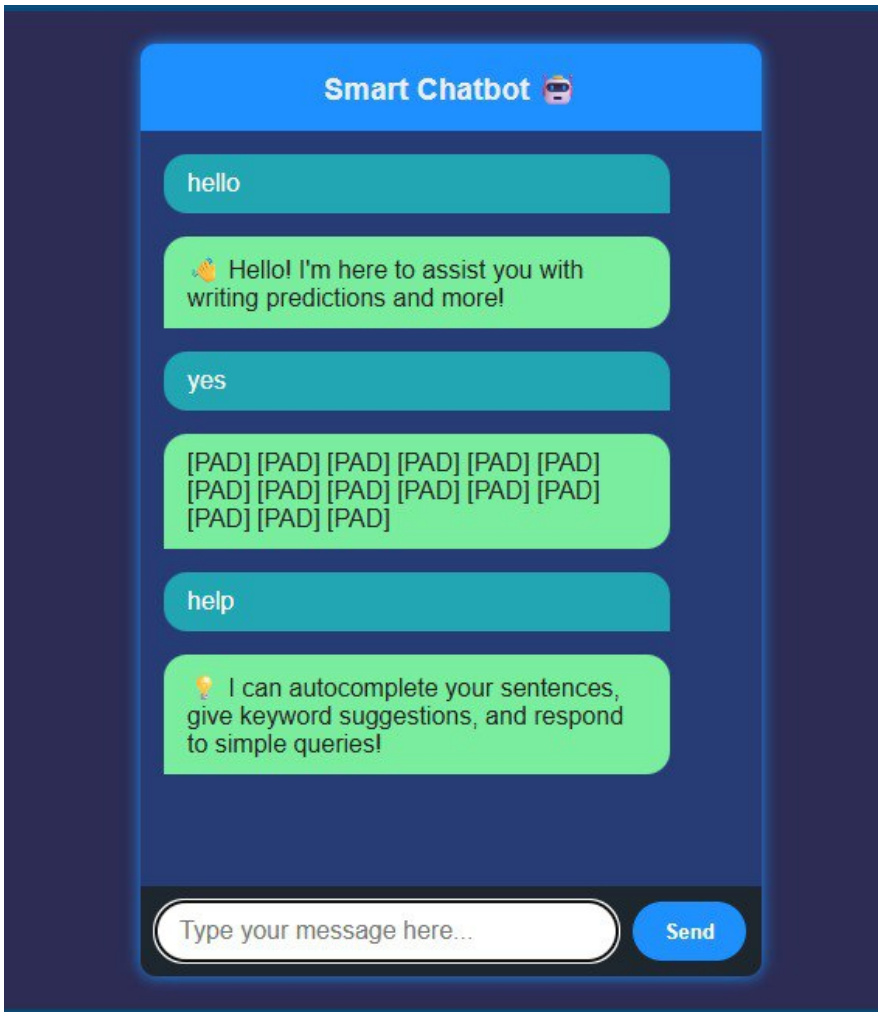

Figure 25: our simple chatbot using our first model

3.7 Conclusion

This chapter presented the design and implementation of the next-word prediction model. We designed a model based on DistilBERT, trained on two different datasets to improve accuracy and robustness. The implementation included training, validation, and early stopping techniques to optimize performance. Finally, a simple local web interface was developed to demonstrate the model's prediction capabilities. This work successfully bridges model design and practical application.

Conclusion

In this project, we studied and implemented a **Next Word Prediction** system, a fundamental task in **Natural Language Processing (NLP)**. We began by understanding the basics of NLP, including the challenges that arise when teaching machines to understand and generate human language. These challenges include ambiguity, context, grammar variations, and more.

We then explored **Language Models**, which are used to predict or generate the next word in a sequence of text. Traditional approaches like n-gram models were discussed, but we focused more on modern deep learning methods using **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks, which are better at capturing patterns in long text sequences.

As part of the project, **we built our own next-word prediction model**. We used preprocessed real-world text data and applied deep learning techniques, specifically combining **DistilBERT for tokenization and embeddings** with an **LSTM layer for sequence learning**. This model was trained on news headlines to predict the next most likely word. We also evaluated the model using accuracy and confusion matrices, and visualized the results to better understand its performance.

Through this work, we learned how to handle text data, how language models work, and how to implement and fine-tune deep learning models for language tasks. Although our model is a simplified version of what large companies use in products like autocomplete and voice assistants, it gave us valuable insights into how such systems function in practice.

In conclusion, this project combined theory and practice, and gave us hands-on experience with one of the most exciting applications of artificial intelligence today—teaching machines to understand and predict language.

References

[1] Hadeer Adel, Abdelghani Dahou, Alhassan Mabrouk, Mohamed Abd Elaziz, Mohammed Kayed, Ibrahim Mahmoud El-Henawy, Samah Alshathri, and Abdelmgeid Amin Ali. Improving crisis events detection using distilbert with hunger games search algorithm. *Mathematics*, 10(3):447, 2022.

[2] AltexSoft. Language models explained, 2024. Accessed: April 29, 2025.

[3] Eva Andrés, Manuel Pegalajar Cuéllar, and Gabriel Navarro. Brain-inspired agents for quantum reinforcement learning. *Mathematics*, 12(8):1230, 2024.

[4] Saurabh Bagchi. Books dataset. `https://www.kaggle.com/datasets/saurabhbagchi/books-dataset`, 2020. Accessed: YYYY-MM-DD.

[5] Margherita Bernabei, Silvia Colabianchi, and Francesco Costantino. Natural language processing applications in manufacturing: a systematic literature review. 09 2022.

[6] Chester L Cofino, Ryan B Escorial, and Debbie Lou B Enquilino. A literature review on natural language processing (nlp) in aiding industry to progress. *International Journal of Engineering Trends and Technology. Seventh Sense Research*, 2024.

[7] Benyamin Ghojogh and Ali Ghodsi. Recurrent neural networks and long short-term memory networks: Tutorial and survey, 2023. Accessed: April 29, 2025.

[8] IBM Corporation. Global AI adoption index 2022. Technical report, IBM, 2022.

[9] Anu Jagannath, Jithin Jagannath, and Prem Sagar Pattanshetty Vasanth Kumar. A comprehensive survey on radio frequency (rf) fingerprinting: Traditional approaches, deep learning, and open challenges. *Computer Networks*, 220:109455, 2022.

[10] Daniel Martin Katz, Dirk Hartung, Lauritz Gerlach, Abhik Jana, and Michael J Bommarito II. Natural language processing in the legal domain. *arXiv preprint arXiv:2302.12039*, 2023.

[11] Hemanth Sankar Kesara. Medium articles dataset. `https://www.kaggle.com/datasets/hsankesara/medium-articles`, 2019. Accessed: [17-05-2025].

[12] Hang Li. Language models: past, present, and future. *Communications of the ACM*, 65(7):56–63, 2022.

[13] Pascal Muam Mah, Iwona Skalna, and John Muzam. Natural language processing and artificial intelligence for enterprise management in the era of industry 4.0. *Applied Sciences*, 12(18), 2022.

[14] Ibomoiye Domor Mienye, Theo G. Swart, and George Obaido. Recurrent neural networks: A comprehensive review of architectures, variants, and applications. *Information*, 15(9):517, 2024.

[15] Elyas Rashno, Amir Eskandari, Aman Anand, and Farhana Zulkernine. Survey: Transformer-based models in data modality conversion. *arXiv preprint arXiv:2408.04723*, 2024.

[16] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[17] Johannes Schneider. What comes after transformers?–a selective survey connecting ideas in deep learning. *arXiv preprint arXiv:2408.00386*, 2024.

[18] ScienceDirect. Machine learning – an overview, n.d. Accessed: April 29, 2025.

[19] Flatworld Solutions. The transformational impact of natural language processing on business data insights, n.d. Accessed: April 29, 2025.

[20] S&P Global. Language modeling: The fundamentals, 2024. Accessed: April 29, 2025.

[21] TseKiChun. Neural network diagram. `https://www.sciencelearn.org.nz/images/5156-neural-network-diagram`, 2023. Image licensed under CC BY-SA 4.0.

[22] TutorAspire. Types of machine learning, n.d. Accessed: April 29, 2025.

[23] Liyang Wang, Yu Cheng, Ao Xiang, Jingyu Zhang, and Haowei Yang. Application of natural language processing in financial risk detection. *arXiv preprint arXiv:2406.09765*, 2024.

[24] C Wei, YC Wang, B Wang, and CCJ Kuo. An overview on language models: Recent developments and outlook. arxiv 2023. *arXiv preprint arXiv:2303.05759*.