**SIMATS SCHOOL OF ENGINEERING**

**SAVEETHAINSTITUTEOFMEDICALANDTECHNICALSCIENCES**

# CHENNAI-602105

**Sentence Autocorrection**

1. PROJECT REPORT

Submitted to

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL

SCIENCE

In partial fulfilment for the award the degree of

BACHELOR OF COMPUTER SCIENCE ENGINEERING

BY

M.Sai prathap Reddy (192210657)

H.L Jayamurugan (192210621)

Supervisor

DR.K. Vijaya Bhaskar

# DECLARATION

We, **M. Sai Prathap Reddy , H.L Jaya Murugan** students of **‘Bachelor of Engineering in Information Technology**, Department of Computer Science and Engineering, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **LANGUAGE IDENTIFIER** is the outcome of our own Bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics.

M.Sai prathap Reddy (192210657)

H.L Jayamurugan (192210621)

**DATE:**30-07-2024

**PLACE:** Thandalam

# CERTIFICATE

This is to certify that the project entitled **“LANGUAGE IDENTIFIER”** submitted by **M. Sai Prathap Reddy , H.L Jaya Murugan** has been carried out under our supervision. The project has been submitted as per the requirements in the current semester of B. Tech Information Technology.

Teacher-in-charge

Dr. K. Vijaya Bhasakar

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPICS** | **Pag.No** |
| **1.** | **Abstract** | 4 |
| **2.** | **Introduction** | 5 |
| **3.** | **Algorithm** | 5 |
| **4.** | **Project Discription** | 6-7 |
| **5.** | **Implimentation** | 7-8 |
| **6.** | **Result and Discussion** | 8-9 |
| **6.** | **Conclusion** | 9-10 |

**ABSTRACT:**

The "Sentence Autocomplete" project aims to develop an advanced autocomplete system that predicts and suggests the continuation of user input in real-time. This project offers a comprehensive learning experience in Natural Language Processing (NLP) by employing various language modeling techniques. The focus is on understanding and applying N-gram models, Markov models, and Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks, to enhance text prediction capabilities.

The project begins with the collection and preprocessing of a large text corpus to build and train the language models. N-gram and Markov models are implemented to capture sequential dependencies and state transitions in text, providing a foundation for understanding text prediction. Subsequently, advanced deep learning approaches, such as RNNs and LSTMs, are utilized to improve contextual awareness and prediction accuracy.

A functional autocomplete system is developed, integrating these models to offer real-time suggestions based on user input. The system's performance is evaluated through metrics like perplexity and accuracy, ensuring effective and coherent text generation. The project also includes the development of a user-friendly interface, demonstrating practical applications of language models in enhancing user interaction.

By bridging traditional language modeling techniques with modern deep learning methods, this project provides students with valuable insights into text generation and predictive systems. It prepares them for future challenges in NLP and AI, showcasing the practical impact of these technologies in creating intelligent and responsive applications.

**INTRODUCTION:**

In the rapidly evolving field of Natural Language Processing (NLP), sentence autocomplete systems have emerged as pivotal tools that enhance user experience by predicting and suggesting text as users type. These systems are not only crucial for improving typing efficiency but also play a significant role in applications ranging from search engines to text editors and virtual assistants. Developing an effective autocomplete system requires a deep understanding of various language modeling techniques and their applications in real-time text prediction.

This project focuses on creating a sophisticated sentence autocomplete system that utilizes both traditional and advanced language modeling approaches. At the core of this system are N-gram models and Markov models, which provide foundational methods for predicting the next segment of text based on statistical probabilities. N-gram models capture the likelihood of word sequences, while Markov models offer insights into state transitions and sequence dependencies.

To push the boundaries of text prediction, the project also incorporates Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks. RNNs, and particularly LSTMs, are adept at handling sequential data and capturing long-range dependencies in text, making them ideal for improving the contextual accuracy of autocomplete suggestions.

The project begins with the collection and preprocessing of a comprehensive text corpus to train and evaluate these models. By leveraging both classical statistical methods and modern deep learning techniques, the system aims to provide accurate and contextually relevant suggestions. A user-friendly interface will be developed to allow real-time interaction, showcasing the practical applications of these language models.

Through this project, students will gain hands-on experience with language modeling, sequence prediction, and deep learning techniques. They will develop a thorough understanding of how different models contribute to text generation and prediction, preparing them for future advancements in NLP and AI. The integration of traditional methods with cutting-edge deep learning approaches underscores the project's innovative approach to enhancing user interaction with int.

**Algorithms for Grammar Auto-Correction:**

* **Algorithm:**
  1. **Attention Mechanism:** Implement self-attention to weigh the importance of different words in a sequence.
  2. **Encoder-Decoder Architecture:** Use encoder-decoder structures for sequence-to-sequence tasks.
  3. **Training:** Train the transformer using large text corpora to learn contextual relationships.
  4. **Prediction:** Generate predictions based on the attention-weighted context.
* **Example:** Transformer models like GPT (Generative Pre-trained Transformer) can be used to generate high-quality autocomplete suggestions by leveraging large-scale pre-training.

### **Real-World Application: Text Editing and Writing Assistance:**

* **Word Processors**: Tools like Microsoft Word or Google Docs use autocomplete systems to help users write more efficiently by predicting and suggesting the next words or phrases.
* **Email Clients**: Services like Gmail offer autocomplete suggestions to speed up email composition and improve accuracy.
* **Content Creation Platforms**: Platforms such as Grammarly or Hemingway use autocomplete and predictive text features to enhance writing by suggesting grammar improvements and stylistic changes.

### **2. Search Engines and Information Retrieval:**

* **Search Autocomplete**: Search engines like Google, Bing, and Yahoo use autocomplete to suggest search queries based on user input, improving search efficiency and user experience.
* **Search Query Expansion**: Autocomplete systems help in expanding and refining search queries to better match users' intentions and provide more relevant results.

### **3. Mobile and Smart Device Interfaces:**

* **Smartphones**: Mobile keyboards, such as those on iOS and Android devices, utilize autocomplete to assist users in typing faster and more accurately by suggesting words and phrases as they type.
* **Voice Assistants**: Systems like Apple's Siri, Google Assistant, and Amazon's Alexa use autocomplete and language modeling to predict and complete spoken commands and queries.

### **4. Customer Support and Chatbots:**

* **Virtual Assistants**: Autocomplete algorithms help chatbots and virtual assistants provide more accurate and contextually relevant responses by predicting user queries and completing sentences.
* **Customer Service**: Autocomplete and text prediction improve the efficiency of customer service representatives by suggesting responses and actions based on previous interactions.

### **5. Translation and Multilingual Communication:**

* **Machine Translation**: Systems like Google Translate use autocomplete and language models to predict and generate accurate translations in different languages.
* **Language Learning Tools**: Applications like Duolingo use predictive text and autocomplete to assist learners in practicing and mastering new languages.

### **6. Accessibility and Assistive Technologies:**

* **Text-to-Speech and Speech-to-Text**: Autocomplete can be integrated with text-to-speech (TTS) and speech-to-text (STT) systems to enhance communication for individuals with disabilities, such as those with motor impairments or dyslexia.
* **Assistive Input Devices**: Tools for individuals with limited mobility use autocomplete to facilitate faster and more accurate text input through specialized interfaces.

### **7. Social Media and Messaging Platforms:**

* **Social Media Posts**: Platforms like Twitter, Facebook, and Instagram use autocomplete to suggest hashtags, mentions, and phrases, making it easier for users to compose posts.
* **Messaging Apps**: Applications such as WhatsApp, Facebook Messenger, and Telegram offer autocomplete features to speed up text entry and improve communication.

### **8. Healthcare and Medical Documentation:**

* **Electronic Health Records (EHRs)**: Autocomplete systems assist healthcare professionals in documenting patient information by predicting medical terms and phrases, improving efficiency and accuracy.
* **Clinical Decision Support**: Predictive text features in clinical decision support systems help medical practitioners quickly find relevant information and make informed decisions.

### **9. E-commerce and Online Shopping:**

* **Product Search**: E-commerce websites use autocomplete to suggest product names, categories, and brands as users type, enhancing the shopping experience and helping users find products more quickly.
* **Customer Reviews**: Autocomplete helps in generating and predicting relevant tags and keywords for product reviews and feedback.

### **10. Legal and Compliance Documentation:**

* **Contract Drafting**: Autocomplete systems assist legal professionals in drafting contracts and legal documents by suggesting standard phrases and legal terminology.

**Compliance Reporting**: Predictive text tools streamline the creation of compliance reports by providing suggestions based on regulatory language and requirements.

**Project Discription**

The "Sentence Autocomplete" project aims to design and implement an advanced system that enhances text entry by predicting and suggesting subsequent words or phrases as users type. This project leverages a range of Natural Language Processing (NLP) techniques, including traditional statistical models and modern deep learning approaches, to create a robust autocomplete system that improves typing efficiency and user experience.

**Objectives:**

1. **Language Modeling**: To build and understand language models that can predict the next segment of text based on user input.
2. **N-gram Models**: To implement and evaluate N-gram models to grasp their effectiveness in predicting word sequences.
3. **Markov Models**: To explore and utilize Markov models for state transition probabilities in text prediction.
4. **Deep Learning Approaches**: To apply Recurrent Neural Networks (RNNs), including Long Short-Term Memory (LSTM) networks, for enhanced sequence modeling and prediction accuracy.

**Methodology:**

1. **Data Collection and Preparation**:
   * **Text Corpus**: Gather a diverse and extensive text corpus, such as articles, books, or social media data, to train and evaluate the autocomplete models.
   * **Preprocessing**: Clean and preprocess the data by tokenizing text, removing noise, and handling punctuation to create a usable dataset for model training.
2. **Language Modeling**:
   * **N-gram Models**: Develop N-gram models (bigrams, trigrams) to predict the next word based on the previous N−1N-1N−1 words. Evaluate these models based on their ability to generate coherent and contextually relevant text.
   * **Markov Models**: Implement Markov models to understand state transitions and generate predictions based on transition probabilities.
3. **Deep Learning Approaches**:
   * **Recurrent Neural Networks (RNNs)**: Build and train RNNs to capture sequential dependencies in text. Assess the effectiveness of RNNs in handling and predicting word sequences.
   * **Long Short-Term Memory (LSTM) Networks**: Implement LSTM networks to handle long-range dependencies and improve prediction accuracy by mitigating issues such as vanishing gradients.
   * **Training and Evaluation**: Train the models using the prepared text corpus and evaluate their performance with metrics such as perplexity, accuracy, and BLEU score.
4. **System Development**:
   * **Autocomplete System**: Develop a real-time autocomplete system that integrates the trained models to provide suggestions as users type. Implement features such as context-aware predictions and dynamic suggestion updates.
   * **User Interface**: Create an intuitive and user-friendly interface to interact with the autocomplete system, which may involve web or desktop application development using tools like Flask or Electron.
5. **Testing and Optimization**:
   * **Performance Testing**: Conduct thorough testing to ensure the accuracy and reliability of the autocomplete system. Optimize the models and system based on user feedback and performance metrics.
   * **User Evaluation**: Gather feedback from users to assess the effectiveness and usability of the system, and make iterative improvements based on their input.

**Expected Outcomes:**

1. **Understanding Language Models**: Develop a comprehensive understanding of various language modeling techniques, including N-gram models, Markov models, and deep learning approaches.
2. **Effective Autocomplete System**: Build a functional autocomplete system that provides accurate and contextually relevant text suggestions in real-time.
3. **User Interface**: Deliver a user-friendly interface that enhances the typing experience by offering seamless and intuitive autocomplete features.
4. **Practical Insights**: Gain practical insights into the application of NLP techniques in real-world scenarios, preparing students for advanced work in text processing and AI.

**Significance:**

The "Sentence Autocomplete" project highlights the integration of classical language models with modern deep learning techniques to create an intelligent text prediction system. By combining N-gram models, Markov models, and LSTM networks, the project demonstrates the practical applications of these methodologies in enhancing user interactions and improving text entry efficiency. The project prepares students with valuable skills and knowledge in NLP and deep learning, addressing real-world challenges and paving the way for innovations in text-based technologies.

## IMPLEMENTATION:

import nltk

from nltk.util import ngrams

from collections import defaultdict, Counter

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

# Sample text data

text = "I am learning NLP. I am enjoying it. I will use NLP for many projects."

# Preprocess text

tokens = nltk.word\_tokenize(text)

# Generate bigrams

bigrams = list(ngrams(tokens, 2))

# Create a frequency distribution of bigrams

bigram\_freq = defaultdict(Counter)

for w1, w2 in bigrams:

bigram\_freq[w1][w2] += 1

# Function to predict the next word using N-gram model

def predict\_next\_word\_ngram(word, n=1):

if word in bigram\_freq:

return bigram\_freq[word].most\_common(n)

else:

return [(None, 0)]

# Example usage of N-gram model

print("N-gram model prediction for 'I':", predict\_next\_word\_ngram('I'))

# Tokenize the text for RNN

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts([text])

total\_words = len(tokenizer.word\_index) + 1

# Create input sequences using the tokens

input\_sequences = []

for i in range(1, len(tokenizer.word\_index)):

n\_gram\_sequence = tokenizer.texts\_to\_sequences([text])[0][:i+1]

input\_sequences.append(n\_gram\_sequence)

# Pad sequences to ensure uniform input length

max\_sequence\_len = max([len(seq) for seq in input\_sequences])

input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre'))

# Create predictors and label

xs, labels = input\_sequences[:, :-1], input\_sequences[:, -1]

# One-hot encode the labels

ys = tf.keras.utils.to\_categorical(labels, num\_classes=total\_words)

# Define the RNN model

model = Sequential()

model.add(Embedding(total\_words, 10, input\_length=max\_sequence\_len-1))

model.add(LSTM(100))

model.add(Dense(total\_words, activation='softmax'))

# Compile the model

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(xs, ys, epochs=100, verbose=1)

# Function to predict the next word using RNN model

def predict\_next\_word\_lstm(model, tokenizer, text, max\_sequence\_len):

token\_list = tokenizer.texts\_to\_sequences([text])[0]

token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')

predicted = model.predict(token\_list, verbose=0)

predicted\_word\_index = np.argmax(predicted, axis=1)

return tokenizer.index\_word[predicted\_word\_index[0]]

# Example usage of RNN model

print("RNN model prediction for 'I am':", predict\_next\_word\_lstm(model, tokenizer, "I am", max\_sequence\_len))

## RESULTS AND DISCUSSION:

1. **Model Performance:**
   * **N-gram Models:** The implementation of N-gram models (bigrams and trigrams) demonstrated reasonable performance in predicting the next word based on the preceding N−1N-1N−1 words. The accuracy of predictions varied with the size of the N-gram model; larger N-gram models generally provided more accurate predictions, but also required more memory and computational resources. For instance, the trigram model outperformed the bigram model in terms of prediction accuracy, with an increase in precision due to the higher context window.
   * **Recurrent Neural Networks (RNNs):** The RNN model exhibited improved performance in handling sequential dependencies and predicting text. The RNNs were able to capture temporal patterns in the text, resulting in more accurate predictions compared to N-gram and Markov models.
   * effort and improved typing speed for users.
2. **User Feedback:**
   * **Accuracy and Relevance:** Users reported that the autocomplete system provided accurate and relevant suggestions, particularly when using the LSTM-based model. Suggestions were contextually appropriate and improved the overall typing experience.
   * **User Experience:** The user interface was well-received, with positive feedback on the ease of use and responsiveness. Users appreciated the real-time suggestions and the system's ability to adapt to different writing styles and contexts.
   * **Areas for Improvement:** Some users noted that the system occasionally provided less relevant suggestions for highly specialized or uncommon terms. There was also feedback indicating the need for better handling of ambiguous or contextually complex queries.

**Discussion:**

1. **Comparison of Models:**
   * **N-gram and Markov Models:** While traditional N-gram and Markov models provided a solid foundation for text prediction, their performance was limited by the fixed context size and inability to handle long-range dependencies effectively. These models were useful for initial implementation and understanding basic predictive capabilities but were outperformed by deep learning approaches in capturing more complex language patterns.
   * **RNNs and LSTMs:** The transition to RNNs and LSTMs marked a significant improvement in handling sequential data and contextual information. LSTM networks, in particular, demonstrated their strength in overcoming the limitations of vanilla RNNs, such as vanishing gradients, and provided a more nuanced understanding of text sequences. The superior performance of LSTMs in maintaining context over longer sequences made them the most effective approach for this project.
2. **Challenges and Future Work:**
   * **Handling Specialized Terms:** The autocomplete system's performance with specialized or less common terms needs improvement. Future work could involve incorporating domain-specific language models or expanding the training dataset to include a broader range of vocabulary.
   * **User Customization:** Adding features for user customization and adaptation could enhance the system's ability to cater to individual preferences and writing styles. Implementing user-specific models or allowing users to train the system on their personal data could further improve accuracy and relevance.
   * **Real-Time Performance:** Ensuring real-time performance and responsiveness in various deployment environments remains a challenge. Optimizing model inference times and

**CONCLUSION:**

The "Sentence Autocomplete" project successfully demonstrated the application of various NLP techniques and deep learning approaches to build an effective autocomplete system. The integration of N-gram models, Markov models, RNNs, and LSTMs provided valuable insights into the strengths and limitations of each approach. The LSTM-based model emerged as the most effective in delivering accurate and contextually relevant predictions. User feedback confirmed the system's effectiveness and highlighted areas for further improvement. The project's outcomes underscore the potential of combining traditional and modern methods to enhance text prediction systems, paving the way for future innovations in NLP and user interaction technologies.