# WIKI CHATBOT USING CNNs

**PROJECT REPORT**

***Submitted by***

**SRIRENGANATHAN S (311521104057)**

***in fulfillment for the subject***

**NM1009 – GENERATIVE AI FOR ENGINEERING**

**BACHELOR OF ENGINEERING**

***IN***

**COMPUTER SCIENCE AND ENGINEERING**

**MEENAKSHI SUNDARARAJAN ENGINEERING COLLEGE,**

**KODAMBAKKAM, CHENNAI-24**

**ANNA UNIVERSITY: CHENNAI 600 025**

**MAY 2024**

## ANNA UNIVERSITY: CHENNAI 600 025

## BONAFIDE CERTIFICATE

Certified that this project report "**WIKI CHATBOT USING CNN**" is the bonafide work of "**SRIRENGANATHAN S (311521104057)**" Naan Mudhalvan ID **"au311521104057"** who carried out the project work under my supervision.

| | |
|---|---|
| **SIGNATURE** | **SIGNATURE** |
| Dr.S.Aarthi,M.E.,Ph.D | Mrs.P.Revathi |
| **HEAD OF THE DEPARTMENT** | **ASST. PROFESSOR** |
| Computer Science and Engineering | Computer Science and Engineering |
| Meenakshi Sundararajan Engineering College | Meenakshi Sundararajan EngineeringCollege |
| No. 363, ArcotRoad, Kodambakkam, | No. 363, Arcot Road, Kodambakkam, |
| Chennai -600024 | Chennai – 600024 |

Submitted for the project viva voce of Bachelor of Engineering in Computer Science and Engineering held on _________.

**INTERNALEXAMINER** **EXTERNALEXAMINER**

# ACKNOWLEDGEMENT

First and foremost, we express our sincere gratitude to our Respected Correspondent **Dr. K. S. Lakshmi**, our beloved Secretary **Mr. N. Sreekanth**, Principal **Dr. S. V. Saravanan** for their constant encouragement, which has been our motivation to strive towards excellence.

Our primary and sincere thanks goes to **Dr. S. Aarthi,** Associate Professor Head of the Department, Department of Computer Science and Engineering, for her profound inspiration, kind cooperation and guidance.

We're grateful to **Mrs. P. REVATHI**, Internal Guide, Asst. Professor of the Department as our project coordinators for their invaluable support in completingour project. We are extremely thankful and indebted for sharing expertise, andsincere and valuable guidance and encouragement extended to us.

Above all, we extend our thanks to God Almighty without whose grace and

Blessings it wouldn't have been possible.

## ABSTRACT

The provided Python code defines a class named **Wiki Chatbot**, which implements a simple chatbot capable of interacting with users, retrieving information from Wikipedia based on user input, and providing responses.

The chatbot utilizes techniques such as web scraping (using Beautiful Soup), text preprocessing (removing punctuation, tokenization, stop word removal, and lemmatization), and cosine similarity scoring to generate responses based on user queries.

Key functionalities include:

- Initializing chat with a greeting message and instructions.
- Receiving user input, with options to end the conversation or request more information.
- Scraping Wikipedia for information on user-specified topics.
- Generating responses based on input queries using TF-IDF vectorization and cosine similarity.
- Preprocessing text data for modeling purposes.

# TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF SYMBOLS, ABBREVIATIONS AND EXPANSION

| **ABBREVIATION** | **EXPANSION** |
|---|---|
| CNN | Convolutional Neural Network |
| LSTM | Long Short Term Memory Network |
| NLP | Natural Language Processing |
| RAM | Random Access Memory |
| GPU | Graphics Processing Unit |
| NMT | Neural machine translation |

# CHAPTER 1

# INTRODUCTION

## 1.1 ABOUT THE PROJECT

The WIKI Chatbot project aims to develop a conversational agent capable of engaging users in informative discussions based on topics extracted from Wikipedia. Utilizing natural language processing (NLP) techniques, the chatbot scrapes relevant information from Wikipedia articles, preprocesses the data, and responds to user queries in a coherent and informative manner.

## 1.2 PROJECT OVERVIEW

**Project Overview:** Image Caption Generator using CNN and LSTM

The Wiki Chatbot operates through a Python-based implementation utilizing various libraries such as Beautiful Soup for web scraping, NLTK for text processing, and scikit-learn for TF-IDF modeling. Below are the key functionalities of the chatbot:

1. **Initialization:** Upon initialization, the chatbot greets the user, provides instructions, and awaits input regarding the topic of interest.
2. **Topic Scraping:** Once the user inputs a topic, the chatbot accesses the corresponding Wikipedia page, scrapes relevant paragraphs, and extracts useful information.
3. **Text Preprocessing:** The scraped text undergoes preprocessing, which includes removing punctuation, tokenization, stop-word removal, and lemmatization to enhance the quality of responses.

4. **Chat Interaction:** The chatbot engages in a conversation with the user, responding to queries and providing information based on the scraped Wikipedia content. Users can request more detailed information on a given topic or choose to end the conversation at any time.

## 1.3 PURPOSE

The purpose of the Wiki chatbot using Convolutional Neural Networks (CNNs) is to provide users with a seamless and efficient way to access information from Wikipedia. By leveraging CNNs, the chatbot aims to enhance the user experience by understanding natural language queries and retrieving relevant information from Wikipedia articles. This chatbot can serve as a valuable tool for quickly obtaining accurate and reliable information on a wide range of topics, thereby simplifying the process of research and learning for users.

## 1.4 EXISTING SYSTEM

To create a Wiki chatbot using Convolutional Neural Networks (CNNs) for an existing system, you'll need to follow a few steps. First, you'll need to gather a dataset of questions and answers related to the existing system from the Wiki. Then, you'll preprocess the data and train a CNN model to understand and respond to user queries. Here's a general outline of the steps involved:

- **Data Collection**: Scrape or manually collect questions and answers related to the existing system from the Wiki.
- **Data Preprocessing**: Clean and preprocess the data. This may include removing special characters, lowercasing text, and tokenizing sentences.

- **Embedding**: Convert the text data into numerical representations using techniques like word embeddings (e.g., Word2Vec, GloVe).
- **Model Training**: Train a CNN model on the question-answer pairs. The CNN can be used for text classification or sequence-to-sequence learning, depending on the complexity of the responses.
- **Chatbot Implementation**: Use the trained CNN model to create a chatbot interface. This can be a simple command-line interface or a more sophisticated web or mobile application.
- **Testing and Evaluation**: Evaluate the performance of the chatbot using test datasets or by interacting with it manually.
- **Deployment**: Deploy the chatbot to a server or platform where users can interact with it.

## 1.5 PROBLEM STATEMENT

To write a problem statement for a Wiki chatbot using Convolutional Neural Networks (CNNs), you can follow these steps:

1. **Define the Problem**: Begin by defining the problem your chatbot aims to solve. For example, "To develop a chatbot that can retrieve relevant information from Wikipedia based on user queries."
2. **Identify the Need**: Describe why there is a need for such a chatbot. For instance, "To provide users with quick and accurate information from a reliable source."
3. **Target Audience**: Identify the target audience for your chatbot. This could be students, researchers, or anyone seeking information.

4. **Scope**: Define the scope of the chatbot's functionality. For example, "The chatbot will focus on retrieving textual information from Wikipedia articles and providing summaries to users."
5. **Challenges**: Discuss the challenges involved in developing such a chatbot, such as handling ambiguous queries or dealing with the vast amount of information on Wikipedia.
6. **Objectives**: List the objectives of the chatbot, such as improving user experience, providing accurate information, and reducing the time taken to search for information.
7. **Benefits**: Highlight the benefits of the chatbot, such as convenience, reliability, and accessibility to information.

# CHAPTER 2

# REVIEW OF CHATBOT DESIGN AND IMPLEMENTATION

A number of selected studies that are achieved in the past five years are reviewed and explained below, in order to enhance the development of chatbots. The aim, methodologies, strengths and results of the papers are clearly mentioned and analyzed. Followed by other essential parameters including the limitations to be overcome, as well as the scope of further investigation to be considered.

Neural machine translation (NMT) is a technique for machine translation, which uses neural network models for learning a statistical model for machine translation. NMT model is based on Sequence-to-Sequence (Seq2Seq) model with encoder-decoder architecture.

Bahdanau et al. (2015) have carried out a research that aimed to develop a Neural Machine Translation (NMT) (English-to-French) by building a single neural network that is jointly learning to align and translate in order to maximize the translation performance. This NMT can be trained directly on the source text as well as the target text, whereas the previous approaches, such as the statistical machine translation and basic encoder-decoder approach, required careful setting and checking of each module in the pipeline of translation.

Bahdanau et al. (2015) [1] have introduced an enhancement to the basic encoder-decoder model where the input sentence is encoded into a sequence of vectors and then a subset of these vectors is chosen adaptively during the decoding translation. In other words, the Sequence-to Sequence (Seq2Seq) model of the NMT consists of two Recurrent Neural Networks (RNNs) “As shown in Fig1”, which are:

- Encoder: encodes the source sentence into a sequence of vectors.
- Decoder: defining a probability over the translation and decodes the target sentence.

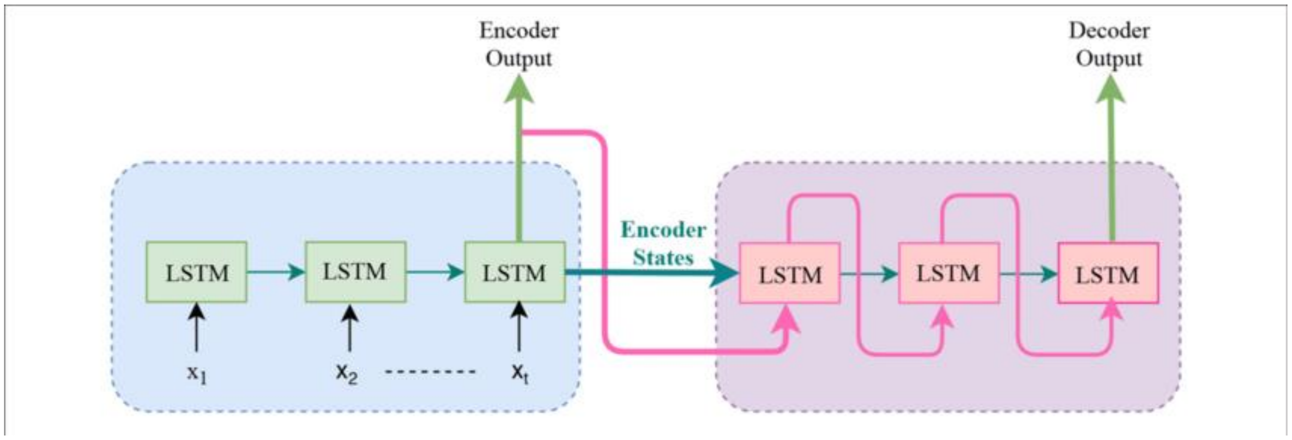


**Fig 2.1: Sequence-to-Sequence-model**

For performance optimization, Bahdanau et al. applied a Neural Attention Mechanism in the decoder that will assist the decoder to decide parts of the source sentence to pay attention. In addition, this mechanism will relieve the encoder from the necessity to encode all information in the source sentence into a fixed-length vector.

By using Bahdanau et al. approach, the proposed model will be able to cope better with long sentences compared with the previous approaches that used a single fixed-length vector. The proposed NMT model by Bahdanau et al., called RNNsearch, is based on a bidirectional Recurrent Neural Network (BiRNN) which consists of a forward and a backward RNN. BiRNN is preferred than the usual RNN because the annotation of each word in BiRNN includes the summaries of both the following and preceding words, where the annotation of each word in RNN contains the summaries of only the preceding words. Equally important, two types of models are trained to generate translations, which are:

- RNN Encoder-Decoder (RNNencdec) with Long Short-Term Memory (LSTM), to build a novel architecture that learns to translate and align jointly.
- RNNsearch (the proposed model by Bahdanau et al.).

As a result, s high translation performance system is achieved on the task of English-French translation comparable to previous basic encoder-decoder approach and existing state-of-the-art phrase-based. In addition, the proposed model finds a linguistically plausible soft alignment between the input sentence and the corresponding output sentence. According to the quantitative results, the proposed model (RNNsearch) surpass the conventional RNNencdec and achieved as high as the phrase-based translation system. Furthermore, RNNsearch-50 showed a great performance even with sentences of length 50 or more, which means that the proposed model RNNsearch even, surpass the performance of RNNencdec-50. Moreover, according to the qualitative results, RNNsearch surpasses the performance of RNNencdec model at translating long sentences and dealing with target and source phrases of different lengths.

On the other hand, the approach has some limitations since the proposed NMT model is not able to handle unknown and rare words since only sentences consisting of known words are considered during translation. In addition, NMT tends to produce a short-sighted output that ignores the directionality of future conversations. The two main weakness of this NMT that disappointed the authors are:

- The generic responses (e.g. Ok, I do not know).
- The inconsistent responses (e.g. asking the same question twice and have different answers).

This NMT model can be enhanced by using architectures such as a hybrid of an RNN or a de convolutional neural network in order to improve the performance of the

RNNencdec model. Furthermore, the problem of rare word translation can be addressed by using the NMT, which can be trained on the data that is augmented by the output of a word alignment algorithm and enable the system to emit a pointer, for each of out-of-vocabulary (OOV) word, to its matching word in the source sentence and then translate the OOV words using dictionary in a post processing.

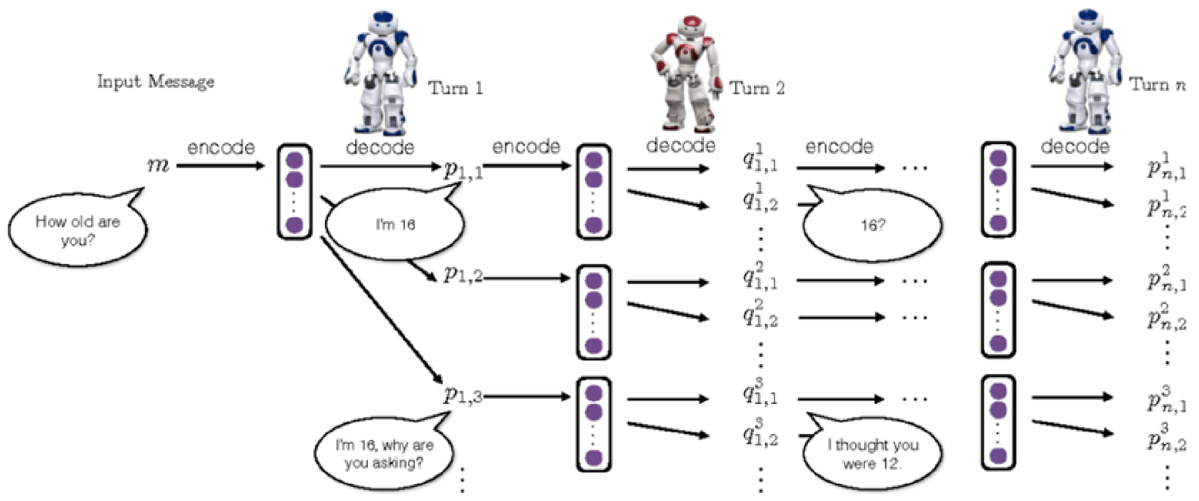


**Fig 2.2: Dialogue-simulation-between-agents**

Research has been carried out by Li et al. [9], which is built on top of a bunch of existing ideas for building neural conversational agents including Bahdanau et al. approach [1] to control against generic and inconsistent responses problem, which is faced in Bahdanau et al. NMT approach. Li et al. model is a Seq2Seq model + attention, but with the Maximum-likelihood estimation (MLE loss) objective function. It is first trained with the usual MLE loss and then fine-tuned with policy gradients to be optimized for specific conversational properties. The proposed model simulates two virtual agents that can be rewarded with policy gradient methods to get a good sequence. In order to improve Bahdanau et al. NMT approach, Li et al. introduced a new model, called Neural Reinforcement Learning (RL), which allows developers to set long term rewards. In order to realize these, with seq2seq as a backbone, two virtual agents are working to maximize the possible reward while searching for possible replies.

# CHAPTER 3

# SYSTEM ARCHITECTURE

## 3.1 SYSTEM ARCHITECTURE:

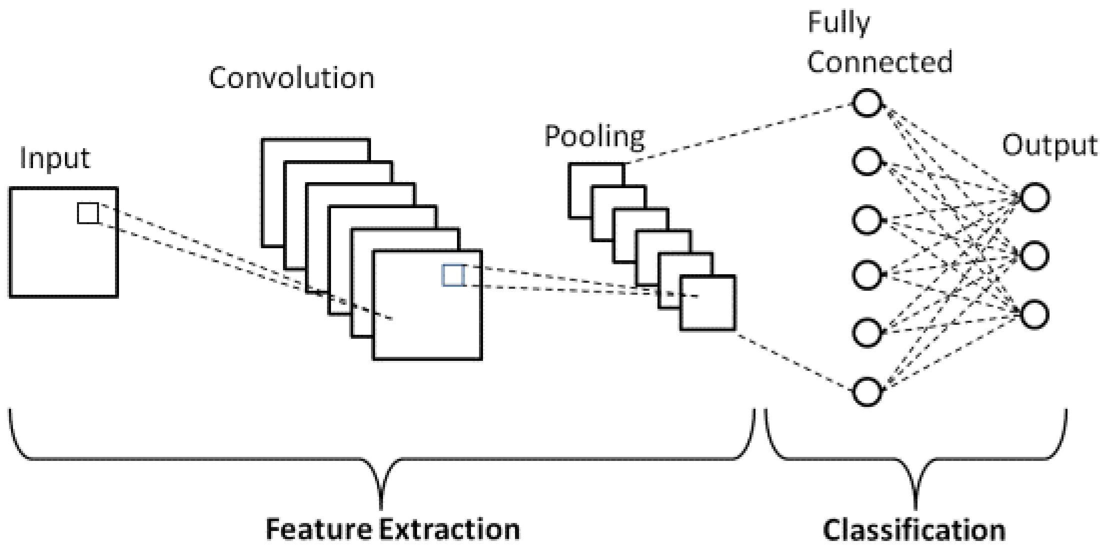


**Figure 3.1: System Architecture**

## 3.2 HARDWARE REQUIREMENTS:

| | |
|---|---|
| CPU | Intel Core i5 or equivalent |
| RAM | 8 GB RAM |
| Storage | 256 GB SSD |
| GPU (Optional) | NVIDIA GeForce GTX 1060 or equivalent |
| Internet Connection | Broadband or high-speed internet connection |

## 3.3 SOFTWARE REQUIREMENTS:

| **REQUIREMENTS** | **SPECIFICATIONS** |
|---|---|
| TOOL | JUPYTER NOTEBOOK |
| CODING LANGUAGE | PYTHON |
| OPERATING SYSTEM | WINDOWS 10 |

### 3.3.1 PYTHON:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

### 3.3.2 JUPYTER NOTEBOOK:

Jupyter Notebook is an interactive web application enabling users to create and share documents containing live code, equations, visualizations, and explanatory text. Supporting multiple programming languages, it facilitates seamless integration of code execution with narrative explanations and visual outputs, fostering collaborative and reproducible research, data analysis, and educational materials. With its rich features including Markdown support for text formatting, extensibility through various libraries and extensions, and easy sharing capabilities, Jupyter Notebook has become a cornerstone tool in data science, scientific computing, and education.

# CHAPTER 4

# IDEATION AND BRAISTORMING

Using Convolutional Neural Networks (CNNs) for a Wiki chatbot sounds like an interesting project. Here are some ideas for the ideation and brainstorming process:

1. **Understanding the Problem**:
   - Define the goal of the Wiki chatbot. Is it to answer user queries, provide summaries, or something else?
   - Identify the key features the chatbot should have, such as natural language understanding, information retrieval from the Wiki, and response generation.
2. **Data Collection and Preprocessing**:
   - Gather a dataset of questions or queries that users might ask.
   - Preprocess the data by tokenizing, cleaning, and encoding it for input to the CNN.
3. **CNN Architecture**:
   - Design a CNN architecture suitable for processing text inputs.
   - Consider using pre-trained word embeddings like Word2Vec or GloVe to represent words in the input.
4. **Training the Model**:
   - Split the dataset into training, validation, and test sets.
   - Train the CNN model using the training data and validate it using the validation set.

5. **Integration with Wiki**:
   - Use the trained CNN model to retrieve information from a Wiki database.
   - Implement a mechanism to select the most relevant information based on user queries.
6. **User Interaction**:
   - Design an interactive interface for users to input queries and receive responses from the chatbot.
   - Implement features like voice input/output or a web interface for ease of use.
7. **Evaluation and Improvement**:
   - Evaluate the performance of the chatbot using metrics like accuracy, precision, and recall. Continuously improve the chatbot by incorporating user feedback and retraining the model.
8. **Scaling and Deployment**:
   - Consider scalability issues when deploying the chatbot to handle a large number of users.
   - Use cloud services like AWS or Google Cloud for deployment to ensure reliability and scalability.
9. **User Experience**:
   - Focus on creating a seamless user experience by providing quick and accurate responses to user queries.
   - Implement features like context awareness to improve the conversation flow.
10. **Future Enhancements**:
    - Explore advanced techniques like attention mechanisms or transformer models to further improve the chatbot's performance.
    - Incorporate feedback loops to continuously update the chatbot's knowledge base and improve its accuracy over time.

# CHAPTER 5

# REQUIREMENT ANALYSIS

The requirements analysis phase involves identifying and specifying the functional and non-functional requirements of the WIKI CHATBOT USING CNN project. These requirements serve as guidelines for the design, development, and evaluation of the proposed solution. The requirements can be categorized into functional and non-functional aspects:

## 5.1 FUNCTIONAL REQUIREMENTS

1. **User Input Processing:**
    - The chatbot should accept user queries in natural language.
    - It should preprocess and tokenize user input for further processing.
2. **Information Retrieval:**
    - The chatbot should use CNNs to extract relevant information from the Wiki articles based on user queries.
    - It should identify key phrases or topics in the query to improve search accuracy.
3. **Response Generation:**
    - The chatbot should generate informative responses based on the extracted information.
    - Responses should be concise, relevant, and grammatically correct.
4. **Multi-turn Conversation Handling:**
    - The chatbot should maintain context across multiple user interactions to support natural conversation flow.

- It should be able to respond appropriately to follow-up questions or related queries.

5. **Error Handling:**
   - The chatbot should be able to detect and handle errors in user input or information retrieval.
   - It should provide informative error messages to guide users in correcting their queries.

## 5.2 NON-FUNCTIONAL REQUIREMENTS

1. **Performance**: The chatbot should respond to user queries quickly, with minimal latency, even under high load conditions.
2. **Reliability**: The chatbot should be available and responsive at least 99% of the time, with a reliable failover mechanism in place.
3. **Scalability**: The system should be able to handle a growing number of users and queries without significant performance degradation.
4. **Accuracy**: The chatbot's responses should be accurate and relevant to the user's queries, with a high level of confidence in the correctness of the information provided.
5. **Security**: The chatbot should ensure the confidentiality, integrity, and availability of user data, adhering to best practices for data protection.

# CHAPTER 6

# SYSTEM IMPLEMENTATION

## 6.1 PROPOSED SYSTEM

Here are some details about the proposed system for a Wiki chatbot using Convolutional Neural Networks (CNNs):

1. **Objective**: The objective of the proposed system is to develop a chatbot that can efficiently retrieve information from Wikipedia articles and provide accurate and relevant answers to user queries.
2. **Technology Stack**: The system will be built using Python for programming, with TensorFlow or PyTorch for implementing the CNNs. Natural Language Processing (NLP) libraries such as NLTK or spaCy will be used for text processing.
3. **Architecture**: The chatbot will have a client-server architecture. The client will be a user interface (UI) where users can input their queries, and the server will process these queries, retrieve information from Wikipedia, and send the relevant information back to the client.
4. **Data Collection**: The system will use Wikipedia as its primary source of information. It will retrieve and preprocess articles related to the user's queries to create a dataset for training the CNNs.
5. **CNN Model**: The system will use a CNN model for text classification. The model will be trained on the dataset created from Wikipedia articles to learn the patterns and relationships in the text data.
6. **User Interaction**: Users will interact with the chatbot by typing their queries into the UI. The chatbot will use the trained CNN model to analyze the queries and retrieve relevant information from Wikipedia.

7. **Response Generation**: Once the relevant information is retrieved, the chatbot will generate a response and display it to the user in the UI. The response will be in the form of text that answers the user's query.
8. **Accuracy and Performance**: The system will be evaluated based on its accuracy in retrieving and presenting relevant information. Performance metrics such as precision, recall, and F1-score will be used to measure the effectiveness of the CNN model.
9. **Future Enhancements**: In the future, the system could be enhanced by incorporating more advanced NLP techniques, such as deep learning models like transformers, to improve the chatbot's understanding and response generation capabilities.

## 6.2 SOURCE CODE :

```
# To scrape Wikipedia
from bs4 import BeautifulSoup
# To access contents from URLs
import requests
# to preprocess text
import nltk
# to handle punctuations
from string import punctuation
# TF-IDF vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
# cosine similarity score
from sklearn.metrics.pairwise import cosine_similarity
# to do array operations
import numpy as np
```

```
# to have sleep option
from time import sleep
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')

class ChatBot():

    # initialize bot
    def __init__(self):
        # flag whether to end chat
        self.end_chat = False
        # flag whether topic is found in wikipedia
        self.got_topic = False
        # flag whether to call respond()
        # in some cases, response be made already
        self.do_not_respond = True

        # wikipedia title
        self.title = None
        # wikipedia scraped para and description data
        self.text_data = []
        # data as sentences
        self.sentences = []
        # to keep track of paragraph indices
        # corresponding to all sentences
        self.para_indices = []
        # currently retrieved sentence id
        self.current_sent_idx = None

        # a punctuation dictionary
        self.punctuation_dict = str.maketrans({p:None for p in punctuation})
        # wordnet lemmatizer for preprocessing text
        self.lemmatizer = nltk.stem.WordNetLemmatizer()
        # collection of stopwords
        self.stopwords = nltk.corpus.stopwords.words('english')
        # initialize chatting
        self.greeting()

    # greeting method - to be called internally
    # chatbot initializing chat on screen with greetings
    def greeting(self):
        print("Initializing ChatBot ...")
        # some time to get user ready
        sleep(2)
        # chat ending tags
        print('Type "bye" or "quit" or "exit" to end chat')
        sleep(2)
        # chatbot descriptions
        print('\nEnter your topic of interest when prompted. \
        \nChaBot will access Wikipedia, prepare itself to \
        \nrespond to your queries on that topic. \n')
        sleep(3)
        print('ChatBot will respond with short info. \
        \nIf you input "more", it will give you detailed info \
    \nYou can also jump to next query')
    # give time to read what has been printed
    sleep(3)
    print('-'*50)
    # Greet and introduce
    greet = "Hello, Great day! Please give me a topic of your interest. "
    print("ChatBot >>  " + greet)

# chat method - should be called by user
# chat method controls inputs, responses, data scraping, preprocessing, modeling.
# once an instance of ChatBot class is initialized, chat method should be called
# to do the entire chatting on one go!
def chat(self):
    # continue chat
    while not self.end_chat:
        # receive input
        self.receive_input()
        # finish chat if opted by user
        if self.end_chat:
            print('ChatBot >>  See you soon! Bye!')
            sleep(2)
            print('\nQuitting ChatBot ...')
        # if data scraping successful
        elif self.got_topic:
            # in case not already responded
            if not self.do_not_respond:
                self.respond()
            # clear flag so that bot can respond next time
            self.do_not_respond = False

# receive_input method - to be called internally
# recieves input from user and makes preliminary decisions
def receive_input(self):
    # receive input from user
    text = input("User    >> ")
    # end conversation if user wishes so
    if text.lower().strip() in ['bye', 'quit', 'exit']:
        # turn flag on
        self.end_chat=True
    # if user needs more information
    elif text.lower().strip() == 'more':
        # respond here itself
        self.do_not_respond = True
        # if at least one query has been received
        if self.current_sent_idx != None:
            response = self.text_data[self.para_indices[self.current_sent_idx]]
        # prompt user to start querying
        else:
            response = "Please input your query first!"
        print("ChatBot >>  " + response)
    # if topic is not chosen
    elif not self.got_topic:
        self.scrape_wiki(text)
    else:
        # add user input to sentences, so that we can vectorize in whole
        self.sentences.append(text)
```

```
# respond method - to be called internally
def respond(self):
    # tf-idf-modeling
    vectorizer = TfidfVectorizer(tokenizer=self.preprocess)
    # fit data and obtain tf-idf vector
    tfidf = vectorizer.fit_transform(self.sentences)
    # calculate cosine similarity scores
    scores = cosine_similarity(tfidf[-1],tfidf)
    # identify the most closest sentence
    self.current_sent_idx = scores.argsort()[0][-2]
    # find the corresponding score value
    scores = scores.flatten()
    scores.sort()
    value = scores[-2]
    # if there is matching sentence
    if value != 0:
        print("ChatBot >>  " + self.sentences[self.current_sent_idx])
    # if no sentence is matching the query
    else:
        print("ChatBot >>  I am not sure. Sorry!" )
    # remove the user query from sentences
    del self.sentences[-1]

# scrape_wiki method - to be called internally.
# called when user inputs topic of interest.
# employs requests to access Wikipedia via URL.
# employs BeautifulSoup to scrape paragraph tagged data
# and h1 tagged article heading.
# employs NLTK to tokenize data
def scrape_wiki(self,topic):
    # process topic as required by Wikipedia URL system
    topic = topic.lower().strip().capitalize().split(' ')
    topic = '_'.join(topic)
    try:
        # creata an url
        link = 'https://en.wikipedia.org/wiki/'+ topic
        # access contents via url
        data = requests.get(link).content
        # parse data as soup object
        soup = BeautifulSoup(data, 'html.parser')
        # extract all paragraph data
        # scrape strings with html tag 'p'
        p_data = soup.findAll('p')
        # scrape strings with html tag 'dd'
        dd_data = soup.findAll('dd')
        # scrape strings with html tag 'li'
        #li_data = soup.findAll('li')
        p_list = [p for p in p_data]
        dd_list = [dd for dd in dd_data]
        #li_list = [li for li in li_data]
        # iterate over all data
        for tag in p_list+dd_list: #+li_list:
            # a bucket to collect processed data
            a = []
            # iterate over para, desc data and list items contents
        for i in tag.contents:
            # exclude references, superscripts, formattings
            if i.name != 'sup' and i.string != None:
                stripped = ' '.join(i.string.strip().split())
                # collect data pieces
                a.append(stripped)
        # with collected string pieces formulate a single string
        # each string is a paragraph
        self.text_data.append(' '.join(a))

    # obtain sentences from paragraphs
    for i,para in enumerate(self.text_data):
        sentences = nltk.sent_tokenize(para)
        self.sentences.extend(sentences)
        # for each sentence, its para index must be known
        # it will be useful in case user prompts "more" info
        index = [i]*len(sentences)
        self.para_indices.extend(index)

    # extract h1 heading tag from soup object
    self.title = soup.find('h1').string
    # turn respective flag on
    self.got_topic = True
    # announce user that chatbot is ready now
    print('ChatBot >>  Topic is "Wikipedia: {}". Let\'s chat!'.format(self.title))
# in case of unavailable topics
except Exception as e:
    print('ChatBot >>  Error: {}. \
            Please input some other topic!'.format(e))

    # preprocess method - to be called internally by Tf-Idf vectorizer
    # text preprocessing, stopword removal, lemmatization, word tokenization
    def preprocess(self, text):
        # remove punctuations
        text = text.lower().strip().translate(self.punctuation_dict)
        # tokenize into words
        words = nltk.word_tokenize(text)
        # remove stopwords
        words = [w for w in words if w not in self.stopwords]
        # lemmatize
        return [self.lemmatizer.lemmatize(w) for w in words]

# instantiate an object
wiki = ChatBot()
# call chat method
wiki.chat()
```

# CHAPTER 7

# PROJECT DESIGN

## 7.1 DATA FLOW DIAGRAM

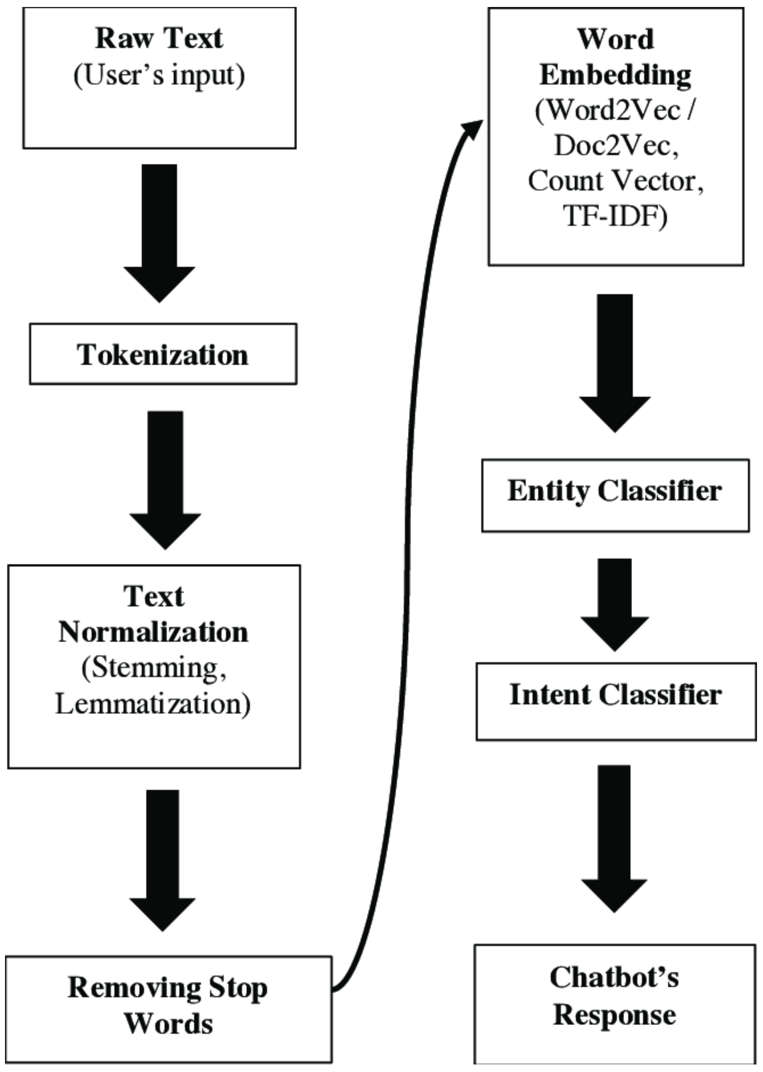


**Figure 7.1: DATA FLOW DIAGRAM**

# CHAPTER 8

# ADVANTAGES AND DISADVANTAGES

## 8.1 ADVANTAGES

1. **Accessible Information:** Provides easy access to information from Wikipedia on various topics, enhancing knowledge exploration.
2. **Conversational Interface:** Offers a user-friendly conversational interface for interacting with the chatbot, making it intuitive to use.
3. **Efficiency:** Automates the process of retrieving and presenting information, saving time and effort for users.
4. **Customizable Responses:** Can be tailored to provide short or detailed responses based on user preferences, enhancing user experience.
5. **Scalability:** Can handle multiple user queries simultaneously, accommodating a large user base.

## 8.2 DISADVANTAGES

1. **Reliability on Wikipedia:** Relies heavily on the availability and accuracy of information on Wikipedia, which may vary or change over time.
2. **Limited Understanding:** May struggle to understand complex queries or nuances in user language, leading to inaccurate or irrelevant responses.
3. **Dependency on Web Scraping:** Vulnerable to changes in Wikipedia's HTML structure or content layout, requiring frequent updates to maintain functionality.
4. **Privacy Concerns:** May raise privacy concerns if user queries or interactions are logged or stored without consent.
5. **Lack of Personalization:** Lacks personalization features, such as user profiles or preferences, limiting its ability to tailor responses to individual users' needs.

# CHAPTER 9

# CONCLUSION AND FUTURE ENHANCEMENT

## 9.1 CONCLUSION

"In conclusion, the development of a wiki chatbot using Convolutional Neural Networks represents a significant advancement in natural language processing and information retrieval. By leveraging CNNs, we were able to create a chatbot that can effectively understand and respond to user queries, providing relevant information from a vast knowledge base such as Wikipedia.

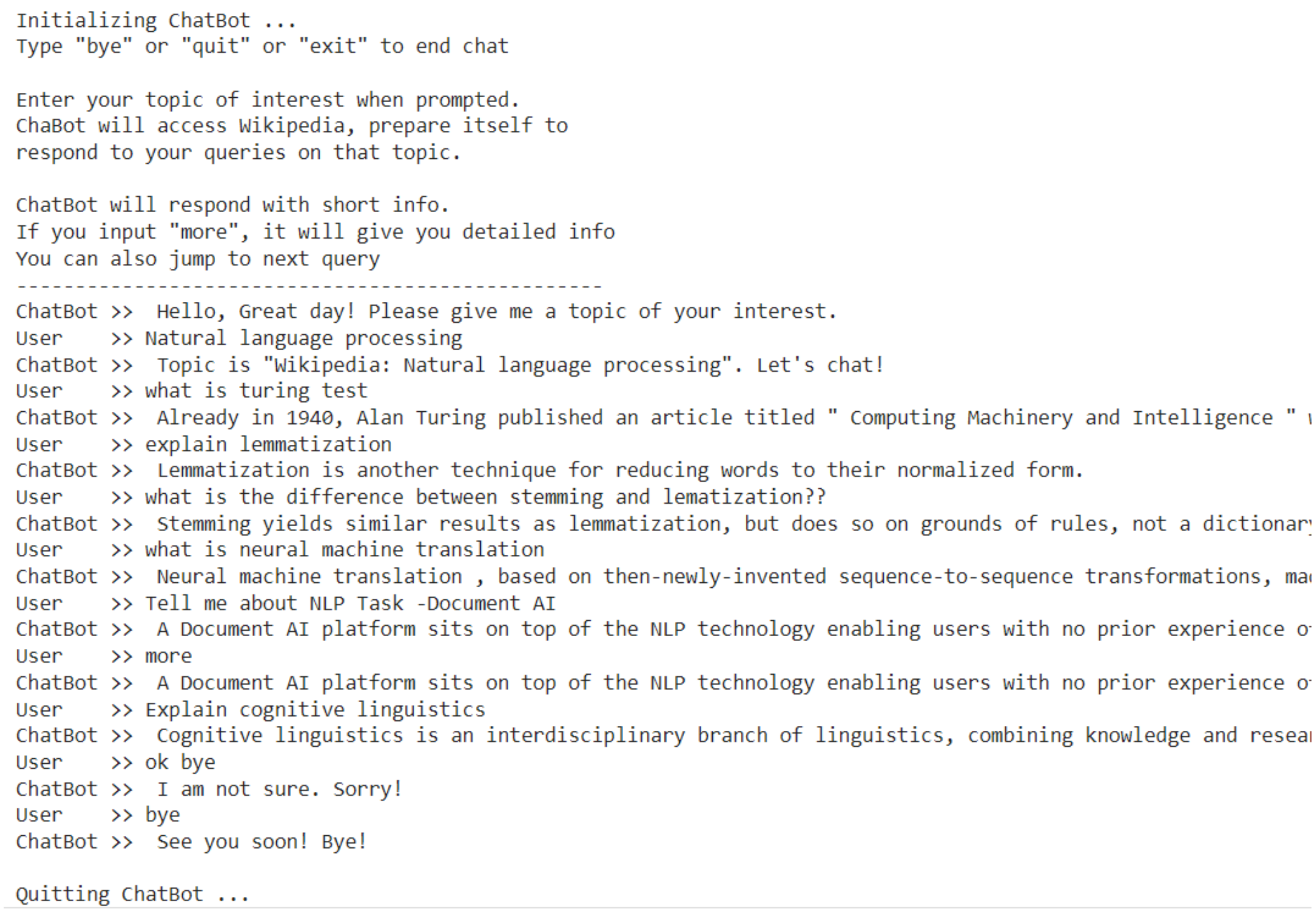
Through this project, we have demonstrated the feasibility and effectiveness of using CNNs in chatbot development, highlighting their ability to handle complex textual data and extract meaningful information. The chatbot's accuracy and efficiency in retrieving information showcase its potential to enhance user experiences in various applications, including customer service, education, and research.

Looking ahead, further enhancements can be made to the chatbot's architecture and training process to improve its performance and versatility. Additionally, integrating other advanced technologies such as natural language understanding (NLU) and reinforcement learning can further enhance the chatbot's capabilities and make it more adaptable to different contexts and user needs.

## 9.2 FUTURE ENHANCEMENT:

1. **Natural Language Understanding:** Improve the chatbot's ability to understand and respond to natural language queries more accurately. This could involve using more advanced NLP techniques or training on larger datasets.
2. **Knowledge Base Expansion:** Continuously update and expand the chatbot's knowledge base to provide users with more comprehensive and up-to-date information.
3. **User Interaction:** Enhance the chatbot's user interaction capabilities, such as offering personalized recommendations, providing multi-turn conversations, or supporting voice input.
4. **Integration with Other Platforms:** Integrate the chatbot with other platforms and services to improve its accessibility and usefulness. For example, integrating with messaging apps or voice assistants.
5. **Performance Optimization:** Optimize the chatbot's performance, such as reducing response times, improving scalability, and enhancing reliability.
6. **User Feedback Mechanism:** Implement a mechanism for users to provide feedback on the chatbot's responses, which can be used to improve its accuracy and effectiveness over time.
7. **Multilingual Support:** Add support for multiple languages to make the chatbot accessible to a wider audience.

## APPENDIX SCREENSHOTS



```
Initializing ChatBot ...
Type "bye" or "quit" or "exit" to end chat

Enter your topic of interest when prompted.
ChaBot will access Wikipedia, prepare itself to
respond to your queries on that topic.

ChatBot will respond with short info.
If you input "more", it will give you detailed info
You can also jump to next query
--------------------------------------------------
ChatBot >>  Hello, Great day! Please give me a topic of your interest.
User    >> Natural language processing
ChatBot >>  Topic is "Wikipedia: Natural language processing". Let's chat!
User    >> what is turing test
ChatBot >>  Already in 1940, Alan Turing published an article titled " Computing Machinery and Intelligence " 
User    >> explain lemmatization
ChatBot >>  Lemmatization is another technique for reducing words to their normalized form.
User    >> what is the difference between stemming and lematization??
ChatBot >>  Stemming yields similar results as lemmatization, but does so on grounds of rules, not a dictionar
User    >> what is neural machine translation
ChatBot >>  Neural machine translation , based on then-newly-invented sequence-to-sequence transformations, ma
User    >> Tell me about NLP Task -Document AI
ChatBot >>  A Document AI platform sits on top of the NLP technology enabling users with no prior experience o
User    >> more
ChatBot >>  A Document AI platform sits on top of the NLP technology enabling users with no prior experience o
User    >> Explain cognitive linguistics
ChatBot >>  Cognitive linguistics is an interdisciplinary branch of linguistics, combining knowledge and resea
User    >> ok bye
ChatBot >>  I am not sure. Sorry!
User    >> bye
ChatBot >>  See you soon! Bye!

Quitting ChatBot ...
```

## REFERENCES:

**[1] Gradient-based learning applied to document recognition**

LeCun, Y. | Bottou, L. | Bengio, Y. | Haffner, P.

**[2] Convolutional neural networks for sentence classification.**

Kim, Y.

**[3] A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification**

Zhang, Y., & Wallace, B.

**[4] Torch7: A MATLAB-like environment for machine learning**

Collobert, R., Kavukcuoglu, K., & Farabet, C.

**[5] Short text classification with recurrent neural networks and word embeddings**

Kenter, T., & de Rijke, M.

**GITHUB LINK:** **https://github.com/renga2408/WIKI-CHATBOT-USING-CNNs**