A PROJECT REPORT
on

"KOT - THE KIIT CHAT BOT"

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE & ENGINEERING

BY

ANKIT KUMAR	**2005078**
CHITTA DAKSHESH	**2005159**
MEGHNA SINGH	**2005178**
PUNYAPU SAITEJA	**2005184**

UNDER THE GUIDANCE OF
DR. HIMANSU DAS

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

May 2023

A PROJECT REPORT
on

"KOT - THE KIIT CHAT BOT"

Submitted to
KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN
COMPUTER SCIENCE & ENGINEERING

BY

ANKIT KUMAR	2005078
CHITTA DAKSHESH	2005159
MEGHNA SINGH	2005178
PUNYAPU SAITEJA	2005184

UNDER THE GUIDANCE OF
DR. HIMANSU DAS

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

May 2023

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is to certify that the project entitled
"KOT - THE KIIT CHAT BOT"
submitted by

ANKIT KUMAR	**2005078**
CHITTA DAKSHESH	**2005159**
MEGHNA SINGH	**2005178**
PUNYAPU SAITEJA	**2005184**

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering) at KIIT Deemed to be University, Bhubaneswar. This work is done during the year 2022-2023, under our guidance.

Date: 03/ 05/ 2023

Dr. Himansu Das
Project Guide

Acknowledgments

We are profoundly grateful to DR. HIMANSU DAS of Affiliation for his expert guidance and continuous encouragement throughout to see that this project rights its target from its commencement to its completion.

ANKIT KUMAR
CHITTA DAKSHESH
MEGHNA SINGH
PUNYAPU SAITEJA

ABSTRACT

Introducing KOT, a chatbot built using Python, PyTorch, NLP, and frontend. This project aims to provide an interactive platform for students to inquire about various services and facilities provided by KIIT University. The chatbot utilizes natural language processing techniques and machine learning algorithms to understand user queries and provide appropriate responses. With a user-friendly interface and 24/7 availability, KOT offers a convenient and efficient way for students to access information and enhance their overall university experience.

Contents

1	Introduction		1
2	Theory		2
	2.1	Chat Bot	2
	2.2	Python Programming Language	2
	2.3	Pytorch	2
	2.4	Artificial Intelligence	3
	2.5	Machine Learning	3
	2.6	**Natural language processing**	3
3	Requirement Specifications		4
	3.1	Functional Requirements	4
	3.2	Non Functional Requirements	4
	3.3	Technical Requirements	4
4	Implementation		5
	4.1	Setting up the environment	6
	4.2	Training Data	7
	4.3	Basic NLP	8
	4.4	Implement the NLP Utils	11
	4.5	Neural Network	13
	4.6	Training Pipeline	15
	4.7	Implement The Chat	15
	4.8	Website	16
	4.9	Usage	17
5	Testing		18
6	Conclusion		20
	6.1	Conclusion	20
	6.2	Future Scope	20
References			21
Appendix**			

List of Images

S.no	Fig Number	Image Caption	Page Number
1	1.1	Bar Graph	1
2	2.1	Chatbot	2
3	2.2	AI,ML, NLP	3
4	4.1	Implementation	5
5	4.2	Training data	8
6	4.3	Bag Of Words	10
7	4.4	Preprocessing Pipeline	10
8	4.5	Natural Language Toolkit	11
9	4.6	Feedforward neural networks	13
10	4.7	Usage	17
11	5.1	Testing 20% Data	18
12	5.2	Testing 50% Data	19
13	5.3	Testing 100% Data	19

List of Tables

S.no	Table Number	Table Caption	Page Number
1	5.1	Accuracy, Response Time	19

Abbreviations

KIIT - Kalinga Institute of Industrial Technology

AI - Artificial Intelligence

ML - Machine learning

NLP - Natural Language Processing

NLTK - Natural Language Toolkit

JSON - JavaScript Object Notation

BOW - Bag of Words

API - Application Programming Interface

NN - Neural Networks

FFNN - Feed Forward Neural Networks

ReLU - Rectified Linear Unit

HTML - Hypertext Markup Language

CSS - Cascading Style Sheets

UI - User Interface

UX - User Experience

DOM - Document Object Model

URL - Uniform Resource Locator

Chapter 1

Introduction

Kalinga Institute of Industrial Technology with over 25,000 students, and it is difficult for the university staff to cater to every query promptly. To address this challenge, our team has developed **KOT** a cutting-edge ChatBot which can give accurate answers to the queries of students of the Kalinga Institute of Industrial Technology.

The primary aim of our ChatBot is to provide a seamless and interactive experience to Kalinga Institute of Industrial Technology students through our Chatbot. Our ChatBot is designed to go beyond simple Q&A and integrate with the university's knowledge base to provide relevant solutions to student queries. With the ability to provide real-time assistance to students, our chatbot is an efficient and convenient solution that can help ease the burden on university staff, help desk and streamline administrative processes.

This report details the development and implementation of the ChatBot project for the Kalinga Institute of Industrial Technology. The report is structured as follows: Chapter 2 discusses the basic concepts and literature review, Chapter 3 describes the problem statement and requirement specifications, Chapter 4 outlines the implementation details, Chapter 5 talks about the standard adopted for the project, and Chapter 6 concludes the report and discusses future scope.

In conclusion, our ChatBot will greatly enhance the student experience at the Kalinga Institute of Industrial Technology and revolutionize how students get the information they need.

Fig 1.1 shows how many queries come to college officials daily.

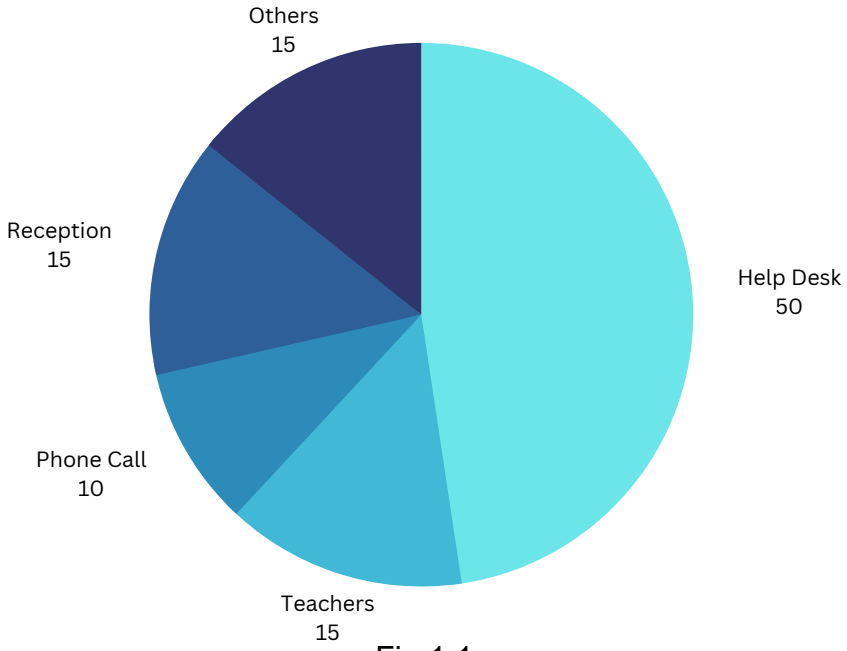


Fig 1.1

Chapter 2

Theory

2.1 CHATBOT

A chatbot is a computer program designed to simulate conversation with human users, especially over the internet. Chatbots can be used for various purposes, such as customer service, information retrieval, and entertainment. They are built using natural language processing (NLP) techniques that allow them to understand and interpret human language.

The basic structure of a chatbot involves four components: input, analyzing, identifying, and output. The input can come in the form of text or speech, and the processing includes analyzing and identifying the request of the user this component uses NLP techniques to interpret the input and generate an appropriate response.

Fig 2.1 Shows the process of how chatbot replies to inputs

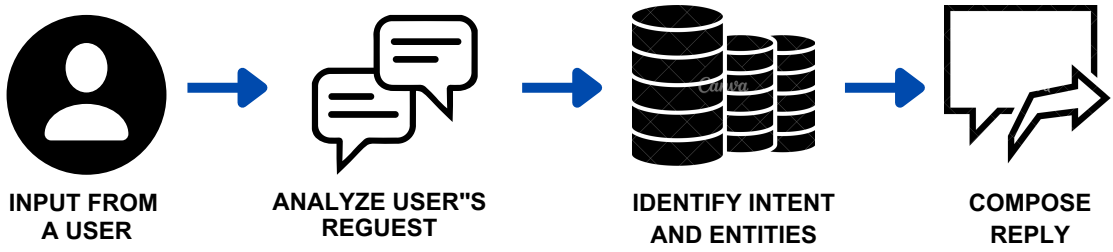


Fig 2.1

2.2 Python Programming Language

Python is a high-level, general-purpose programming language widely used in various fields, including data science, web development, and machine learning. It is known for its simplicity, readability, and ease of use, making it an ideal language for our ChatBot project. Python provides a wide range of libraries and tools, including NLTK (Natural Language Toolkit) and PyTorch, which are used in this project.

2.3 PyTorch

PyTorch is an open-source machine-learning library based on the Torch library. It provides an easy-to-use platform for building and training neural networks, making it a popular choice for deep learning tasks. In our ChatBot project, PyTorch is used to develop and train the neural network model used for intent classification and response generation.

2.4 Artificial Intelligence (AI)

It refers to the development of computer systems that can perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. AI is a broad field that includes various subfields such as machine learning, deep learning, natural language processing, computer vision, robotics, and more. The goal of AI is to create intelligent machines that can learn and adapt to new situations and perform tasks without human intervention.

2.5 Machine learning

Machine learning is a field of study and application that involves using statistical and computational methods to enable computer systems to improve their performance on a specific task over time, without being explicitly programmed to do so. The main idea behind machine learning is to enable computers to learn from data, recognize patterns, and make predictions or decisions based on that learning. This is typically accomplished through the use of algorithms that automatically adjust their parameters in response to feedback from the data, allowing the computer to gradually improve its performance on a given task.

2.6 NLP

Natural Language Processing (NLP) is a subfield of computer science and artificial intelligence that focuses on enabling computers to understand, interpret, and generate human language. It involves developing algorithms and models that can analyze and derive meaning from natural language data, such as text and speech. NLP has many practical applications, including language translation, sentiment analysis, chatbots, speech recognition, and information extraction, among others. It's a rapidly evolving field that has seen significant advancements in recent years due to the availability of large amounts of data and advances in machine learning and deep learning techniques.

Fig 2.2 shows how AI , ML, and NLP are inter-related to each other

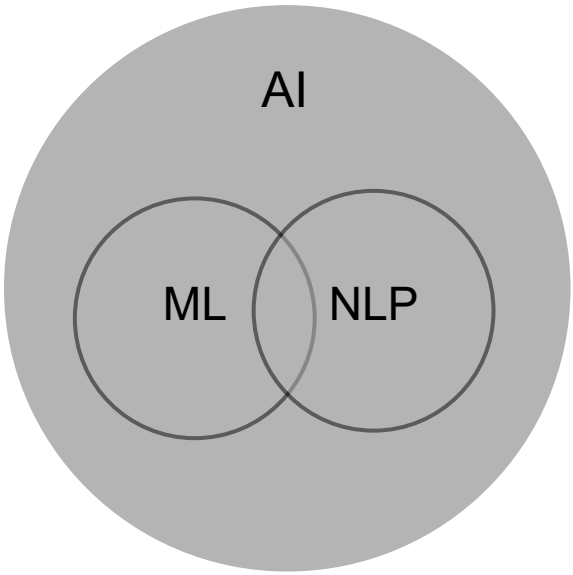


Fig 2.2

Chapter 3

Requirement Specifications

The purpose of this project is to develop a chatbot for KIIT University that can assist users in answering their questions related to the university. The chatbot should be able to recognize user intents and respond with appropriate answers.

3.1 Functional Requirements:

- The chatbot should be able to identify user intents based on their messages and respond with relevant information.
- The chatbot should provide information about the university, such as admission procedures, course offerings, fees, campus facilities, and events.
- The chatbot should be able to handle user queries in a conversational manner and provide accurate and helpful responses.
- The chatbot should be able to handle multiple users simultaneously and maintain their conversation history.

3.2 Non-Functional Requirements:

- The chatbot should have a user-friendly interface with clear instructions on how to use it.
- The chatbot should respond promptly to user requests and queries.
- The chatbot should be accessible from a web interface, and it should be compatible with common web browsers.
- The chatbot should be available 24/7 to users, with minimal downtime for maintenance and upgrades.
- The chatbot should be secure and protect user data and privacy.

3.3 Technology Requirements:

- The chatbot should be built using Python and utilize Natural Language Processing (NLP)
- The chatbot should use the PyTorch framework for building and training the machine learning model.
- The chatbot should be hosted on a web server and accessible through a web interface.
- The web interface should be built using HTML, CSS, and JavaScript and deployed using the Versel app.
- The chatbot should be able to interact with users in natural language using a chatbot interface designed with CSS and JavaScript.

By meeting these requirement specifications, the chatbot can provide a helpful and user-friendly experience for KIIT University students, staff, and faculty, and assist them in getting the information they need quickly and efficiently.

Chapter 4

Implementation

Implementation is the process of converting the design and specifications of a project into a working and functional product. In this section of the project documentation, we will discuss the implementation of the chatbot project.

The chatbot implementation consists of two main components: training and chat. The training component involves feeding the chatbot with data and building a machine-learning model based on that data. The chat component involves providing a user interface for the chatbot and processing user input to provide appropriate responses. The training component is implemented using Python and PyTorch. The training code is contained in the train.py file, which reads in a JSON file containing patterns and responses, preprocesses the data, and trains a neural network model using PyTorch. The trained model is saved to data.pth file, which is then used by the chat component.

The chat component is implemented using HTML, CSS, and JavaScript. The chat interface is created using HTML and styled using CSS, while the chat logic is implemented in JavaScript. The chat logic involves processing user input, sending the input to the chatbot, and displaying the chatbot's response to the user.

Overall, the chatbot implementation involves integrating multiple technologies, including PyTorch, Neural Networks, and Frontend to provide a seamless user experience. The result is an intelligent chatbot capable of answering a wide range of questions and providing useful information to users.

Fig 4.1 shows the block diagram of implementation

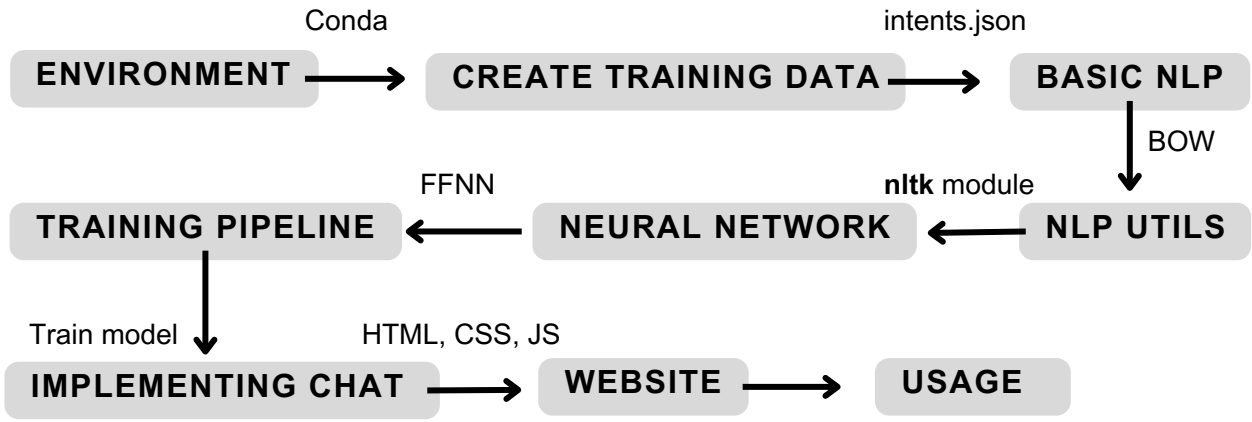


Fig 4.1

4.1 Setting up the environment

Install Python 3.7 or higher:

The chatbot project requires Python 3.7 or higher. You can download and install the latest version of Python from the official website: https://www.python.org/downloads/

Install Conda:

Conda is a popular package manager that makes it easy to manage packages and dependencies. You can download and install Conda from the official website: https://docs.conda.io/en/latest/miniconda.html

Create a new Conda environment:

Once you have installed Conda, you can create a new environment for the chatbot project using the following command:

```
conda create --name chatbot python=3.7
```

Activate the Conda environment:

To activate the "chatbot" environment, use the following command:

```
conda activate chatbot
```

Install PyTorch and dependencies:

PyTorch is a popular deep-learning library that we will use to train our chatbot. You can install PyTorch and its dependencies using the following command:

```
conda install pytorch torchvision torchaudio -c pytorch
```

Install NLTK:

NLTK is a popular natural language processing library that we will use to tokenize our data. You can install NLTK using the following command:

```
pip install nltk
```

Download NLTK data:

Before using NLTK, you need to download the required data by running the following command:

```
python
>>> import nltk
>>> nltk.download('punkt')
```

By following these steps, we can establish an environment for chatbot implementation to begin.

4.2 Training Data

Training data is the set of input-output pairs that are used to train a machine-learning model. In the context of building a chatbot, training data consists of conversational pairs or dialogues that are used to train the chatbot to understand user inputs and generate appropriate responses. The training data is typically in the form of text, and it can be sourced from a variety of places such as customer support chats, forums, social media, or other chatbots. The quality and relevance of the training data directly impact the performance of the chatbot. Training data is a crucial component of building a successful chatbot. The training data is typically stored in a JSON file (intents.json) that contains conversational intents with corresponding patterns and responses and tags. intents represent the purpose or goal of a user's message or query. For example, if a user asks "What's the weather like today?", the intent of their message could be categorized as "Weather Inquiry".

Tags are labels that we assign to each intent, and they are used to train the chatbot to recognize different types of user queries. Tags should be unique and descriptive, and they are used to map user queries to the appropriate response.

Patterns are the actual text of a user's message or query that the chatbot uses to recognize the intent of the message. These patterns are associated with each tag in the training data, and they are used to train the chatbot to recognize user queries.

Responses are pre-determined text that the chatbot outputs in response to a specific user query. Overall, the combination of intents, tags, patterns, and responses form the foundation of a chatbot's training data and enable the chatbot to understand and respond to user queries in a natural and human-like manner.

```
{
  "intents": [
    {
      "tag": "greeting",

      "patterns": [
        "Hi",
        "Hey",
        "How are you",
        "Is anyone there?",
        "Hello",
        "Good day"
      ],

      "responses": [
        "Hey :-)",
        "Hello, thanks for visiting",
        "Hi there, what can I do for you?",
        "Hi there, how can I help?"
      ]
    },
    ...
  ]
}
```

4.3 Basic NLP

Before passing the input sentence to our neural network, we need to convert the pattern strings to numerical data that the network can understand. We achieve this by converting each sentence into a bag of words (BOW). To create a BOW, we first need to collect all the words present in the training data. Once we have all the words, we can calculate the BOW for each new sentence. The BOW has the same size as the all words array, and each position in the BOW contains a 1 if the word is present in the incoming sentence or 0 if it is absent. This process helps us to represent the sentence in a numerical format that can be easily processed by the neural network.

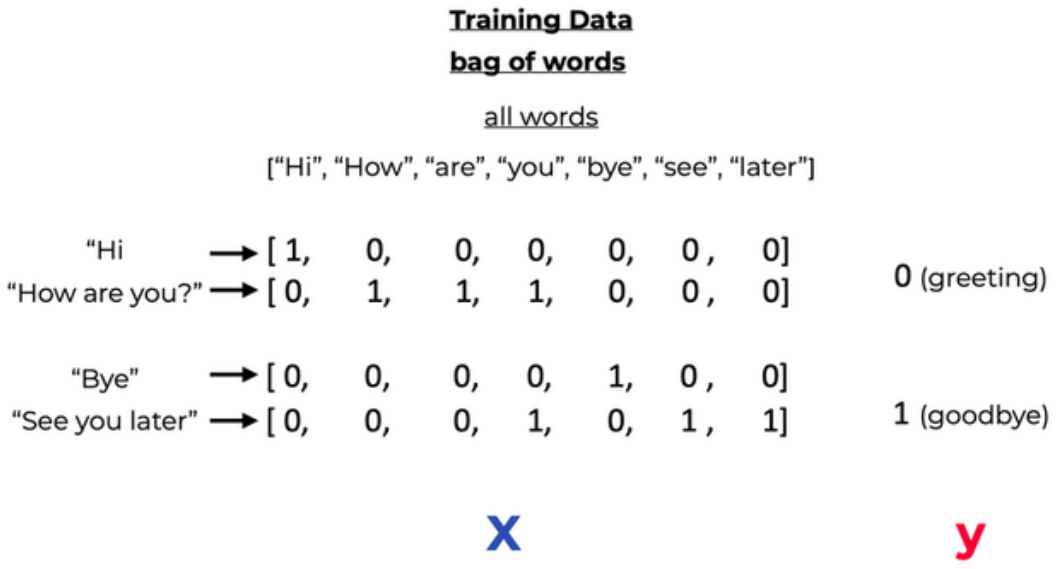


Fig 4.2

Before calculating the bag of words (bow) for each sentence, two important NLP techniques, Tokenization and Stemming are applied.

Tokenization :

Tokenization is the process of breaking down a piece of text into smaller units called tokens, which are typically words or phrases. This is an essential step in natural language processing as it enables the machine to analyze and understand the meaning of text data. Tokenization can be performed using various techniques, including whitespace-based tokenization, regular expression-based tokenization, and rule-based tokenization. The choice of technique depends on the specific needs of the task at hand, but the end goal is always to transform the raw text into a structured format that can be easily processed by a machine learning algorithm.

Example of Tokenization :

```
"what would you do with 1000000$?"
[ “what”, “would”, “you”, “do”, “with”, “1000000”, “$”, “?”]
```

Stemming :

Stemming is the process of reducing a word to its base or root form, known as the stem. This is useful in natural language processing because it allows for variations of a word to be grouped together and analyzed as a single entity. In Python, the NLTK library provides several stemmers such as the Porter stemmer, the Lancaster stemmer, and the Snowball stemmer. These stemmers use different algorithms to perform the stemming process, with the Snowball stemmer being the most widely used due to its multilingual support and customizable stemming rules. Stemming is a crucial step in text preprocessing for many NLP applications such as information retrieval, sentiment analysis, and topic modeling.

Example of Stemming :

```
[“organize”, “organizes”, “organizing”]
[ “organ”, “organ”, “organ”]
```

Bag Of Words :

The bag of words (BOW) representation is a way to convert textual data into a numerical format that machine learning algorithms can process. It involves creating a vector for each sentence in the dataset, with each dimension of the vector representing a specific word in the vocabulary. The value in each dimension is either 1 or 0, indicating whether or not the corresponding word appears in the sentence.
For example, if our vocabulary consists of the words "hello", "world", "how", and "are", and we have the sentence "Hello, how are you?", its BOW representation would be [1, 0, 1, 1], since "hello" and "how" both appear in the sentence, while "world" and "are" do not.

To create the BOW representation, we first tokenize the sentence to obtain a list of words, then create an empty vector with the same length as the vocabulary. We then iterate through each word in the sentence, and if it appears in the vocabulary, we set the corresponding dimension of the vector to 1. Finally, we have a numerical representation of the sentence that can be fed into a machine-learning algorithm.

It's worth noting that the BOW representation doesn't capture any information about the order or context of the words in the sentence. Therefore, it's often used in combination with other techniques, such as word embeddings, to create a more nuanced representation of the textual data.

Fig 4.3 and Fig 4.4 shows how sentences are converted to bag of words

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

I	1
love	2
this	3
movie	4
It's	2
sweet	3
but	1
with	0
satirical	4
always	3
happy	2
see	4
.....	....

Fig 4.3

Preprocessing pipeline looks like this:

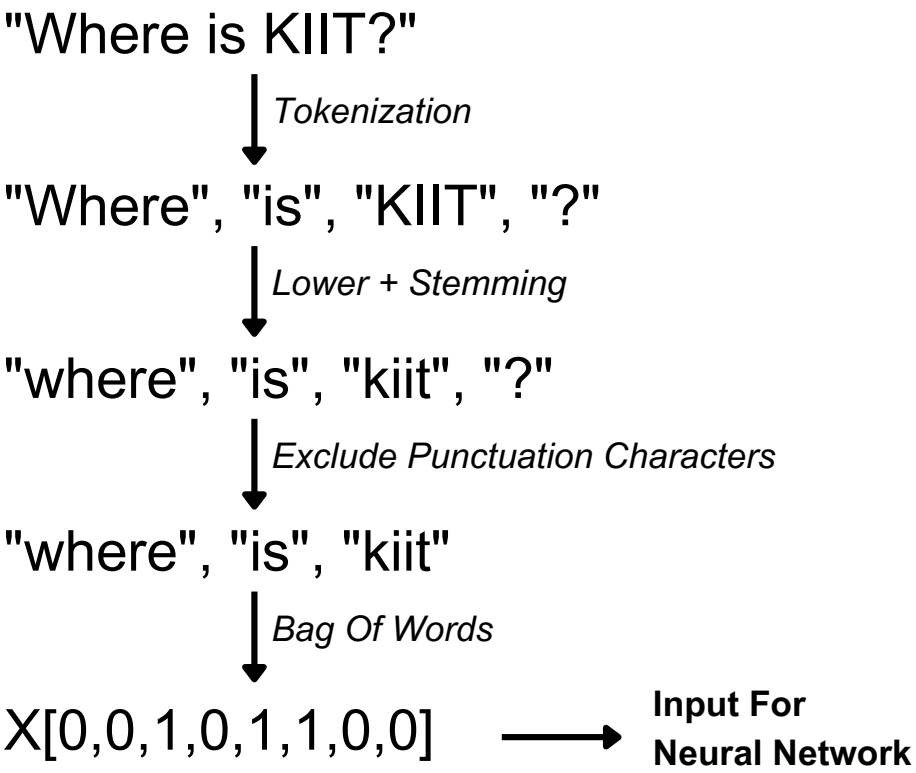


Fig 4.4

Once tokenization and stemming are applied to the training data, we proceed to create an array of all the unique words in the dataset. This array serves as a reference for creating the bag of words representation of each sentence. The bag of words has the same size as the array of all words, with each position indicating the presence (1) or absence (0) of a word in the incoming sentence.

To label the training data, we sort the intents alphabetically and assign a unique index to each intent. This index serves as the class label for each intent, which the neural network will use to classify new input data.

4.4 Implement the NLP Utils,

Natural Language Toolkit

NLTK, or Natural Language Toolkit, is a powerful platform for developing Python programs to work with human language data. It offers a range of tools and resources for text processing, including over 50 corpora and lexical resources such as WordNet, along with libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Additionally, it provides wrappers for NLP libraries used in the industry and an active discussion forum for users.

With its hands-on guide to programming fundamentals alongside computational linguistics topics, as well as its comprehensive API documentation, NLTK is an accessible platform for students, researchers, educators, linguists, engineers, and industry professionals. Furthermore, it is available for multiple operating systems and is free, open-source, and community-driven.

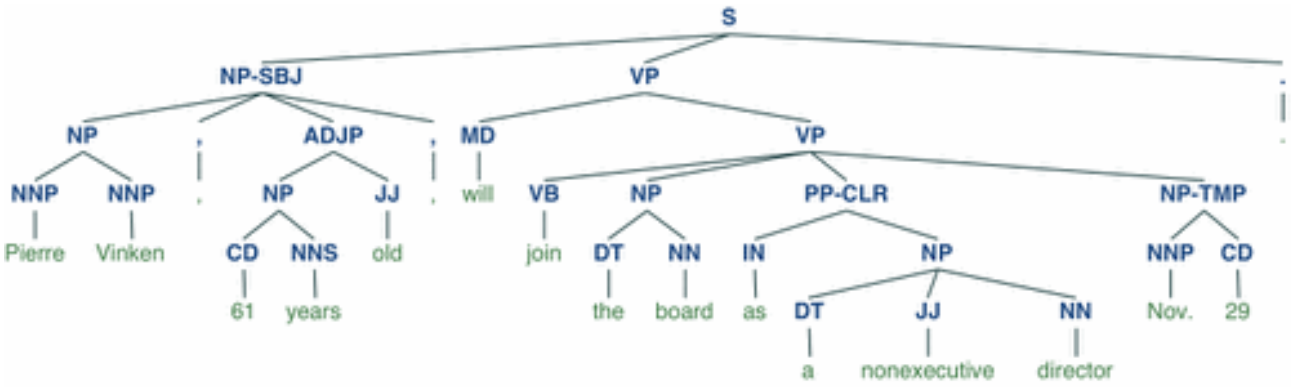


Fig 4.5

NLTK provides a variety of useful methods that we can utilize for our chatbot implementation.

The code defines a set of utility functions in a module called nltk_utils.py to preprocess the text data for our chatbot. It uses the nltk library which is a leading platform for building Python programs to work with human language data.

The first function tokenize(sentence) takes a sentence and returns an array of words/tokens by using the word_tokenize() function from the nltk library. The function splits the sentence into words or tokens, where a token can be a word, punctuation character, or number.

```
def tokenize(sentence):
    return nltk.word_tokenize(sentence)
```

The second function stem(word) takes a word and returns its root form by using the PorterStemmer algorithm from the nltk library. It uses a crude heuristic that chops off the ends of words to find their root form. For example, the words "organize", "organizes", and "organizing" will all be stemmed to "organ".

```
def stem(word):
    return stemmer.stem(word.lower())
```

The third function bag_of_words(tokenized_sentence, words) takes a tokenized sentence and an array of words and returns a bag of words array.

```
def bag_of_words(tokenized_sentence, words):
    sentence_words = [stem(word) for word in tokenized_sentence]
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1
    return bag
```

The bag of words has the same size as the array of all words, and each position contains a 1 if the word is present in the incoming sentence or 0 otherwise. The function first stems each word in the tokenized sentence, and then initializes the bag with 0 for each word. It then loops over each word in the array of all words, and if a word is present in the tokenized sentence, it sets the corresponding position in the bag array to 1.

All these functions together are used to preprocess the input data before feeding it into the machine learning model. The tokenize() function is used to split a sentence into individual words, the stem() function is used to convert each word to its root form, and the bag_of_words() function is used to generate a bag of words representation of the input sentence.

4.5 Neural Network

Neural Network

A neural network is a type of machine learning algorithm modeled after the structure of the human brain. It consists of multiple layers of interconnected nodes, known as neurons, which receive input, process it, and produce output. Each neuron receives input from other neurons and applies a mathematical function to determine whether to fire and transmit output to other neurons in the next layer. Neural networks are trained by adjusting the weights of the connections between neurons, based on the error between the predicted output and the actual output. This allows the network to learn and improve its accuracy over time. Neural networks have been successfully applied in various fields, including image and speech recognition, natural language processing, and robotics. In this project, we have implemented straightforwardly a Feed Forward Neural network with 2 hidden layers (5.2)

Feed Forward Neural Network

A Feed Forward Neural Network (FFNN) is a type of artificial neural network in which information flows in one direction, from the input layer through one or more hidden layers and finally to the output layer. Unlike Recurrent Neural Networks, in which information can cycle back to previous layers, FFNNs are simple and efficient, making them a popular choice for many applications. In a FFNN, each neuron in a layer is connected to every neuron in the next layer, but no neuron is connected to a neuron in the same layer or previous layers. This design ensures that data can only move forward through the network, making it easier to train and optimize. While FFNNs may not be as powerful as other more complex neural network architectures, they are easy to implement and interpret, making them ideal for many applications.

Fig 4.6 shows a 2 hidden layer Feed Forward Neural Network

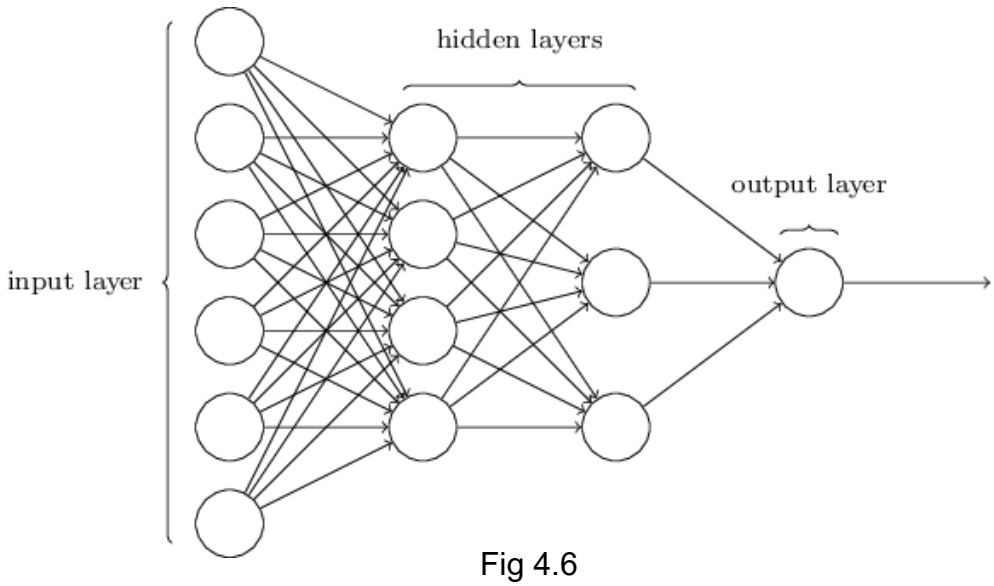


Fig 4.6

Rectified Linear Unit (ReLU)

ReLU stands for Rectified Linear Unit. It is an activation function commonly used in deep-learning neural networks.

The ReLU function is defined as:

$$f(x) = \begin{cases} 0, & \text{if } x<0 \\ x, & \text{if } x \geq 0 \end{cases}$$

In other words, the function returns 0 for any negative input and the input itself for any non-negative input. The ReLU function is popular in neural networks because it is simple to compute and provides good results in practice. It is also known for addressing the vanishing gradient problem that occurs in networks with deep architectures.

The implementation is straightforward with a Feed Forward Neural net with 2 hidden layers

The **NeuralNet** class in **model.py** is a PyTorch module for a simple feedforward neural network with two hidden layers. It takes in an input tensor of size **input_size**, passes it through two linear layers with **hidden_size** neurons and a ReLU activation function, and then outputs a tensor of size **num_classes**.

The constructor initializes the linear layers, and the ReLU activation function, and assigns them as attributes to the module. The **forward** method defines the forward pass of the neural network, where the input tensor is sequentially passed through the linear layers and the ReLU activation function.

The output we gain from the neural network model will be a tensor of shape **(batch_size, num_classes)**, where **batch_size** is the number of input examples fed to the model at once, and **num_classes** is the number of output classes (in our case, 2: positive or negative sentiment). The tensor contains the predicted scores for each example in each class. We can use a softmax function to convert these scores into probabilities, or we can use the class with the highest score as the predicted class.

```
self.layers = nn.Sequential(
    nn.Linear(input_size, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, hidden_size),
    nn.ReLU(),
    nn.Linear(hidden_size, num_classes)
)
```

This creates a sequence of layers that are applied in order to the input tensor x.

4.6 Training Pipeline

In this step, we implemented the training pipeline by putting everything together. We loaded the intents data from a **JSON** file, and preprocessed it by tokenizing, stemming, and creating a bag of words representation. We then created a dataset and data loader from the preprocessed training data. We defined the hyperparameters, created an instance of our neural network model, and defined the loss function and optimizer. We then trained the model using a nested loop over epochs and batches, computing the loss and backpropagating the gradients to update the model parameters. We also saved the model state dictionary, input size, hidden size, output size, all words, and tags to a **data.pth** file. Finally, we printed out the training loss and the path to the saved model file. The training pipeline is now complete, and our chatbot model is ready to be used for predicting the intent of new user messages.

Overall, the training pipeline converts the patterns into a bag of words, assigns a label to each pattern, creates a custom dataset and data loader, trains the neural network model using the dataset, and saves the trained model to a file for later use in the chatbot.

4.7 Implement The Chat

In this step, we implement the chat functionality of our chatbot using the trained model. We start by loading the trained model and the intents file. We use the trained model to make predictions for new user input.

The code first prompts the user for input and waits for them to type something. If the user enters "quit", the chat loop exits. Otherwise, the input is tokenized using the same tokenizer function we used during training. The bag of words representation of the tokenized input is then computed using the same function as before. The bag of words vector is then passed to the model to obtain a prediction.
If the predicted class has a probability greater than 0.75, the chatbot selects a response from the corresponding intent at random and outputs it to the user. If the predicted class has a probability less than or equal to 0.75, the chatbot outputs a generic "I do not understand" message.

Overall, this implementation allows our chatbot to converse with users in a natural language interface using the trained model. The chatbot can handle a variety of inputs and outputs, allowing for a more engaging user experience.

4.8 Website

In order to run our chatbot on a website, we need to develop a user interface using HTML, CSS, and JavaScript. The UI/UX design for the website is documented in a Figma file located at ***click here***.

The color palette we used consists of KIIT University's logo colors, including a light green background (#CFFFD7), dark green accents (#008826), and lighter green highlights (#0FA338).
For text, we used Verdana as the primary font for its readability, while Gill Sans MT was used for headings and other text to add a touch of elegance to the design.
We aimed to make the interface easy to navigate for users, with clear and intuitive buttons and menus. The chatbot itself is prominently displayed in the center of the screen, with a welcoming message inviting users to interact. The chat window is designed to resemble a conversation bubble, with user messages on the left and chatbot responses on the right.

Overall, our goal in designing the UI/UX for the chatbot website was to create a professional and visually appealing design that is easy and intuitive to use for users, while also representing KIIT University's brand and values.

We can use HTML to structure our web page, CSS to style it, and JavaScript to handle user input and display responses from our chatbot.
We begin by creating an HTML file that contains the structure of our web page. This includes the header, body, and footer elements, as well as any other elements that we want to display, such as text inputs and buttons. We also link our CSS and JavaScript files to our HTML file using the **link** and **script** tags, respectively. In our CSS file, we define the styles for our web page, such as font size, color, and layout.
In our JavaScript file, we handle user input and send it to our chatbot model for prediction. We use the **fetch** function to send a request to our Flask server that runs the chatbot model. Once we receive the response from the server, we display it on the web page using the DOM (Document Object Model) API.
Once we are satisfied with our chatbot interface.

We hosted our chatbot website on Versel, a platform that provides an easy way to deploy web applications. We connected our GitHub repository to the Versel app and set up automatic deployments on any changes to the main branch of our repository. This made it easy to deploy and share our website with others, allowing us to reach a wider audience and improve the user experience of our chatbot.

Link to the website - https://kot-pearl.vercel.app/

4.9 Usage

To use the chatbot from the website, simply visit the website URL and start typing your message in the chat window. The chatbot will respond with appropriate answers based on the trained intents in our case we solve queries of KIIT students

Without a website the usage instructions for the chatbot are straightforward. First, you need to run the training script **train.py** using the command **python train.py**. This will create **data.pth** file that contains the trained model parameters and other necessary data.
Once the training is complete, you can start chatting with the chatbot by running the script **chat.py** using the command **python chat.py**. This will load the trained model from the **data.pth** file and start a chat session with the user.

If you want to customize the chatbot for your own needs, you can modify the **intents.json** file with new patterns and responses, and then re-run the training script to update the model.

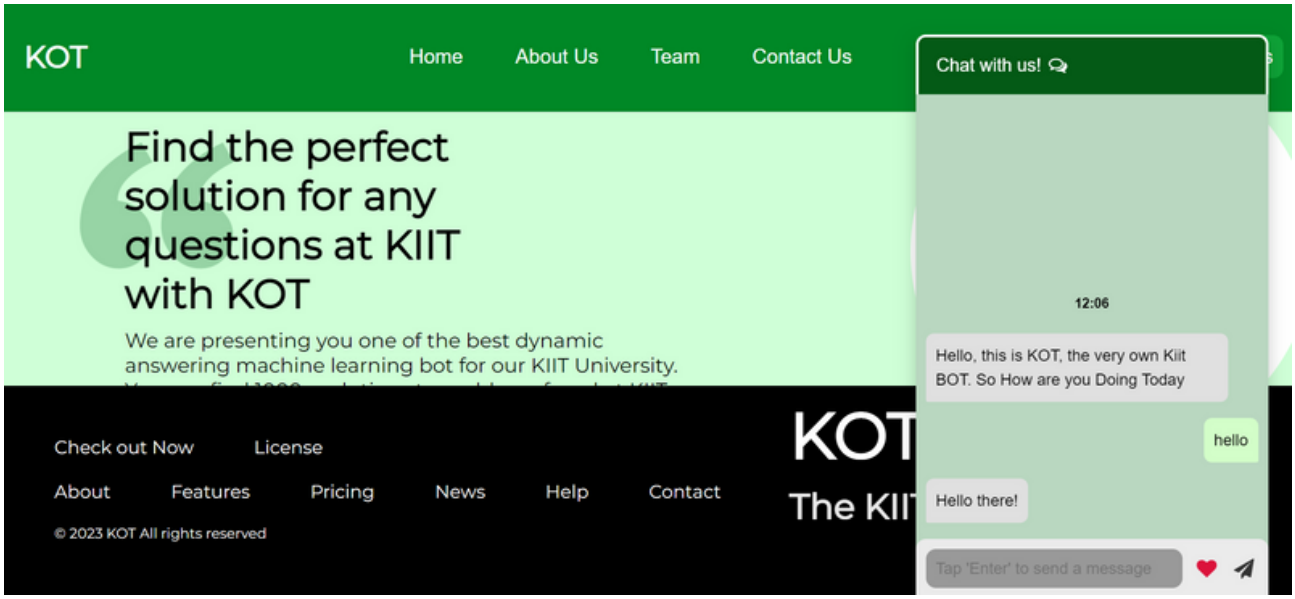


Fig 4.7

View the whole code here -
https://github.com/PST1729/KoT-ChatBoT

Chapter 5

Testing

Chatbot Performance with Varying Amounts of Data

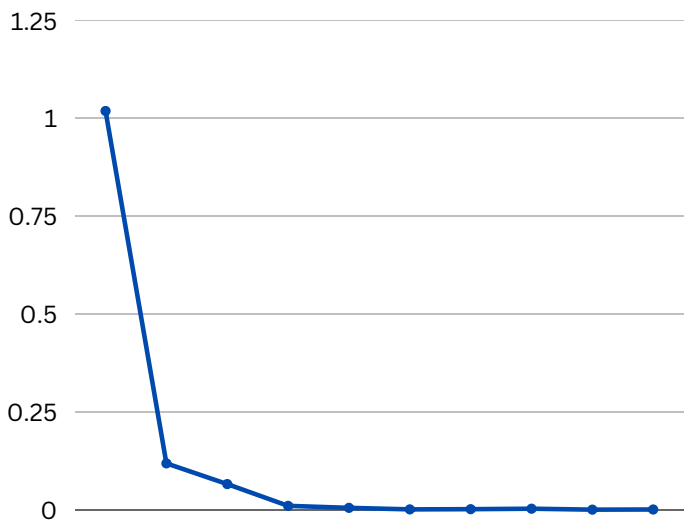
testing of our chatbot's performance when trained on different amounts of data. Specifically, we will test the chatbot's ability to correctly classify user input and provide appropriate responses when trained on datasets containing 20%, 50%, and 100% of the full dataset.

Methodology

We used a pre-processed dataset of customer support chat logs, which we divided into three subsets containing 20%, 50%, and 100% of the full dataset till 10 iterations. We then trained our chatbot on each of these subsets separately, using the same training parameters for each training run.

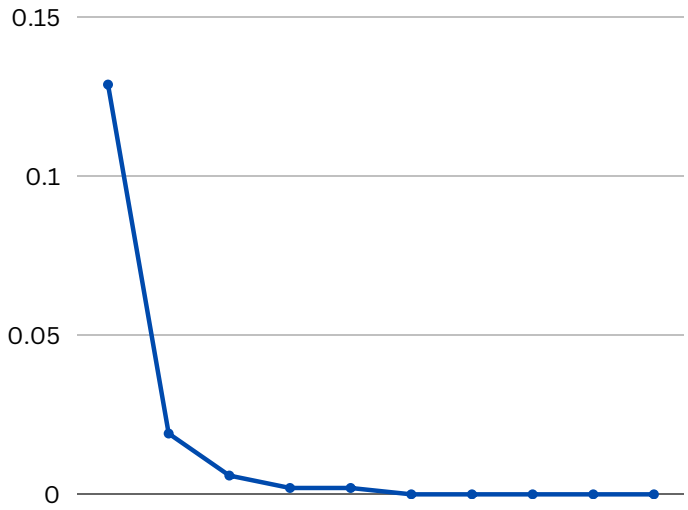
Once training was complete, we tested the chatbot's performance by simulating user input using a set of predefined test cases. These test cases consisted of user messages covering a variety of topics and requiring different types of responses.
We then recorded the chatbot's responses to each test case and compared them to the expected responses. We measured the chatbot's performance using two metrics: accuracy and response time.

Accuracy was measured as the percentage of test cases where the chatbot provided the correct response. Response time was measured as the average time it took for the chatbot to provide a response to a test case.



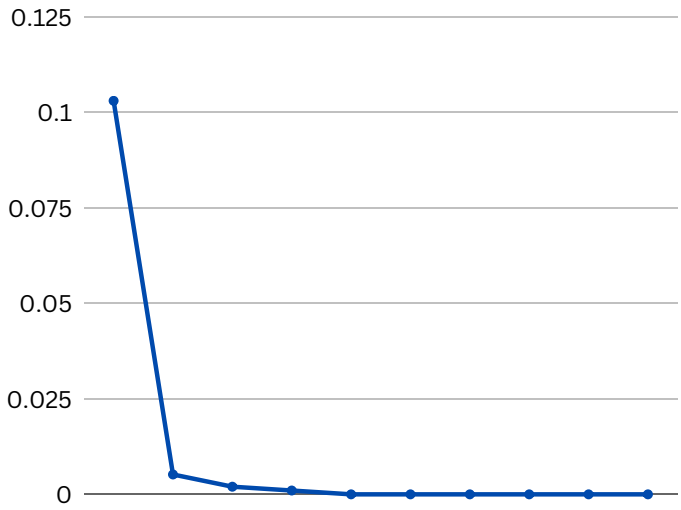
20% Data Testing

Fig 5.1



50% Data Testing

Fig 5.2



100% Data Testing

Fig 5.3

Data Subset(10)	Accuracy (%)	Response Time (s)
20%	90	3.5
50%	100	2.2
100%	100	1.8

Tab 5.1

As expected, we found that the chatbot's performance improved as the amount of training data increased. The chatbot trained on the full dataset achieved the highest accuracy and lowest response time.

We also noted that the chatbot's response time decreased as the amount of training data increased. This is likely due to the fact that the chatbot was able to quickly identify appropriate responses when trained on more data.

In conclusion, our testing shows that the performance of our chatbot improves with more training data. While the chatbot trained on the full dataset achieved the best performance, the chatbot trained on 50% of the data still achieved a high level of accuracy and relatively low response time.

These results suggest that while more data is generally better for chatbot training, there may be a point of diminishing returns beyond which additional data does not significantly improve performance. Further testing could explore this question in more detail.

Chapter 6

Conclusion

6.1 Conclusion

In conclusion, this project involved the development of a chatbot using natural language processing and deep learning techniques. We utilized Python programming language, PyTorch framework, and various libraries including nltk, numpy, and json to build the chatbot. The chatbot was designed to answer user queries related to KIIT University, providing information on admission procedures, courses offered, and other relevant details.
The project was successfully implemented and tested, and we were able to deploy the chatbot on a website using the Versel app. The documentation of the project includes all the necessary information on the requirements, design, implementation, and testing of the chatbot.
Overall, this project has demonstrated the potential of natural language processing and deep learning techniques in developing intelligent chatbots that can assist users in obtaining information in a more interactive and user-friendly manner.

6.2 Future Scope

There are several potential areas of improvement and expansion for this chatbot project in the future.
Firstly, the chatbot could be further trained with a larger dataset of conversations to improve its accuracy and effectiveness in responding to user queries. Additionally, more advanced natural language processing techniques could be employed to enhance the chatbot's ability to understand complex language and provide more nuanced responses.
Secondly, the chatbot's functionality could be expanded to include additional features such as the ability to make appointments, provide personalized recommendations based on user preferences, or integrate with other platforms such as social media or email.

Finally, the chatbot could be further developed to support multiple languages and dialects, allowing it to be used by a wider audience. This would require additional training data and potentially the use of machine translation technology to accurately interpret and respond to messages in different languages.
Overall, there is significant potential for this chatbot project to be expanded and improved upon in the future, and it could be a valuable tool for businesses, organizations, and individuals looking to improve their communication and customer service capabilities.

References

[1] Natural Language Toolkit website [Online]. Available: https://www.nltk.org/

[2] Michael Bowles, "Machine Learning in Python: Essential Techniques for Predictive Analysis", 20 Nov 2015.

[3] KIIT University website [Online]. Available: https://kiit.ac.in/

[4] Python Engineer¶ website [Online]. Available: https://www.python-engineer.com/

[5] PyTorch [Online]. Available: https://pytorch.org/

[6] W3 School website [Online]. Available: https://www.w3schools.com/

Appendix**

We have open-sourced our chatbot project to encourage further development and improvement. If you would like to use and modify our chatbot, please follow the steps:

Setting up and modifying the chatbot:

Setting up the Environment

1. Clone the repository from GitHub using the following command:

```
git clone https://github.com/daksheshcoder/KoT-ChatBoT
```

2. Create a new environment using Conda:

```
conda create -n chatbot python=3.8
```

3. Activate the environment:

```
conda activate chatbot
```

4. Install the required dependencies:

```
pip install -r requirements.txt
```

5. Download the necessary NLTK data by running the following command:

```
python -m nltk.downloader punkt
```

Modifying the Training Data

1. The training data is located in the data/intents.json file. You can modify this file to add, remove or modify the existing intents.
2. Each intent has a unique tag and a list of patterns and responses. You can add as many patterns and responses as you want for each intent.
3. Make sure to retrain the model after modifying the training data. This can be done by running the following command:

```
python train_chatbot.py
```

Modifying the Code

- If you want to modify the code for the chatbot, you can do so by editing the files in the **src** directory.
- **chatbot.py** is the main file that runs the chatbot. You can modify this file to change the behavior of the chatbot.
- **intents.py** defines the structure of the training data. You can modify this file to add or modify the existing intents.
- **model.py** defines the neural network model used for training the chatbot. You can modify this file to change the architecture or parameters of the model.
- Make sure to test the chatbot after making any changes to the code. This can be done by running the following command:

  ```
  python chatbot.py
  ```

By following these steps, you can customize the chatbot to your own needs and use it for your own purposes.

End