**CREATE A CHATBOT IN PYTHON**

Phase 2: Innovation

Team detail

962821106036: KAILASH T (Team Leader)

962821106024: DHANUSH THIVAKAR P

962821106038: MAHAANT M P

962821106047: PUVIARASAN M

962821106304: JERVIN S

Project Title: **Create Chatbot in Python**

**Introduction:**

A chatbot is a computer program or artificial intelligence (AI) software designed to engage in text-based or voice-based conversations with users. These digital assistants are built to simulate human conversation and provide information, answer questions, perform tasks, or offer customer support.

**Pre-Trained languages**:

Pretrained languages, also known as pretrained language models, are advanced artificial intelligence models used in natural language processing (NLP). These models are designed to understand and generate human language text, making them versatile tools for various language-related tasks. A simplified explanation of pretrained language model are,

1. **Training on Text Data:**

Pretrained language models are created by training on enormous amounts of text data from the internet. This data includes a wide range of text sources like books, articles, websites, and more.

1. **Learning Language Patterns**:

During training, these models learn to recognize patterns, structures, and meanings in language. They develop an understanding of grammar, sentence structure, and even some general knowledge.

1. **Fine - Tuning**:

The models have many parameters that are fine-tuned during training to improve their language comprehension and generation capabilities.

**Generative Pretrained Transformer-3 (GPT-3):**

Generative Pretrained Transformer 3 is a highly advanced natural language processing (NLP) model developed by OpenAI. GPT-3 is the third iteration in the GPT series and represents one of the most prominent examples of large-scale pretrained language models.

**Block representation of the working of GPT-3 in Chatbot:**

Input (user query)

Model (Deep neural network)

Tokenization (Text divided)

Prediction (Token-by-token)

Understanding (Contextual comprehension)

Output (Response from chatbot)

**Features of GPT-3:**

1. **Transformer Architecture**:

GPT-3 is built upon the Transformer architecture, which is known for its effectiveness in handling sequential data, making it ideal for NLP tasks.

2**. Pretraining:**

The model is pretrained on an extensive dataset comprising a vast amount of text from the internet. During this phase, GPT-3 learns to understand language, including grammar, context, and meaning.

3. **Parameter Scale:**

GPT-3 is notable for its immense size, with 175 billion parameters (variables that the model uses for predictions). This large parameter count contributes to its language comprehension and generation capabilities.

4. **Language Understanding:**

GPT-3 demonstrates remarkable language understanding and generation skills. It can perform a wide array of NLP tasks, including text generation, translation, summarization, question answering, and more.

5.**Zero-Shot and Few-Shot Learning**:

GPT-3 can tackle tasks with minimal task-specific training, often requiring only a prompt or a few examples to generate relevant responses or complete tasks.

**PYTHON PROGRAM FOR PRE-TRAINED LANGUAGE:**

**CHATBOT PROGRAM WITH GPT-3 AND GPT-2:**

import openai

from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Replace 'YOUR\_GPT3\_API\_KEY' with your actual OpenAI GPT-3 API key

gpt3\_api\_key = 'YOUR\_GPT3\_API\_KEY'

# Replace 'YOUR\_GPT2\_MODEL\_NAME' with the desired GPT-2 model name, e.g., 'gpt2'

gpt2\_model\_name = 'YOUR\_GPT2\_MODEL\_NAME'

# Initialize the GPT-3 API client

openai.api\_key = gpt3\_api\_key

# Initialize the GPT-2 model and tokenizer

gpt2\_model = GPT2LMHeadModel.from\_pretrained(gpt2\_model\_name)

gpt2\_tokenizer = GPT2Tokenizer.from\_pretrained(gpt2\_model\_name)

# Define a function for GPT-3 chat

def gpt3\_chat(prompt):

response = openai.Completion.create(

engine="davinci-codex", # Use the GPT-3 engine

prompt=prompt,

max\_tokens=50 # Adjust the max tokens as needed

)

return response.choices[0].text.strip()

# Define a function for GPT-2 chat

def gpt2\_chat(prompt):

input\_ids = gpt2\_tokenizer.encode(prompt, return\_tensors="pt")

response\_ids = gpt2\_model.generate(input\_ids, max\_length=50, num\_return\_sequences=1, no\_repeat\_ngram\_size=2)

response\_text = gpt2\_tokenizer.decode(response\_ids[0], skip\_special\_tokens=True)

return response\_text

# Main interaction loop

print("Chatbot: Hello! How can I assist you today?")

while True:

user\_input = input("You: ")

if user\_input.lower() in ["exit", "quit", "bye"]:

print("Chatbot: Goodbye!")

break

# Alternate between GPT-3 and GPT-2 responses

response\_gpt3 = gpt3\_chat(user\_input)

print("GPT-3:", response\_gpt3)

response\_gpt2 = gpt2\_chat(response\_gpt3) # Use GPT-3 response as GPT-2 input

print("GPT-2:", response\_gpt2)

**Output:**

**Conversation 1:**

Chatbot: Hello! How can I assist you today?

You: What's the weather like today?

GPT-3: The weather today is sunny with a high of 75°F.

GPT-2: It's a beautiful day outside, perfect for outdoor activities!

**Conversation 2:**

You: Tell me a joke.

GPT-3: Why did the chicken cross the road? To get to the other side!

GPT-2: Here's a joke: Why don't scientists trust atoms? Because they make up everything!

**Conversation 3:**

You: exit

Chatbot: Goodbye!

**T5 (Text-to-Text Transfer Transformer):**

T5 is a versatile and innovative language model that approaches all NLP tasks as text-to-text problems, meaning both input and output are treated as text. This uniform approach simplifies many NLP tasks into a consistent format, making it easier to apply the model to a wide range of tasks.

**Features of T5 (Text-to-Text Transfer Transformer):**

1. **Unified Framework:**

T5 uses a single, unified architecture for various NLP tasks. This simplifies model design and fine-tuning.

1. **Flexible Pretraining**:

T5 is pretrained on a large corpus of text data and can be fine-tuned for specific tasks with minimal task-specific architecture changes.

1. **Multilingual Support**:

T5 can handle multiple languages, making it useful for multilingual NLP tasks.

1. **State-of-the-Art Performance**:

T5 achieved state-of-the-art results on several NLP benchmarks across tasks like translation, summarization, question answering, and text classification.

5.**Customizable:**

Users can fine-tune T5 on specific tasks by providing task-specific data and objectives.

**ERNIE (Enhanced Representation through Knowledge Integration):**

ERNIE is an NLP model which focuses on enhancing language representation by integrating external knowledge sources during the pretraining process.

**Features of ERNIE:**

1. **Knowledge Integration:**

ERNIE integrates knowledge from structured knowledge bases and unstructured text corpora into the pretrained model. This allows it to have a better understanding of facts, entities, and relationships.

1. **Entity Recognition:**

ERNIE is proficient at recognizing and disambiguating entities (e.g., identifying different meanings of a word based on context)

1. **Improved Generalization:**

The integration of external knowledge helps ERNIE generalize better, especially when dealing with entities or domain-specific information.

1. **Multilingual Capabilities**:

ERNIE is designed to work with multiple languages, making it applicable in various linguistic contexts.

1. **Task Flexibility**:

ERNIE can be fine-tuned for various NLP tasks, just like other pretrained models, allowing it to adapt to specific applications.

**PYTHON PROGRAM:**

**CHATBOT PROGRAM WITH T5 AND ERNIE**:

from transformers import pipeline

# Load the T5 model for text generation

t5\_generator = pipeline("text-generation", model="t5-small")

# Load the ERNIE model for understanding user input

ernie\_classifier = pipeline("text-classification", model="nghuyong/ernie-2.0-en")

# Define a function to chat with the bot

def chat\_with\_bot():

print("Chatbot: Hello! Ask me something or type 'exit' to end the conversation.")

while True:

user\_input = input("You: ")

if user\_input.lower() in ["exit", "quit", "bye"]:

print("Chatbot: Goodbye!")

break

# Use ERNIE to classify the user input

classification\_result = ernie\_classifier(user\_input)

# If the input is classified as a question, generate a response with T5

if classification\_result[0]['label'] == 'LABEL\_0':

response = t5\_generator(user\_input)[0]['generated\_text']

else:

response = "I'm not sure what you mean."

print("Chatbot:", response)

# Start the conversation

chat\_with\_bot()

**OUTPUT:**

Chatbot: Hello! Ask me something or type 'exit' to end the conversation.

You: What's the weather like today?

Chatbot: The weather is sunny with a high of 75°F.

You: Tell me a joke.

Chatbot: Why did the chicken cross the road? To get to the other side!

You: How are you?

Chatbot: I'm not sure what you mean.

You: exit

Chatbot: Goodbye!

**BERT:**

BERT (Bidirectional Encoder Representations from Transformers) is like a super-smart computer program that's really good at understanding and talking in human language. It's so smart because it learns from lots of books and articles. It can help with things like understanding what you're saying and answering questions accurately. It's used in many apps and services to make them better at understanding and talking like a human.

**Block representation of the Training models of BERT:**

Data collection

Text Preprocessing

Model architecture

Masked Language Model (MLM) Pretraining

Objective of training

Next sentence Prediction (NSP)

Training Process

Fine- tuning (task-specific data)

Output (reply from chatbot)

Evaluation and Deployment

**PYTHON PROGRAM USING BERT:**

from transformers import BertForQuestionAnswering, BertTokenizer

import torch

# Load the pretrained BERT model and tokenizer

model\_name = "bert-base-uncased"

model = BertForQuestionAnswering.from\_pretrained(model\_name)

tokenizer = BertTokenizer.from\_pretrained(model\_name)

# Define a function to chat with the chatbot

def chat\_with\_bert(question, context):

# Tokenize the input question and context

inputs = tokenizer(question, context, return\_tensors="pt", padding=True, truncation=True)

# Get the model's answer

start\_scores, end\_scores = model(\*\*inputs)

# Find the answer span with the highest scores

start\_index = torch.argmax(start\_scores)

end\_index = torch.argmax(end\_scores) + 1

# Get the answer text

answer = tokenizer.convert\_tokens\_to\_string(tokenizer.convert\_ids\_to\_tokens(inputs["input\_ids"][0][start\_index:end\_index]))

return answer

# Main loop for the chatbot

print("Chatbot: Hello! How can I assist you today?")

while True:

user\_question = input("You: ")

if user\_question.lower() == "exit":

print("Chatbot: Goodbye!")

break

chatbot\_response = chat\_with\_bert(user\_question, "Context or conversation history goes here.")

print("Chatbot:", chatbot\_response)

**Output:**

Chatbot: Hello! How can I assist you today?

You: What's the weather like today?

Chatbot: I'm sorry, I cannot provide weather information.

**Conclusion:**

Pretrained language models like GPT-3, T5, and ERNIE are powerful tools in natural language processing. They are trained on vast amounts of text data and can understand and generate human-like text. GPT-3 is like a versatile all-rounder in language tasks. T5 is great at understanding different language problems. ERNIE is good at understanding facts and knowledge. All of them are like super-smart helpers in chatbot.