Research Project Report: AI-Powered Chatbot for Customer Service

Abstract

This report provides an in-depth look at the development, fine-tuning, and deployment of an AI-powered chatbot designed to enhance customer service interactions. The project utilized advanced NLP techniques, focusing on model fine-tuning using a pretrained language model (LLaMA-2-7B) and deployment via Flask, Docker, and CSS for an intuitive user interface. This report outlines the methodologies, challenges, solutions, and future prospects of the project.

---

1. Introduction

Artificial Intelligence (AI) is transforming how we work and live every day, from facial recognition to personalized learning. One of the significant advancements in AI is the development of chatbots powered by Large Language Models (LLMs). These chatbots can handle customer queries efficiently, providing instant responses and improving overall customer satisfaction.

This project aims to fine-tune a pre-trained LLaMA-2-7B model to create a custom chatbot tailored for customer service applications. The chatbot is designed to understand and respond to user queries accurately, making interactions seamless and efficient.

2. Problem Statement

The primary objective of this project is to create a custom chatbot using a fine-tuned LLaMA-2-7B model. The chatbot should be capable of handling a variety of customer service tasks, including answering frequently asked questions, providing information about products and services, and assisting with common issues.

3. Technical Approach

3.1 Model Selection

The project utilizes the LLaMA-2-7B model from Meta, a large language model known for its robust language understanding capabilities. This model was chosen for its ability to generate coherent and contextually relevant text based on the input it receives.

3.2 Data Collection and Preparation

The Alpaca dataset from Stanford University was used as the general domain dataset for fine-tuning the model. This dataset contains 52K instruction data for diverse tasks, generated by text-davinci-003 from 175 manually crafted seed tasks.

3.3 Fine-Tuning

Fine-tuning was conducted on the Intel Developer Cloud (IDC), utilizing the high-performance capabilities of Intel's 4th Generation Xeon Scalable processors. The fine-tuning process involved several steps:

- Setting up the environment and installing necessary libraries.
- Loading and preprocessing the Alpaca dataset.
- Configuring the training parameters.
- Running the fine-tuning process to adapt the LLaMA-2-7B model to the specific requirements of customer service tasks.

3.4 Inference and Evaluation

Post fine-tuning, the model was evaluated using a set of predefined queries to test its performance and accuracy. The inference process was also carried out on IDC, leveraging Intel's optimized infrastructure for efficient execution.

4. Implementation

4.1 Environment Setup

The development environment was set up using Google Cloud AI Notebooks and Kaggle for their support of high-performance hardware requirements. The following libraries and tools were used:

- Transformers library by Hugging Face for model handling.
- Intel Extension for Transformers for performance optimization.
- Flask for building the web application interface.
- Docker for containerization and deployment.

4.2 Training Script

The training script involved defining the model, tokenizer, and training arguments, followed by the actual fine-tuning process. Below is a simplified version of the training script used:

```
from transformers import TrainingArguments, AutoModelForCausalLM,
AutoTokenizer

from intel_extension_for_transformers.neural_chat import
finetune_model



tokenizer =
AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")

model =
AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf")
```

```
training_args = TrainingArguments(

    output_dir='./finetuned_model',

    do_train=True,

    do_eval=True,

    num_train_epochs=3,

    per_device_train_batch_size=4,

    per_device_eval_batch_size=4,

    logging_dir='./logs',

    logging_steps=500,

)


finetune_model(model, tokenizer, './alpaca_data.json',
training_args)
```

4.3 Web Application

A Flask web application was created to provide a user-friendly interface for interacting with the chatbot. The application includes HTML templates, CSS for styling, and JavaScript for handling user inputs.

app.py:

```
from flask import Flask, render_template, request, jsonify

from transformers import AutoModelForCausalLM, AutoTokenizer

import torch


tokenizer = AutoTokenizer.from_pretrained("./finetuned_model")

model = AutoModelForCausalLM.from_pretrained("./finetuned_model")
```

```
app = Flask(__name__)


@app.route("/")

def index():

    return render_template('chat.html')


@app.route("/get", methods=["GET", "POST"])

def chat():

    msg = request.form["msg"]

    input = msg

    return get_chat_response(input)


def get_chat_response(text):

    new_user_input_ids = tokenizer.encode(str(text) +
tokenizer.eos_token, return_tensors='pt')

    bot_input_ids = new_user_input_ids

    chat_history_ids = model.generate(bot_input_ids,
max_length=1000, pad_token_id=tokenizer.eos_token_id)

    return tokenizer.decode(chat_history_ids[:,
bot_input_ids.shape[-1]:][0], skip_special_tokens=True)


if __name__ == '__main__':

    app.run()
```

chat.html:

```html
<!DOCTYPE html>

<html>

<head>

    <title>Chatbot</title>

    <link rel="stylesheet" type="text/css" href="{{
url_for('static', filename='style.css') }}">

</head>

<body>

    <div class="chat-container">

        <div class="chat-box">

            <div class="messages" id="messages"></div>

            <form id="chat-form">

                <input type="text" id="text" name="msg"
placeholder="Type your message...">

                <button type="submit">Send</button>

            </form>

        </div>

    </div>

    <script src="{{ url_for('static', filename='script.js')
}}"></script>

</body>

</html>
```

Style.css:

```css
body, html {

    height: 100%;
    margin: 0;

    background: linear-gradient(to right, #263340, #323741,
#21214e);

    color: white;

}


.chat-container {

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100%;

}


.chat-box {

    background-color: rgba(0, 0, 0, 0.4);

    border-radius: 15px;

    width: 60%;

    max-width: 800px;

    padding: 20px;

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.5);

}


.messages {

    height: 400px;

    overflow-y: auto;
    margin-bottom: 20px;

}


#chat-form {

    display: flex;

}


#text {

    flex: 1;

    padding: 10px;

    border-radius: 5px;

    border: none;

}


button {

    padding: 10px;

    border-radius: 5px;

    border: none;

    background-color: #5865f2;

    color: white;

    cursor: pointer;

}
```

5. Challenges Faced

5.1 Understanding Methodologies

One of the initial challenges was comprehending the methodologies involved in fine-tuning LLMs. The complexity of the process required a thorough understanding of the underlying principles and the steps involved.

5.2 Hardware Limitations

Performing fine-tuning on a large model like LLaMA-2-7B required significant computational resources. Initial attempts on a personal laptop led to frequent crashes, necessitating a switch to more robust environments like Google Cloud AI Notebooks and Kaggle.

5.3 Integration and Deployment

Integrating the fine-tuned model into a web application and ensuring smooth deployment presented another set of challenges. Ensuring that the model performed efficiently within the Flask application required careful optimization.

6. Results and Evaluation

The fine-tuned model was evaluated using a set of predefined queries to test its performance and accuracy. The chatbot demonstrated a high level of understanding and relevance in its responses, showcasing the effectiveness of the fine-tuning process.

7. Conclusion

The project successfully demonstrated the process of fine-tuning a large language model to create a custom chatbot for customer service applications. The challenges encountered during the project provided valuable learning experiences, and the final chatbot showcased the potential of AI in enhancing customer interactions.

8. Future Work

Future enhancements could include further fine-tuning with more domain-specific data, implementing additional features like voice recognition, and deploying the chatbot on various platforms to expand its accessibility.

9. References

- Intel AI Tools: Intel AI Analytics Toolkit
- Alpaca Dataset: Stanford University