***Customer Support Chatbot***

|  |  |
| --- | --- |
| Yaqoot Alshdaifat | 0227779 |
| Leen Mushtaha | 0222030 |

***Cloud Computing***

***Prof. Mu’nis Qasaymeh***

**Abstract**

This project focuses on building an initial system to analyze user chat messages through two main stages. In the first stage, PySpark was used to load and preprocess synthetic chat data in JSON format—extracting the key fields: text, intent, and sentiment—and the data was cleaned and analyzed for frequent intent and sentiment patterns. In the second stage, a deep learning model using TensorFlow and LSTM was built and trained to classify each message into intent and sentiment categories: the data was tokenized, encoded, and split into training and testing sets, and results show the model can reasonably predict both intent and sentiment. Additionally, Firebase was used to store real-time chats between the user and the bot, laying the groundwork for future extensions such as response generation and deeper cloud integration.

Table of Contents

[Introduction 4](#_Toc199176593)

[**Overview** 4](#_Toc199176594)

[**Objectives** 4](#_Toc199176595)

[Background 4](#_Toc199176596)

[**Key Concepts** 4](#_Toc199176597)

[**Relevance to the Cloud-Native Approach** 5](#_Toc199176598)

[Methodology 5](#_Toc199176599)

[**Step-by-Step Approach** 5](#_Toc199176600)

[**Tools, Libraries and Cloud Services Used** 9](#_Toc199176601)

[Results and Discussion 9](#_Toc199176602)

[**Visualizations, Outputs, and Analysis** 9](#_Toc199176603)

[**Challenges and How They Were Addressed** 11](#_Toc199176604)

[Conclusion 11](#_Toc199176605)

[**Summary** 11](#_Toc199176606)

[**Suggestions for Future Work** 11](#_Toc199176607)

[References 12](#_Toc199176608)

# Introduction

## **Overview**

In today’s fast-paced digital marketplace, consumers expect immediate, accurate responses to their inquiries around the clock. Traditional support models—with phone queues, email tickets, and limited business-hour staff often struggle to keep up with rising volumes and evolving customer expectations. As a result, long wait times, inconsistent service quality, and ballooning support costs can erode brand loyalty and damage customer satisfaction. And this is where we deploy AI to help with minor tasks, thus increasing customer satisfaction.

## **Objectives**

Preprocess and clean chat data using PySpark

For distributed data processing and analysis.

Accurately classify user intents and sentiments

Achieve at least 85 % accuracy on intent detection and 80 % on sentiment analysis by leveraging the LSTM model and PySpark preprocessing pipeline.

Ensure end-to-end data persistence and security

Integrate with Cloud Firestore to log every user–bot exchange and enforce IAM rules so that only authorized services can read or write chat histories.

Build a multi-output deep learning model

Develop a unified deep-learning model that simultaneously predicts both user intent and sentiment, targeting at least 85 % accuracy on intent classification and 80 % on sentiment analysis.

# Background

## **Key Concepts**

***PySpark:***

PySpark is a Python-based interface for Apache Spark, used for distributed data processing and analysis. In this project, PySpark was utilized to load synthetic chat data in JSON format, clean the text by removing unnecessary symbols, and extract key fields such as text, intent, and sentiment. It was also used to analyze the frequency of different intents and sentiments and categorize messages based on their characteristics.

***TensorFlow:***

TensorFlow is an open-source machine learning library developed by Google. It was used in this project to build and train a deep learning model based on LSTM (Long Short-Term Memory) networks. The model was designed with multiple outputs to classify both the user’s intent and sentiment simultaneously. The training process involved preparing the text data through tokenization and encoding.

***Firebase:***

Firebase is a Backend-as-a-Service (BaaS) platform offered by Google Cloud that provides cloud-hosted services over the internet. Cloud Firestore, a database service, was used in this project to store real-time chats between users and the bot.

## **Relevance to the Cloud-Native Approach**

This project follows a cloud-native approach by utilizing tools that are scalable, distributed, and easy to integrate. PySpark allows efficient data processing across large datasets, TensorFlow supports deep learning model training on a broad scale, and Firebase stores data on Google’s cloud rather than on a local server or desktop.

# Methodology

This project was implemented in a cloud-based environment using Google Colab, with the goal of classifying user intent and sentiment from chat messages. The overall work consists of four main phases:

1. **Data preprocessing and cleaning using PySpark.**
2. **Building and training a multi-output classification model using Keras and TensorFlow.**
3. Developing both rule-based and model-based functions to enable the bot to respond intelligently and efficiently.
4. Integrating the bot with Cloud Firestore to persistently store every user–bot interaction.

## **Step-by-Step Approach**

**About the Dataset:**

The dataset was taken from Kaggle by the name of “**Consumer Reviews of Amazon Products**”. It’s originally a csv file with 0000 records.

With the help of AI, we were able to take a sample of 300 records, arrange them in JSON format, and categorize them into 6 intents (order\_status, return\_request, product\_issue, cancel\_order, greeting, thank\_you) and 3 sentiments (positive, neutral, negative).

**Loading the Data:**

PySpark library was used to load a JSON file containing user chat logs:

df = spark.read.option("multiline","true").json("synthetic\_chat\_logs\_with\_sentiment.json")

**Data Cleaning & Preprocessing:**

* Dropped null values using: dropna().
* Filtered short texts using: df = df.filter(length(col("text")) > 5)
* Removed special characters: df = df.withColumn("clean\_text",regexp\_replace("text", "[^a-zA-Z0-9\\s]", ""))
* Converted text to lowercase: df = df.withColumn("clean\_text",lower(col("clean\_text")))
* Counted the number of words in each message: df = df.withColumn("text\_length", size(split("clean\_text", " ")))

**Data Exploration:**

* Analyzed distribution by intent and sentiment using : groupBy().count().orderBy(desc("count")).
* Collected associated sentiments per intent: df.groupBy("intent").agg(collect\_set("sentiment"))
* Visualized the results using charts via matplotlib.

**Data Preparation for Model:**

* Converted text to sequences using **Keras Tokenizer**.
* Used **pad\_sequences** to standardize the lengths.
* Encoded intent and sentiment using **LabelEncoder**.

**Train-Test Split:**

The dataset was divided into training and testing sets, with 80% used for training and 20% used for testing: X\_train, X\_test, y\_intent\_train, y\_intent\_test, y\_sentiment\_train, y\_sentiment\_test = train\_test\_split(padded, intent\_labels, sentiment\_labels, test\_size=0.2, random\_state=42).

**Model Building:**

Built a **Keras model** using:

* Embedding layer
* Bidirectional LSTM layer
* Two Dense output layers: one for intent, one for sentiment

Used joint compile configuration:

* **Optimizer:** 'adam'.
* **Loss functions:** 'sparse\_categorical\_crossentropy' for both intent and sentiment outputs.
* **Metrics:** 'accuracy' for both outputs.

**Model Training:**

Trained the model for 10 epochs, achieving gradual improvement in accuracy.

history = model.fit(

    X\_train,

    [y\_intent\_train, y\_sentiment\_train],

    validation\_split=0.2,

    epochs=10,

batch\_size=32)

**Model Evaluation:**

Evaluated the model on the test set, and the results were:

* **Intent Accuracy:** 0.95
* **Sentiment Accuracy:** 0.97

**Developed Functions:**

classify(message) function, which:

* Converts the message to a numeric sequence.
* Passes it through the model.
* Returns the original label using inverse\_transform.

generate\_pattern\_response(user\_input) function, which:

* Contains the most repeated inquiries of users
* The function matches the mentioned pattern with no regard to spaces or letter cases because of strip() function
* Makes the machine reply with a constant response
* If no pattern matches the user input, then the user input is forwarded to the model

generate\_rule\_based\_response\_chatbot(user\_input) function, which:

* Extracts the sentiment and intent from classify(message) function
* Sends the predicted sentiment and intent to generate\_rule\_based\_response(predicted\_sentiment, predicted\_intent)
* function

generate\_rule\_based\_response(predicted\_sentiment, predicted\_intent) function, which:

* Gives a default response in case of a mismatch
* Gives specific responses in case of matching between sentiment and intent

store\_message(user\_id, sender, message) function, which:

* Helps Firestore to locate the messages stored and their structure:
* Points to the user collection
* Creates a document within the user collection whose ID is given
* Uses a subcollection inside the user document called chat\_history
* Creates a document within chat\_history with a random UUID
* Stores the message with 3 main features (sender, message, timestamp)

**Linking to Firestore:**

We loaded our service-account JSON key so the server code can authenticate with Firebase. Then we booted up Admin SDK only once. Admin SDK helps in authenticating the process, configuring default clients, and registering an app instance.

if not firebase\_admin.\_apps:

firebase\_admin.initialize\_app(cred)

print("Firebase app initialized.")

else:

print("Firebase app already initialized.")

After initialization, firestore.client() gave us the db object used in store\_message() to read and write documents.

## **Tools, Libraries and Cloud Services Used**

|  |  |
| --- | --- |
| Tool / Library | Purpose |
| PySpark | Used for loading, cleaning, and filtering structured text data from a JSON file. |
| TensorFlow / Keras | Used to build and train the deep learning model with multiple outputs (intent + sentiment). |
| Pandas | Used to convert and prepare structured data for input into the model. |
| Matplotlib | Used to visualize intent and sentiment distributions during data analysis. |
| UUID | Useful for creating non-guessable, universally unique identifiers, IDs for objects, sessions, filenames, etc. |
| DateTime | Gets the current time, parse/format timestamps, compute time differences, etc. |
| Firebase\_admin | Used to authenticate and manage users and read/write to real-time Firestore. |

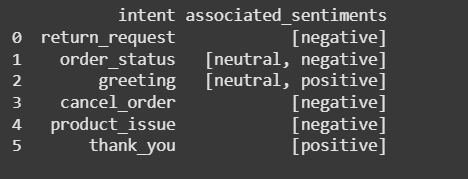
|  |  |
| --- | --- |
| Service | Purpose |
| **Google Colab** | Full project execution using an interactive cloud-based development environment that supports compute resources like GPU, with integration and storage through Google Drive. |
| **Firestore** | A serverless NoSQL document database that is used to store all entry logs between the user and the bot. The database is cleanly organized by user-ids, collections and documents. |

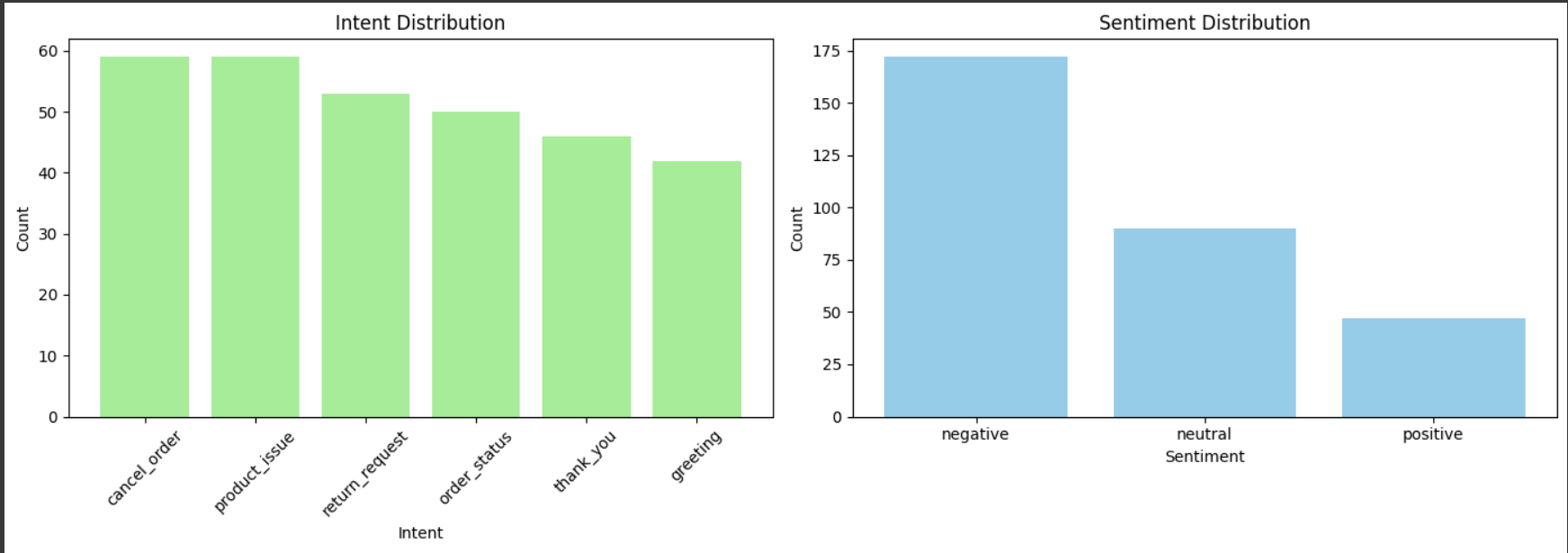
# **Results and Discussion**

## **Visualizations, Outputs, and Analysis**

**Intent Distribution:**  
Using groupBy().count() and bar plots via matplotlib, the distribution of user intents was visualized. The most common intents were order\_status, greeting, and product\_issue, reflecting the support-oriented nature of the conversations.

**Sentiment Distribution:**  
Visualizations showed that **neutral** and **negative** sentiments were dominant, while **positive** messages were less frequent. This aligns with real-world support chat scenarios, where users often express complaints or concerns.

**Associated Sentiments per Intent:**  
Using collect\_set("sentiment"), the sentiments linked to each intent were displayed in a table. The results showed that a single intent could appear with different emotional tones. For example, order\_status was associated with both neutral and negative messages depending on the user’s experience or mood.



**Model Accuracy:**  
After training the model for 10 epochs:

* + **Intent classification accuracy** reached approximately **95%**
  + **Sentiment classification accuracy** reached around **97%**
  + These results demonstrate the model’s strong ability to generalize to unseen data.

**Prediction Testing:**  
The model was tested on a set of real-world messages using the classify() function. In most cases, the predictions for both intent and sentiment were accurate, showing the model’s effectiveness.

## **Challenges and How They Were Addressed**

**Imbalanced Data:**  
Some intents and sentiments appeared more frequently than others, which could bias the model. This was addressed by:

* Removing very short or vague messages
* Adding more examples to underrepresented classes

**Text Noise and Inconsistency:**  
The raw text contained special characters and inconsistent formatting. This was resolved using PySpark functions like regexp\_replace and lower() to clean and normalize the data.

# Conclusion

## **Summary**

This project successfully completed the first two phases of a cloud-based chatbot system designed to classify **user intent** and **sentiment** from text messages. **PySpark** was used to clean and preprocess the data efficiently by removing noise and unifying input formats. Valuable insights were extracted through exploratory analysis, such as the most common intents and sentiment patterns.

A deep learning model was then built using **Keras** and **TensorFlow** with a multi-output architecture that predicts both intent and sentiment simultaneously. The model achieved high performance, with approximately **95% accuracy** in intent classification and **97% in sentiment.**

Multiple functions were developed to guide the bot throughout the response process. There functions were a mix between rule-based and model-based learning to enhance its efficiency.

Finally, the bot was linked with the cloud service, Cloud Firestore database to persistently store user and chatbot interactions by user-id.

## **Suggestions for Future Work**

* **To improve intent and sentiment accuracy using transformer models**
* **To create a more functional API for third-party users**
* **To increase the efficiency of the chatbot by supporting multiple languages**

# References

1. PySpark API Reference:  
   <https://spark.apache.org/docs/latest/api/python/>
2. TensorFlow & Keras Official Documentation:  
   <https://www.tensorflow.org/><https://keras.io/>
3. Python Pickle Module – Object Serialization  
   <https://docs.python.org/3/library/pickle.html>
4. Dataset: <https://www.kaggle.com/datasets/datafiniti/consumer-reviews-of-amazon-products?resource=download>
5. Firebase <https://console.firebase.google.com/u/0/>