Phase-2 Submission Template

Student Name: Mohammed Ragid V

Register Number: 510623104062

Institution: C.ABDUL HAKEEM COLLEGE OF ENGG & TECH

Department: [COMPUTER SCIENCE AND ENGINEERING]

Date of Submission: [09/05/2025]

Github Repository Link:
https://github.com/MohammedRagid/Revolutionizing-customer-support-with-an-intelligent-chatbot-for-automated-assistance.git

---

1. Problem Statement

''Revolutionizing Customer Support with an Intelligent Chatbot for Seamless Automated Assistance"

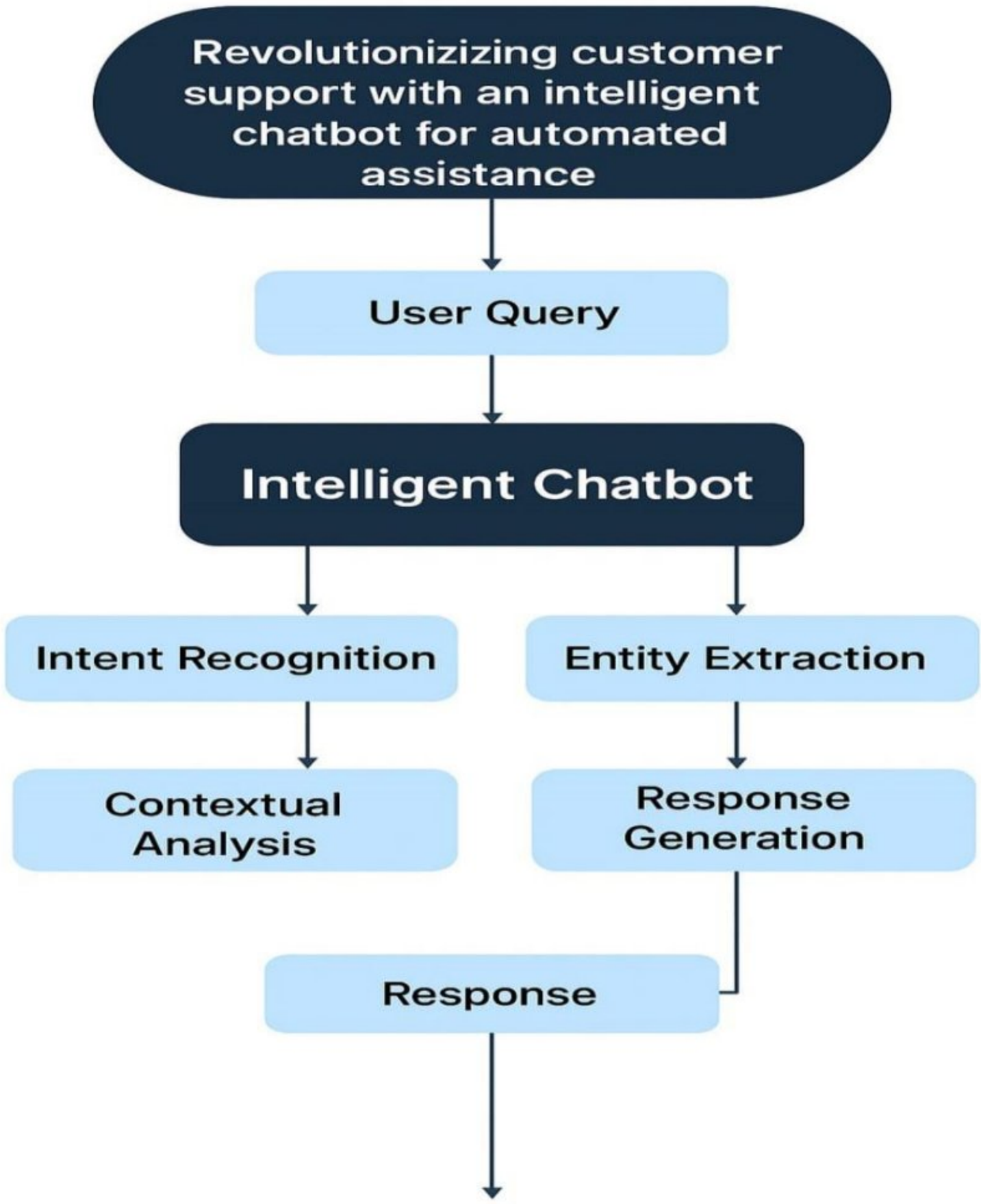
Traditional customer support systems often struggle with high response times, inconsistent service quality, and limited availability, leading to poor customer experiences. There is a need for an intelligent, automated solution that can provide accurate, real-time assistance, reduce support costs, and scale efficiently to meet increasing customer demands.

2. Project Objectives

1. **Develop an AI-powered chatbot** capable of handling common customer queries across multiple channels (e.g., website, mobile app, social media).
2. **Reduce average response and resolution times** by automating routine inquiries and providing instant, 24/7 support.

3. **Enhance customer satisfaction** by delivering consistent, accurate, and personalized responses.
4. **Integrate the chatbot** with existing customer relationship management (CRM) systems to access relevant customer data and improve support quality.
5. **Continuously improve chatbot performance** using machine learning, customer feedback, and data analytics.
6. **Minimize operational costs** by reducing reliance on human agents for repetitive tasks

3. Flowchart of the Project Workflow



4. Data Description

1. Interaction Logs: *Historical chat logs, email transcripts, and call center notes that contain real conversations between customers and support agents. These will help the chatbot understand typical queries, language patterns, and effective responses.*

Frequently Asked Questions (FAQs): A structured list of common customer questions and their standard answers, which can be used to create an initial knowledge base.

3. **Product/Service Information:** *Detailed information about the company's offerings, including specifications, pricing, return policies, and troubleshooting guides, to ensure the chatbot provides accurate and relevant answers.*
4. **Customer Profile Data:***(if applicable and with proper privacy considerations): Includes customer preferences, purchase history, and previous issues to help the chatbot personalize responses.*
5. **Feedback and Ratings:** *Customer satisfaction ratings and feedback on previous chatbot or human support interactions, useful for improving the chatbot's accuracy and tone.*
6. **Intent and Entity Data:** *Structured data for training natural language understanding (NLU) models, mapping user inputs to specific intents (e.g., "track my order") and extracting key entities (e.g., order number).*

5. Data Preprocessing

To enable the chatbot to understand and respond effectively to customer inquiries, raw data must undergo several processing steps:

1. **Data Collection**: *Gather data from various sources such as customer support logs, FAQs, emails, product manuals, and CRM systems.*
2. **Data Cleaning**: *Remove irrelevant or sensitive information, correct grammatical errors, standardize formatting, and eliminate duplicates to ensure high-quality training data.*
3. **Text Preprocessing**: *Apply natural language processing (NLP) techniques such as:*
    - *Tokenization (splitting text into words or phrases)*
    - *Stop-word removal (filtering out common, non-informative words)*
    - *Lemmatization/Stemming (reducing words to their root form)*

- *Named Entity Recognition (to identify important entities like product names, locations, etc.)*

4. **Intent Classification and Entity** : *ExtractionLabel and structure the data to train models that can recognize user intents (e.g., "check order status") and extract relevant entities (e.g., order ID).*
5. **Training Data Annotation**: *Manually or semi-automatically annotate sample conversations to improve the chatbot's machine learning models.*
6. **Model Training and Evaluation**: *Use the processed and labeled data to train the chatbot's NLP and machine learning models. Evaluate performance using metrics such as accuracy, precision, recall, and F1-score.*
7. **Knowledge Base Integration**: *Convert structured FAQ and product information into a format suitable for dynamic retrieval by the chatbot (e.g., using vector embeddings or search indexing).*
8. **Continuous Feedback Loop**: *Collect user interactions with the chatbot post-deployment to retrain and fine-tune the model, improving accuracy over time.*

6. Exploratory Data Analysis (EDA)

1. Data Overview

Start by loading the dataset and understanding its structure:

```
import pandas as pd
df = pd.read_csv('chat_data.csv')
df.head()
df.info()
```

Typical columns might include:

user_message

intent

entities

response

timestamp

user_id

channel (web, app, WhatsApp, etc.)

2. Intent Distribution

Check how balanced the dataset is:

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
sns.countplot(y='intent', data=df, order=df['intent'].value_counts().index)
plt.title("Intent Distribution")
plt.show()
```

Insights: Identify underrepresented intents. You may need data augmentation.

3. Message Length Analysis

```
df['message_length'] = df['user_message'].apply(lambda x: len(str(x).split()))

sns.histplot(df['message_length'], bins=30)
plt.title("Distribution of User Message Lengths")
plt.show()
```

Detect outliers or overly short/long messages.

Useful for padding/sequence truncation.

4. Top Keywords per Intent

Use TF-IDF to identify frequent terms:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
tfidf = TfidfVectorizer(max_features=1000)
X_tfidf = tfidf.fit_transform(df['user_message'])
```

5. Named Entity Recognition (NER) Presence

Check if entities are well-tagged:

```
df['has_entities'] = df['entities'].apply(lambda x: 0 if pd.isnull(x) or x == '[]' else
1)
df['has_entities'].value_counts().plot.pie(autopct='%1.0f%%', labels=['No Entity',
'Has Entity'])
plt.title("Entity Presence in Messages")
plt.show()
```

6. Sentiment Analysis (Optional)

```
from textblob import TextBlob
```

```
df['sentiment'] = df['user_message'].apply(lambda x:
TextBlob(str(x)).sentiment.polarity)
sns.histplot(df['sentiment'], bins=20)
plt.title("User Sentiment Distribution")
plt.show()
```

7. Channel or Time-Based Patterns

```
df['hour'] = pd.to_datetime(df['timestamp']).dt.hour

sns.countplot(x='hour', data=df)
plt.title("Query Volume by Hour")
```

8. Missing/Null Values

```
df.isnull().sum()
```

Especially important for intent and response columns.

9. Word Cloud (Visual Insight)

```
from wordcloud import WordCloud
text = ' '.join(df['user_message'].dropna().tolist())
wordcloud = WordCloud(background_color='white').generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of User Messages")
plt.show()
```

7. Feature Engineering

1. Textual Input Features

Tokenization: Breaking user queries into words, subwords, or tokens.

TF-IDF / Word Embeddings: Using pre-trained embeddings (e.g., BERT, GloVe) to represent text meaning.

Intent Classification Tags: Labels indicating the purpose of the query (e.g., "order status", "refund request").

Named Entity Recognition (NER): Extracting customer names, product names, order IDs, dates, etc.

2. Contextual Features

Session Context: Information from earlier messages in the conversation.

Customer Profile Data: Past interactions, account type, preferences.
Device/Platform Info: Mobile, desktop, web app, etc.
Geolocation: Country or region-based assistance handling.

3. Sentiment & Emotion Features

Sentiment Score: Positive, neutral, or negative user tone.
Emotion Detection: Anger, frustration, urgency (triggers escalation or calming tone).

4.Behavioral Features

Response Time: Time taken by user to respond (could indicate confusion or urgency).
Clickstream Data: Pages or options the user has interacted with before reaching support.

Navigation History: Path taken before contacting support.

5. System & Response-Level Features

Confidence Scores: From the intent classifier to determine if human escalation is needed.
Fallback Frequency: How often the chatbot couldn't answer and had to use default replies.
Response Length/Depth: Can vary based on user type (e.g., new vs. returning customers).

6. Personalization Features

Language Preferences: Multi-language support adaptation.
Time Zone Awareness: For context-appropriate replies.
Interaction History: Previous issues and resolutions

8. Model Building

1. Data Collection

Sources: Historical chat logs, support tickets, FAQs, emails.
Labeling: Label each interaction with intents (e.g., "reset password", "track order") and entities (e.g., "order ID", "product name").

2. Data Preprocessing

Text cleaning (remove punctuation, lowercasing, remove stop words if not using transformers).
Tokenization.

NER tagging (using spaCy, Hugging Face, or custom rules).
Intent labels encoding.

3. Feature Engineering

(From previous step)
Convert text to embeddings: BERT, DistilBERT, or sentence transformers.
Extract contextual/session/user features.
Include sentiment/emotion scores.
Create padded sequences for model input.

4. Model Architecture

Choose one or a combination:
A. Intent Classification Model
Input: User message + context.
Model:
BERT → Dense layers → Softmax (multi-class intent)
Alternative: LSTM/GRU with attention
B. Entity Recognition Model
Input: Tokenized message.
Model:
BERT / BiLSTM + CRF for sequence labeling (NER)
C. Dialogue Management / Response Selection
Rule-based or ML-based (e.g., Rasa's TED policy, or retrieval-based transformer).
Or use a generative model (e.g., fine-tuned GPT or T5) for response generation.

5. Training

Split into train/test/validation.
Use cross-entropy loss for intent classification.
Categorical accuracy/F1-score as evaluation metrics.
Fine-tune on domain-specific data.

6. Evaluation

Accuracy/F1 for intent prediction.
Precision/Recall for NER.
Confusion matrix to analyze misclassified intents.
User feedback loop integration for continuous improvement.

7. Deployment

Containerize with Docker.

Expose via REST API or integrate into front-end using WebSockets.
Use a load balancer and monitoring tools.
Log interactions for retraining.
8. Optional: Add-on Intelligence
Sentiment-aware response modulation
Auto-escalation trigger to human agent based on confidence/sentiment
Multilingual support with translation APIs or multilingual models

9. Visualization of Results & Model insights

1. Intent Classification Results

A. Confusion Matrix

Shows how well the model predicted each intent.

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_true, y_pred, labels=model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=model.classes_)
disp.plot(xticks_rotation='vertical', cmap='Blues')
plt.title("Confusion Matrix of Intent Predictions")
plt.show()
```

B. Classification Report

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

Helps identify low precision or recall intents.

2. Entity Recognition (NER) Evaluation

A. Entity Type Distribution

```
import seaborn as sns
entity_df = df.explode('entities')  # If entities is a list
sns.countplot(y='entity_type', data=entity_df)
plt.title("Entity Type Distribution")
```

B. Entity-Level F1 Score

from seqeval.metrics import classification_report as seq_classification_report
print(seq_classification_report(true_entity_labels, pred_entity_labels))

3. Confidence Score Distribution

Shows how confident the model is across predictions.
sns.histplot(model_confidences, bins=30, kde=True)
plt.title("Distribution of Model Confidence Scores")
plt.xlabel("Confidence Score")

Useful to set a threshold for escalating low-confidence cases to human agents.

4. Word Importance (Model Explainability)

Using SHAP for Interpretability
import shap
explainer = shap.Explainer(model, tokenizer)
shap_values = explainer(user_input_texts[:100])
shap.plots.text(shap_values[0])

See which words contributed most to intent classification.

5. Response Accuracy & Relevance (if applicable)

Manual Rating Visualization
If you gather user feedback:
sns.countplot(x='response_rating', data=feedback_df)
plt.title("User Ratings for Bot Responses")
Embedding Space Visualization
Visualize intents using dimensionality reduction:
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
X_embedded = tsne.fit_transform(embeddings)
plt.scatter(X_embedded[:, 0], X_embedded[:, 1], c=labels, cmap='tab10')
plt.title("Intent Embedding Space (t-SNE)")
plt.show()

6. Real-time Dashboard (Optional)

You can build a dashboard using:
Plotly Dash
Streamlit
Power BI or Tableau (connected via API)

10. Tools and Technologies Used

1. Data Collection & Storage

MySQL / PostgreSQL – for storing structured user data

MongoDB / Firebase – for storing unstructured conversation logs

Google Sheets / Excel – initial dataset curation

Web scraping / APIs – to gather FAQ, support articles

2. Data Processing & EDA

Python – primary language

Pandas / NumPy – data manipulation

Matplotlib / Seaborn / Plotly – visualizations

NLTK / spaCy / TextBlob – basic NLP preprocessing and sentiment analysis

3. NLP & Model Building

Intent Classification & NER

Scikit-learn – basic ML models like SVM, Naive Bayes

TensorFlow / Keras / PyTorch – deep learning models (LSTM, BERT)

Hugging Face Transformers – BERT, DistilBERT, RoBERTa for fine-tuning

spaCy / Flair – for NER and custom pipeline support

Dialog Management & Flow

Rasa – open-source NLP and dialogue framework

Dialogflow / Microsoft Bot Framework / IBM Watson Assistant – cloud-based platforms

OpenAI GPT / LLaMA / Claude – for generative chatbot capabilities

4. Model Evaluation & Explainability

Scikit-learn metrics – confusion matrix, F1, precision, recall

seqeval – for sequence labeling metrics (NER)

SHAP / LIME – model interpretability and feature impact

5. Deployment

Flask / FastAPI – API endpoint for chatbot model

Docker – containerization

NGINX / Gunicorn – production web server

AWS / Azure / Google Cloud – cloud deployment

Kubernetes – scalable orchestration (optional)

6. User Interface & Integration

React / Angular / Vue – chatbot front-end

Botpress / Microsoft Bot Framework Web Chat – plug-and-play UI

Twilio / WhatsApp API / Facebook Messenger API – for multichannel support

Slack / Teams integration – for internal support bots

7. Monitoring & Feedback

ELK Stack (Elasticsearch, Logstash, Kibana) – for logging and analytics

Prometheus + Grafana – for performance monitoring

Mixpanel / Google Analytics – for user behavior tracking

11. Team Members and Contributions

- *Data cleaning* –**Raj kumar**
- *EDA* – **Mohammad Ragid**
- *Feature engineering* – **Mohammad khursheed ahmed**
- *Model development* **-Rhithish**
- *Documentation and reporting*- **Samnesh & Syed Musthafa**