AI Course

Capstone Project   
Final Report

For students (instructor review required)

ⓒ2023 SAMSUNG. All rights reserved.

Samsung Electronics Corporate Citizenship Office holds the copyright of this document.

This document is a literary property protected by copyright law so reprint and reproduction without permission are prohibited.

To use this document other than the curriculum of Samsung Innovation Campus, you must receive written consent from copyright holder.

|  |
| --- |
| Chatbot for Sales Consulting |

Date (29/07/2024)

Group1-HCMOU AI01

Vuong Minh Khanh

Hua Quang Dat

Tran Thi Thuy Dung

Phung Trung Hieu

Content

1. Introduction

1.1. Background Information

1.2. Motivation and Objective

1.3. Members and Role Assignments

1.4. Schedule and Milestones

2. Project Execution

2.1. Data Acquisition

2.2. Training Methodology

2.3. System Diagram

3. Results

3.1. Data Preprocessing

3.2. Exploratory Data Analysis (EDA)

3.3. User Interface

3.4. Testing and Improvements

4. Projected Impact

4.1. Accomplishments and Benefits

4.2. Future Improvements

5. Team Member Review and Comment

6. Instructor Review and Comment

1. Introduction

**1.1. Background Information**

* In today's digital age, chatbots have become an important tool in improving customer service and automating communication processes. Thanks to the development of artificial intelligence (AI) and natural language processing (NLP), it has given the chatbot a better chance to learn, thereby improving the speed and accuracy of communication with customers. Modern chatbots can operate on multiple platforms, including websites, mobile applications, and social messaging platforms such as Facebook Messenger, WhatsApp, etc. In particular, sales is one of the fields where chatbots are most widely used.
* The history of sales chatbots can be traced back to the 2010s when messaging platforms like Facebook Messenger started supporting chatbots. The popularity of sales chatbots has increased as businesses have realized their ability to improve conversion rates and enhance the effectiveness of marketing campaigns. Not only do chatbots help reduce the workload of sales teams, but they also provide 24/7 customer service, making it convenient for customers to make purchases at any time.
* The integration of sales chatbots with CRM (customer relationship management) systems and analytics tools also allows businesses to track and analyze customer shopping behavior, thereby optimizing sales and marketing strategies. With the ability to automate sales processes and enhance user experience, sales chatbots are increasingly becoming an essential part of modern business strategies.

**1.2. Motivation and Objective**

* In an increasingly competitive business environment, businesses are looking for Unique Selling Propositions (USP) to increase sales efficiency and improve customer experience. Companies are also facing a huge challenge in managing high volumes of customer interactions, while also needing to optimize their sales processes to increase revenue. Adopting sales chatbots has become a powerful solution to meet this need. Therefore, chatbots, with their ability to operate continuously and handle multiple interactions simultaneously, provide an effective solution to address these issues.
* The immediate goal is to understand the background knowledge of artificial intelligence and advanced Large Language Models (LLM) with ChatGPT, as a basis for research and analysis of AI chatbot achievements and products. The focus is on research on interactivity and accuracy in providing information, followed by testing and application to optimize the effectiveness of language models in real-life environments.

**1.3. Members and Role Assignments**

* **Vuong Minh Khanh - Team Leader**:
* Collects chat conversation data and evaluates the chatbot's performance.
* Plans tasks, sets goals, and monitors team progress.
* Writes the project report.
* **Hua Quang Dat - Team Member**:
* Researches LangChain integrations and configures the model.
* Designs the chatbot interface.
* Contributes to the report writing.
* **Phung Trung Hieu - Team Member**:
* Handles data preprocessing.
* Tests the bot responses using chat conversations.
* Assists in report documentation.
* **Tran Thi Quynh Dung - Team Member**:
* Collects product data.
* Prepares PowerPoint presentations.

**1.4. Schedule and Milestones**

1. Schedule Overview

* June 11, 2024, to July 17, 2024:
* General Tasks: Team members will research, build chatbots, and collect necessary data.
* July 18, 2024, to July 28, 2024:
* General Tasks: Members will run chatbot tests and evaluate performance.
* Individual Tasks:
* Vuong Minh Khanh: Will evaluate the accuracy of conversations and support team members.
* Hua Quang Dat: Will find a method to evaluate the accuracy of ChatGPT other than manually.
* Phung Trung Hieu: Will complete data preprocessing and research applying preprocessing techniques to the ChatGPT bot.
* Tran Thi Quynh Dung: Will collect additional data to reach at least 1000 product data.
* July 29, 2024, to August 3, 2024:
* Vuong Minh Khanh: Will collect more chat data and check the accuracy of the bot.
* Hua Quang Dat: Will design the chatbot interface.
* Phung Trung Hieu: Will prepare report content.
* Tran Thi Quynh Dung: Will prepare presentation slides.
* August 4, 2024, to August 14, 2024:
* Vuong Minh Khanh: Will finish customizing the chatbot and start writing the report.
* Hua Quang Dat: Will write the report.
* Phung Trung Hieu: Will write the report.
* Tran Thi Quynh Dung: Will prepare presentation slides.

1. Milestones:

* **June 18**:
* Built a simple chatbot with customized data and tested its performance.

* **July 26**:
* Completed the product database.
* **July 31**:
* Upgraded the chatbot's code.
* **August 2**:
* Achieved a significant increase in accuracy.
* **August 9**:
* Customized the retriever and achieved a further increase in accuracy.

2. Project Execution

**2.1. Data Acquisition**

* **Tools Used:**
* **WebScraper.io:** A Chrome Extension for automatic data scraping, enabling easy data export.
* **Python beautifulsoup4 4.12.3:** A library for web scraping to collect additional data inaccessible via WebScraper.io.
* **Data Collection Details:**
* **Source Website:** <https://769audio.vn/> (focused solely on audio electronics)
* **Scope of Data:**
* Tên sản phẩm (Product name)
* Giá (Price)
* Link sản phẩm (Product link)
* Link ảnh (Thumbnail link)
* Tình trạng (Status)
* Giới thiệu (Introduction)
* Chi tiết (Detailed content)
* Tập link ảnh (Product image links)
* **Data Volume:** 1944 rows

2.2. Training Methodology

1. **Data Preprocessing:** The data preprocessing process is carried out to standardize and clean specific columns in the DataFrame. The steps include handling currency formats, normalizing item status, and improving the readability of text columns.

* Based on Google Colab, the first method of operation is to retrieve data from the file Data\_in\_769audio\_vn.csv. Read the data and remove unnecessary columns.
* **df = pd.read\_csv("/content/drive/My Drive/Data\_in\_769audio\_vn.csv", on\_bad\_lines='skip')**: Read data from the CSV file into the DataFrame df. The on\_bad\_lines='skip' parameter skips invalid lines.
* **df = df.drop(columns=["Công Suất", "Mô tả"], axis=1)**: Remove the unnecessary columns "Công Suất" and "Mô tả".

import pandas as pd

# column\_names = ["Tên","link sản phẩm","Công Suất","Giá","Link ảnh","Tình trang","Mô tả","Giới thiệu","Chi tiết","Tập link ảnh"]

df = pd.read\_csv("data\_path", on\_bad\_lines='skip')

* Preprocessing the Price Column:
* def *clean\_currency(price)*: Removes currency symbols and thousand separators from the "Giá" column. If the price is a string, it removes the character "đ" and periods, then converts the value to an integer. If the price is not a string or does not contain numbers, it returns NaN.
* Applies the *clean\_currency* function to the "Giá" column, then converts the data to the nullable integer type *Int64* to handle NaN values.

def clean\_currency(price):

if isinstance(price, str): # Check if the input is a string

# Remove the currency symbol and any trailing whitespace

cleaned\_str = price.replace('đ', '').strip()

# Remove dots used as thousand separators

cleaned\_str = cleaned\_str.replace('.', '')

# Convert to integer

return int(cleaned\_str)

elif np.isnan(price):

# Return NaN as is or consider filling it with a default value like -1 or 0

return np.nan

else:

# If it’s already a number, just return it as int

return int(price)

* The following line processes the "Tình trạng" column:
* *def convert\_status(status):*: Converts the values in the "Tình trạng" column to binary values: 1 for "còn hàng" (in stock) and 0 for "hết hàng" (out of stock) or any values not found. *None* is returned for other or undefined cases.
* Applies the *convert\_status* function to the "Tình trạng" (status) column.

def convert\_status(status):

if status is None:

return None

elif status.lower() == 'còn hàng':

return 1

elif status.lower() == 'hết hàng' or status == "Status label not found on the page.":

return 0

else:

return None

* Processing the "Giới thiệu" (Introduction) and "Chi tiết" (Detailed content) columns:
* **Normalize new lines:** Replace consecutive new line characters with a single new line and remove unnecessary whitespace.
* **Normalize white spaces:** Replace consecutive white spaces and tabs with a single white space.
* **Capitalize the first letter of each sentence:** Capitalize the first letter of each sentence in the paragraphs.
* **Ensure correct spacing after punctuation:** Remove extra spaces before punctuation and ensure proper spacing after punctuation.
* Creates a copy of the DataFrame to protect the original data.
* Applies the *format\_text* function to format the text in the "Giới thiệu" and "Chi tiết"column.

def format\_text(text):

# Normalize new lines, replacing multiple new lines with a single new line

text = re.sub(r'\n\s\*\n', '\n', text.strip())

# Normalize whitespace within lines

text = re.sub(r'[ \t]+', ' ', text)

# Capitalize the first letter of each sentence in each paragraph separately

paragraphs = text.split('\n')

formatted\_paragraphs = []

for paragraph in paragraphs:

sentences = re.split(r'(?<=[.!?]) +', paragraph.strip())

formatted\_sentences = [sentence.capitalize() for sentence in sentences]

formatted\_paragraph = '. '.join(formatted\_sentences)

formatted\_paragraphs.append(formatted\_paragraph)

# Rejoin paragraphs with a single newline

text = '\n'.join(formatted\_paragraphs)

# Ensure proper spacing after punctuations

text = re.sub(r'\s([,.])', r'\1', text)

text = re.sub(r'([,.])([^\s])', r'\1 \2', text)

return text

* **The following line saves the data to a CSV file:**

# Define the path where the CSV will be saved

csv\_file\_path = "data\_path"

# Save the DataFrame to a CSV file

new\_df.to\_csv(csv\_file\_path, index=False)

1. **Data Splitting:**

* After collecting user feedback and objectively evaluating the chatbot's accuracy, we have chosen a chunk size of 65. Initially, we need to calculate the average number of characters per sentence in the "Details" and "Introduction" columns, as these are the two columns containing the most text segments.

import re

import pandas as pd

def count\_avg\_words(df: DataFrame):

sentences = re.split(r'[.\n\n\n]', text)

average\_chars = sum(len(s) for s in sentences if s.strip()) / len(sentences)

return average\_chars

c=count\_avg\_words(df["Chi tiết"])

print(f'Trung bình số kí tự mỗi câu cột Chi tiết: {c}')

Trung bình số kí tự mỗi câu cột Chi tiết **67.33951225097321**

c=count\_avg\_words(df["Giới thiệu"])

print(f'Trung bình số kí tự mỗi câu cột Giới thiệu: {c}')

Trung bình số kí tự mỗi câu cột Giới thiệu **63.251697494731914**

* Based on the two results above, we have chosen a chunk size of 65 to ensure that the chunks do not lose meaning and maintain coherence within a sentence. After determining the chunk size, we will use RecursiveCharacterTextSplitter from the LangChain library to split the text into smaller segments for easier processing.
* RecursiveCharacterTextSplitter splits the text in a way that preserves better semantic structure and allows customization of delimiters. It supports both short and long segments, with overlap capabilities to retain important information at the boundaries of the segments.

loader = CSVLoader(file\_path="data\_path", encoding="utf-8", csv\_args={'delimiter': ','})

data = loader.load()

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=**65**, chunk\_overlap=2)

texts = text\_splitter.split\_documents(data)

1. **Storing Text as Numerical Vectors:**

* To perform data retrieval, we need to create a search index based on FAISS (Facebook AI Similarity Search). For this, we need embeddings models such as BERT, GPT, and in this case, we use *OpenAIEmbeddings()* from the *langchain\_openai* library to achieve the highest efficiency. The embeddings model converts text data into similarity vectors, allowing the model to understand and compare text in a way that computers can process.

from langchain\_openai import OpenAIEmbeddings

embeddings = OpenAIEmbeddings()

docsearch = FAISS.from\_documents(texts, embeddings)

* Reuse this vector store by downloading and loading it locally, eliminating the need for data splitting and vector storage:

docsearch = FAISS.load\_local(DB\_FAISS\_PATH, embeddings, allow\_dangerous\_deserialization=True)

1. **Retrieval:**

* After creating a FAISS vector store, we needed a user query to retrieve only the necessary information. LangChain provides three main search types for retrieval, based on the specific needs of our application. In this project, we mainly focused on two search types, which are:
* *Similarity:* This is a default search type, generally suitable for most applications requiring semantic understanding, and by default returning k = 4 most related documents.

retriever=docsearch.as\_retriever()

* *Similarity score threshold:* This search type sets a threshold for the similarity scores, enhancing output documents, and maintaining a high quality of results, which matches our project orientation.

retriever = docsearch.as\_retriever(

search\_type="similarity\_score\_threshold",

search\_kwargs={

"score\_threshold": 0.3,

"k": 15

}

)

* However, responses sometimes do not cover all product attributes, as keywords for that product mostly appear in the article's content, and the retriever only collects similar chunks to the query, leading to wrong responses (fake product links, incorrect price, etc). Hence, in our ChatBot’s third version, we summarized rows of chunks and collected data rows from those rows.

from langchain.docstore.document import Document

def retrieve\_and\_filter\_chunks(row\_numbers, data, excluded\_columns=["Giới thiệu", "Chi tiết"]):

filtered\_chunks = []

for row\_number in row\_numbers:

# Check if row number is valid before accessing

if row\_number in data.index:

row\_data = data.loc[row\_number]

for col in data.columns:

if col not in excluded\_columns:

filtered\_chunks.append(

Document(page\_content=str(row\_data[col]),

metadata={"source": col,

"row": row\_number}))

return filtered\_chunks

retriever = docsearch.as\_retriever(

search\_type="similarity\_score\_threshold",

search\_kwargs={

"score\_threshold": 0.3,

"k": 20

}

)

# Query

query = """

Dạ, Chào anh chị, em đang quan tâm Loa JBL Control 25, Gía bán bên mình loa này sao anh chị

"""

# Retrieve initial chunks

initial\_docs = retriever.invoke(query)

# Extract row numbers from initial retrieval

row\_numbers = {doc.metadata["row"] for doc in initial\_docs} # Using a set comprehension to ensure uniqueness

# Result as a list

row\_numbers = list(row\_numbers)

row\_numbers = [x - 9 for x in row\_numbers] # due to row index from LangChain CSVLoader is different from normal pd.read\_csv() by 9

print("Row numbers:", row\_numbers)

Row numbers: [**289, 1161, 1038, 1005, 653, 658**]

# Retrieve and filter chunks for each row number

filtered\_docs = retrieve\_and\_filter\_chunks(row\_numbers, data)

filtered\_docs.extend(initial\_docs)

for doc in filtered\_docs:

if doc.metadata["row"] == 667 or doc.metadata["row"] == 658:

print("Row:", doc.metadata["row"], "-", doc.page\_content)

**Row: 658** - Loa JBL Control 25

**Row: 658** -<https://769audio.vn/san-pham/312/loa-jbl-control-25-chinh-hang.html>

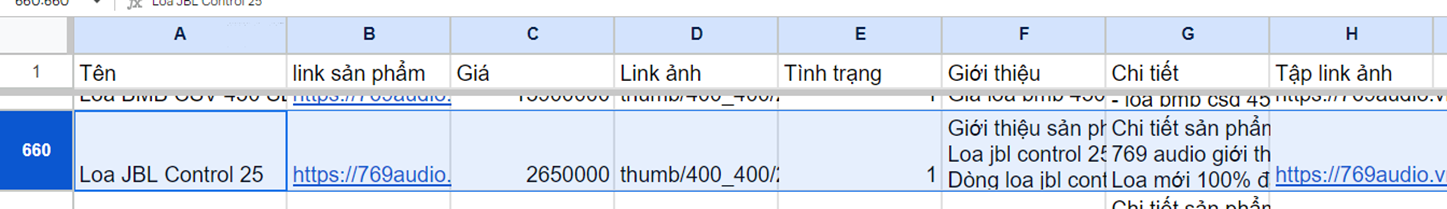
**Row: 658** - 2650000.0

**Row: 658** - thumb/400\_400/2/upload/product/664225935743.jpg

**Row: 658** - 1.0

**Row: 658** -[https://769audio.vn/upload/images/jbl%20control%2025%](https://769audio.vn/upload/images/jbl%20control%2025%25) ...

Row: 667 - Tên: Loa JBL Control 25 ...



*DataFrame starts index at 0, whereas CSV starts index by 2 (column name row at index 1), therefore Row 658 in DataFrame means Row 660 in CSV*

* One trade-off for this is chunk bloat since we must collect more chunks from the database beside the vector store. Fortunately, there are not many additional chunks as they are just brief product information like name, price, status, etc.

total\_letters = 0

for doc in initial\_docs:

# Count only alphabetic characters in each document's content

total\_letters += sum(c.isalpha() for c in doc.page\_content)

print(f"Total number of letters in Initial docs: {total\_letters}")

Total number of letters in Initial docs: **10661**

total\_letters = 0

for doc in filtered\_docs:

total\_letters += sum(c.isalpha() for c in doc.page\_content)

print(f"Total number of letters in Filtered docs: {total\_letters}")

Total number of letters in Filtered docs: **12250**

1. **Response Generation:**

* *Prompt:*
* Our chatbot understands the context of the conversation by a pre-defined premise for both the input and output process, which helps the bot to "roleplay" as a retailer who sells audio electronics.

premise = """

Giả sử bạn là một chuyên gia về thiết bị điện tử âm nhạc. Bạn đang làm việc tại 769audio, một trong ba nhà phân phối thiết bị âm nhạc hàng đầu tại TP.HCM. Bạn sắp tư vấn cho một khách hàng về phương pháp bán hàng phù hợp nhất để bán hàng cho họ, dựa trên thông tin được cung cấp.

...(more requirements)

"""

* *Rephrase:*
* To avoid the obscurity of the user's queries, *contextualize\_q\_system\_prompt* formulates them into more semantic and grammatical queries, increasing the bot's retrieval performance.

**contextualize\_q\_system\_prompt** = """Given a chat history and the latest user question \

which might reference context in the chat history, formulate a standalone question \

which can be understood without the chat history. Do NOT answer the question, \

just reformulate it if needed and otherwise return it as is."""

contextualize\_q\_prompt = ChatPromptTemplate.from\_messages(

[

("system", input\_premise + "\n\n" + contextualize\_q\_system\_prompt),

MessagesPlaceholder("chat\_history"),

("human", "{input}"),

]

)

You: **201 seri 4 nhieu vay chi**

Contextualized Prompt: **Bạn có thể cho tôi biết thêm về 201 seri 4 mà bạn đang đề cập đến không?**

* *Chat history:*
* This maintains and utilizes chat history to enhance the retrieval performance. This mechanism provides context to the retrieval functions, allowing the bot to better understand what that user needs to talk about from past interactions. This is a crucial part of a coherent conversation flow and more accurate and relevant responses.

qa\_prompt = ChatPromptTemplate.from\_messages(

[

("system", premise + "\n\n" + **feedback\_content** + "\n\n" + "{context}"),

MessagesPlaceholder("chat\_history"),

("human", "{input}"),

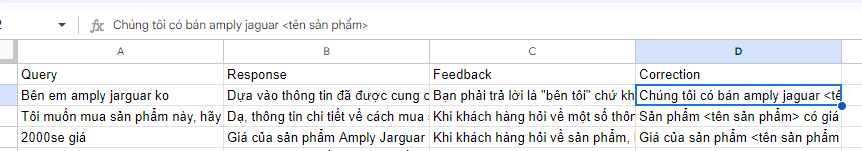
]

)

# Initialize memory and QA system

question\_answer\_chain = create\_stuff\_documents\_chain(llm, qa\_prompt)

* We incorporated feedback as a basic form of Reinforcement Learning to help the bot adapt and learn the response format that actual customers expect (which is *feedback\_content* we used in *qa\_prompt*).



feedback\_content = """

Here are the feedback of customers. Please learn these feedback so that you don't repeat your mistakes.

...

"""

# Accumulate corrections based on feedback dataframe

for index, row in feedback\_df.iterrows():

if pd.notna(row['Correction']):

feedback\_content += f"""

If a user asks: \"{row['Query']}\", you shouldn't answer like this: \"{row['Response']}\",

as the feedback is {row['Feedback']}, but you should answer: {row['Correction']}\n\n

"""

* *Response:*
* Finally, we configured and initialized the RAG for my bot, enabling it to comprehend and respond to questions accurately.

# Create and return the RAG chain

return create\_retrieval\_chain(history\_aware\_retriever, question\_answer\_chain)

* For LangChain LLM, we utilized the ChatGPT model, specifically gpt-4o-mini, for quick responses, cost efficiency. To ensure the highest accuracy in our bot's responses, we set the temperature parameter to 0, optimizing predictivity.

llm = ChatOpenAI(model='gpt-4o-mini', temperature=0)

* For improved conversation flow, we added a "clear context" feature to reset memory, allowing users to initiate new conversations or sometimes seek alternative answers to specific questions.

if query == "clear":

chat\_history = []

qa = initialize\_rag(llm, data, retriever)

print("ChatBot: ", "Hi, I'm 769audio ChatBot. How can I help you?")

continue

*Clear context in Google Colab version*

* For User Interface, reload the page will process “clear context”

**2.3. System Design**

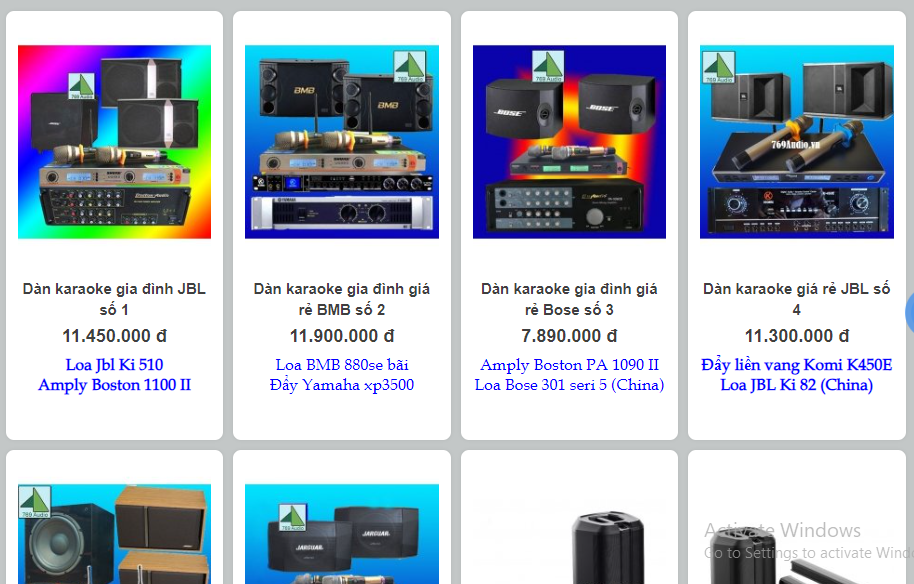


3. Results

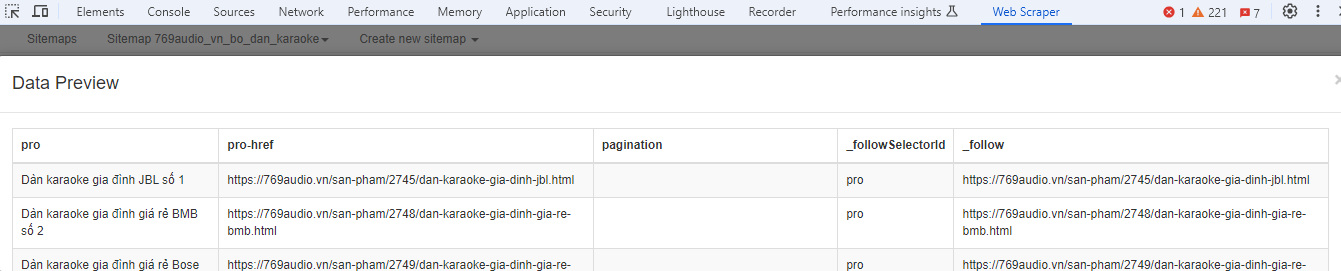
**3.1. Data Preprocessing**

1. **Data Collection with WebScraper.io:**

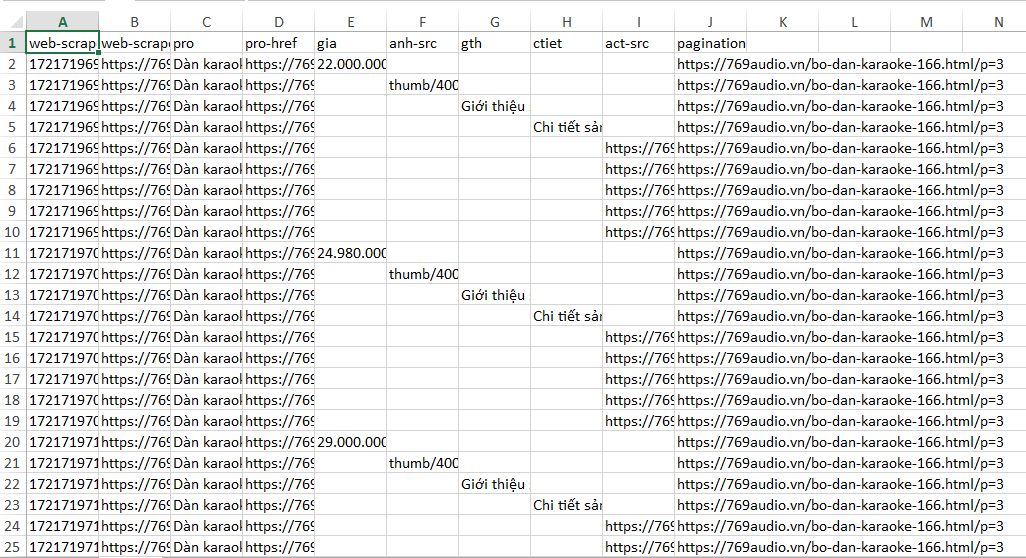
* Data from [https://769audio.vn](https://769audio.vn/):



* Usage of WebScraper.io: We utilized WebScraper.io to automate the data scraping from the website.



* Export to CSV: The scraped data was exported into CSV format to make the data manageable and easier to manipulate



1. **Data Transformation and Integration:**

* Product Features Merging and Appending Data: After collecting the initial data, we merged product features for each product. And all individual CSV files were appended into a single file



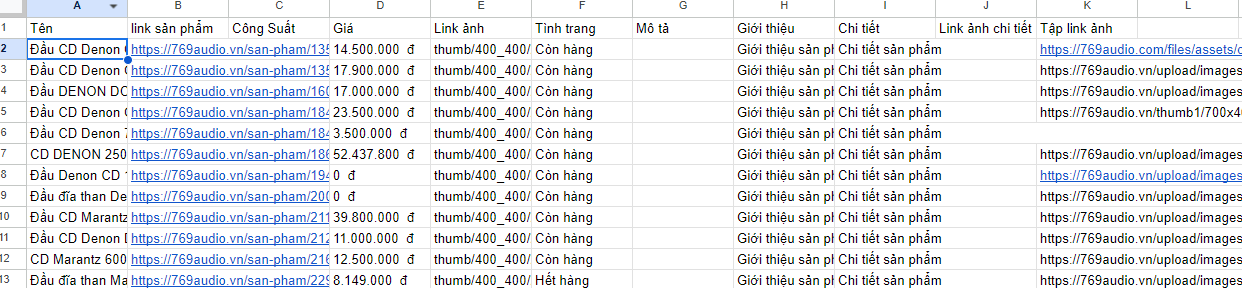
1. **Additional Data Crawling with BeautifulSoup4:**

* Crawling Data Status Feature: WebScraper somehow could not scrape status data, so we used BeautifulSoup4 to crawl them



1. **Final Database Creation:**

* Integration of New Features: The newly crawled data features (status information) were added to the previously consolidated data file.

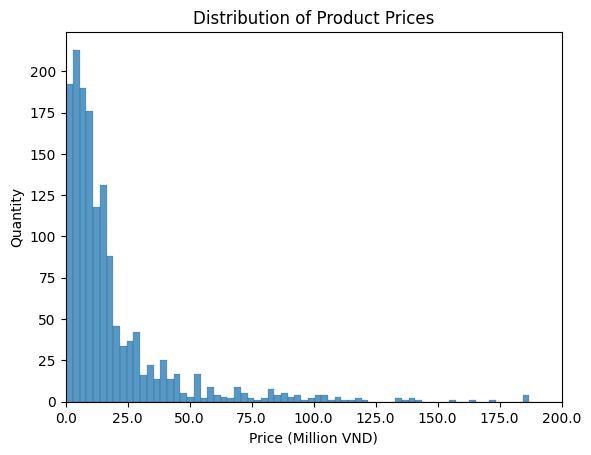


* Data Preprocessing: We removed empty columns, set types for “Giá” (price) column, changing “Tình trạng” (status) column into binary values and trimming unnecessary space in “Chi tiết” (detailed content) column



**3.2. Exploratory Data Analysis (EDA)**

1. **Distribution of Product Prices:**



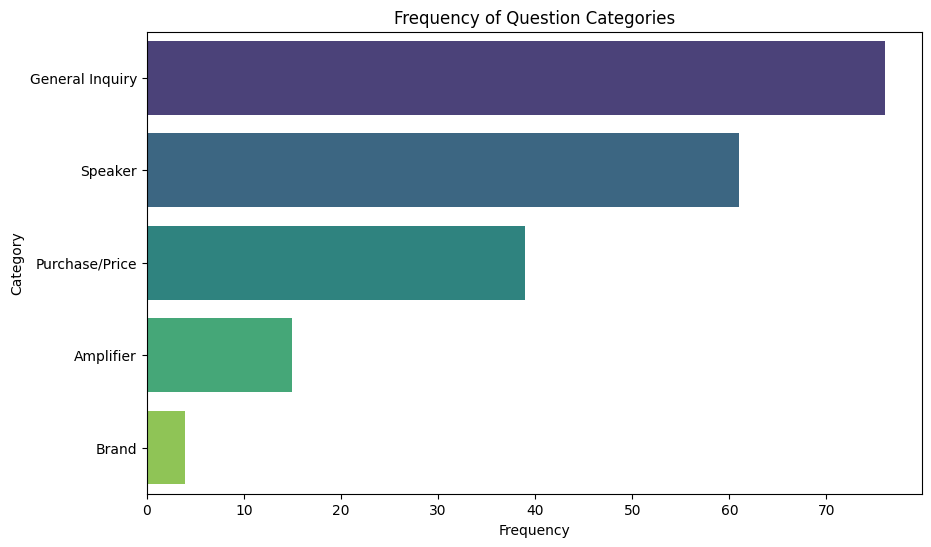
* Focusing on the range from 0 to 200 million VND, where most products are priced, the distribution is heavily skewed towards the lower end, with a significant drop in quantity as prices approach 50 million VND. This indicates the bulk of the product offerings are economically priced, appealing to an affordable market segment.

1. **Word Cloud for Product Contents:**



* The word cloud illustrates a diversity of terms, many related to features and specifications like "âm thanh" (sound), "chất lượng" (quality), "thiết kế" (design), "sử dụng" (usage),... Terms like "bảo hành" (warranty), "công suất" (capacity), or "kết nối" (connection),... should suggest these are key selling points or concerns.

1. **Frequency of Question Categories:**



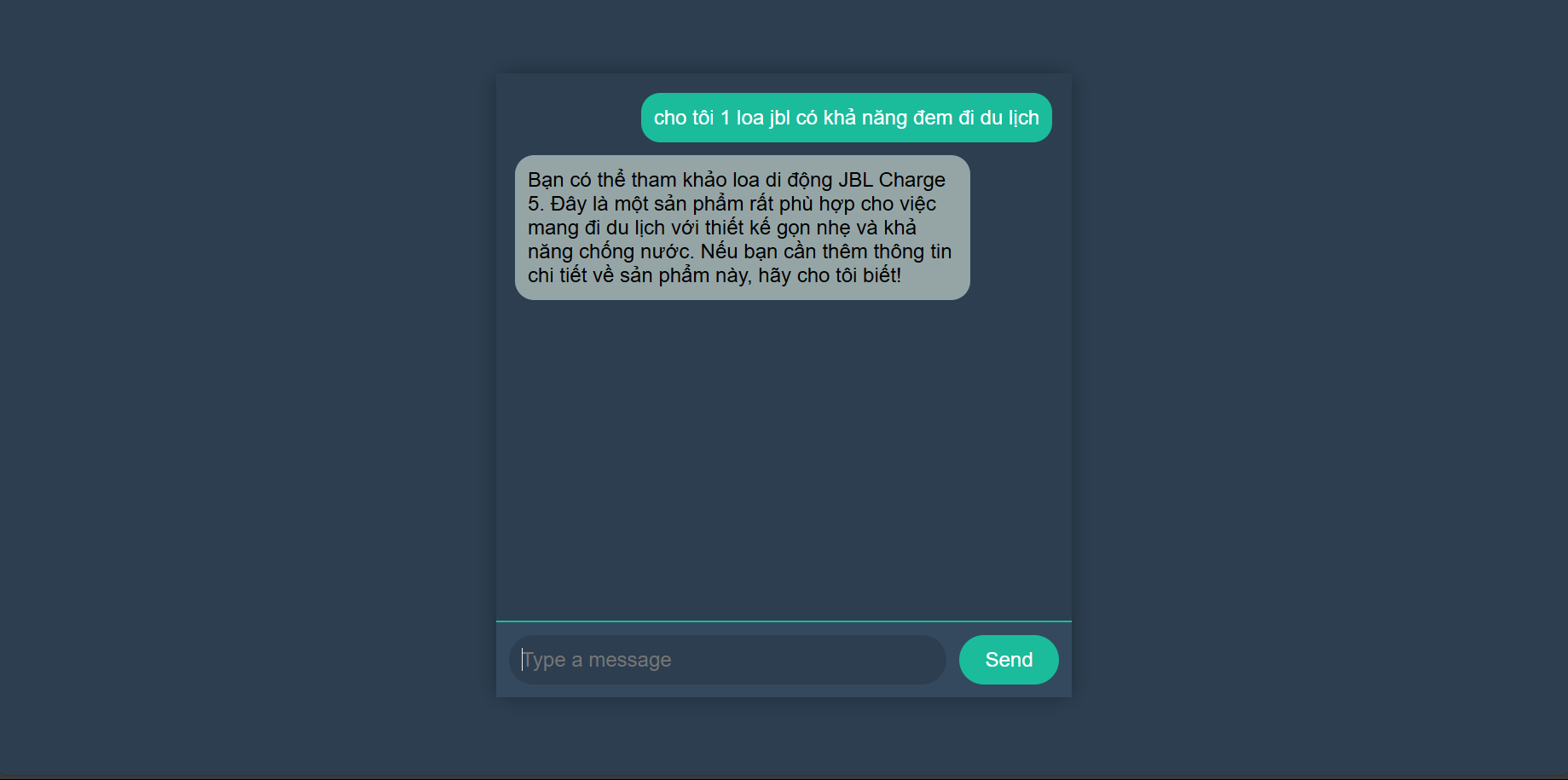
* As we can see, the "General Inquiry" and "Speaker" categories dominate the questions, implying customers' high interest or concerns in these areas, whereas other keywords receive less attention. This could mean customers need guidance or have specific preferences for those generic inquiries and speakers. Lesser focus on "Brand" queries may suggest a lack of familiarity with the available brands among buyers and they prioritize specifically personal needs, such as functionality, features, and price,...

1. **Common Words in Real Responses:**



* Besides commonly used words in chat conversations, this word cloud includes frequent recommendations of specific brands like "JBL", "Bose", or "Denon", with particular product types such as "Loa" (speaker), "Micro" (microphone), or "Amply" (amplifier), suggests purchase trends which are frequently address in responses. Additionally, some keywords indicating discounts and warranties like "Giá" (price), or "Chính hãng" (authentic) might reflect these are key factors giving an impulse to customer purchases.

**3.3. User Interface**



* The interface consists of 2 main parts: message input frame and conversation frame. The human message will be on the right in green and the chatbot's response will be on the left in gray. When you press the 'Send' button, the message will be sent. Users can also press the 'Enter' key on the keyboard to send the message.

**3.4. Testing and Improvements**

1. **First Version overview:**

* We initiated the integration of the LangChain ChatBot using a minimal configuration available in the LangChain documentation. Below are the details of the setup and outcomes:
* **Text Splitting Parameters**:

text\_splitter = RecursiveCharacterTextSplitter(chunk\_size=500, chunk\_overlap=20)

* **LLM Configuration**:
* This setup used gpt-3.5-turbo as the default model:

llm = ChatOpenAI(temperature=0.9)

* **Document Search Retriever**:

retriever=docsearch.as\_retriever()

* **Performance Evaluation**:
* After conducting **68** chat sessions that included chat history, the system achieved an accuracy rate of **57.35%**.

**Precision: 1.00**

**Recall: 0.5735**

**F1-score: 0.729**

**Confusion Matrix:**

**[[ 0 0]**

**[29 39]]**

* Remind: Experienced sellers assessed this rate alongside our team members. + Zalo data
* **Observation**: ChatBot Performance Issues
* We decided to halt the expansion of test data due to evident issues with the bot's performance. Addressing these fundamental problems was prioritized over continuing tests that consistently yielded poor results. This approach allowed us to focus on resolving critical flaws without expending resources on ineffective testing.
* Some evaluations we observed:
* **Understanding the Premise / Roleplay**: The bot failed to grasp the established premise, resulting in incorrect responses that did not align with the intended scenario.
* **Chat History Integration**: There were shortcomings in how the bot followed the chat history. It occasionally produced responses that were irrelevant to previous conversations.
* **Semantic Accuracy**: The bot often delivered semantically incorrect responses, leading to confusion among users.

⇒ These observations indicated a need for further refinement in the bot’s contextual understanding and response accuracy to enhance user interaction quality.

1. **Second Version - Major Adjustments to the ChatBot**

* After persistent issues persisted through over 60 chat sessions, we thoroughly reviewed our code and implementation strategies. Despite numerous prompt adjustments, no significant improvements were observed. This led to an extensive period of research and subsequent modifications.
* It turned out that the code configuration errors hindered the integration of the RAG system with our existing premises and chat history. Therefore, we have taken steps to rectify these issues in the code.
* Key Adjustments Made:
* **Improved RAG Initialization**: Enhanced the retrieval-augmented generation setup with more sophisticated functions, enabling better configuration of the LLM for input and output paraphrasing.
* **Optimized chunk size**: Determined the optimal chunk size by calculating the average character count per sentence across the database.
* **Incorporated Reinforcement Learning**: Implemented a simple reinforcement learning mechanism by capturing user feedback on each incorrect response during testing, allowing for dynamic adjustments.
* **Performance Evaluation:**
* After implementing these changes, we conducted 189 chat sessions that included chat history. The results showed a marked improvement:

**Precision: 1.00**

**Recall: 0.8377**

**F1-score: 0.9117**

**Confusion Matrix:**

**[[ 0 0]**

**[ 31 160]]**

* Given that the test data were primarily derived from actual chat conversations between customers and sellers, we aimed for a precision rate where the sellers' responses to customer inquiries were 100% accurate.
* **Observation**: Identifying Areas for Further Improvements
* While the increase in accuracy from 54% to 84% marked significant progress in chatbot performance, several critical issues still required attention:
* **Handling Complex Queries**: The bot struggled with sophisticated queries that involved multiple entities, often generating incorrect information.
* **Analyzing User Needs**: The bot's performance in understanding and analyzing user needs was suboptimal. This was partly due to the limitations of our database, which only contained basic product features and specifics.
* **Retrieval Limitations**: The LangChain Retriever primarily extracted chunks based on keywords, which were predominantly found in the "Giới thiệu" (Introduction) and "Chi tiết" (Details) columns of our database. This approach often resulted in the omission of relevant information from other columns, leading to inadequate responses.
* **Keyword Recognition**: The bot had trouble with keywords frequently used by users but uncommon in the database, which made it challenging to accurately retrieve the correct products.

⇒ These observations underscored the need for a targeted enhancement strategy focusing on advanced query handling, enriched data resources, and improved retrieval techniques to elevate the bot's overall effectiveness and user satisfaction.

1. **Third Version - Enhancements in Retrieval**

* To refine our retrieval strategy, we shifted from merely gathering chunks related to queries. Instead, we aimed for each chunk to represent a product based on its row index from the metadata.
* This adjustment involved collecting all pertinent product features from the specified rows, excluding those from the "Giới thiệu" (Introduction) and "Chi tiết" (Details) columns which were already retrieved through docsearch.
* **Performance Evaluation:**
* Testing was conducted on 201 chat sessions. The evaluation metrics were as follows:

**Precision: 1.00**

**Recall: 0.8873**

**F1-score: 0.9403**

**Confusion Matrix:**

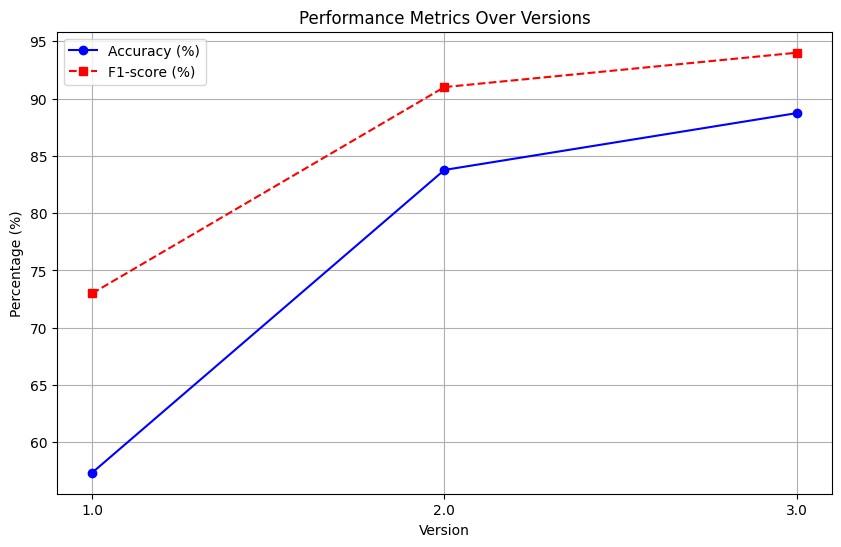
**[[ 0 0]**

**[ 28 173]]**

* **Observation:**
* The enhancements led to a notable reduction in incorrect responses from our bot, attributed to the comprehensive retrieval of needed product information. Additionally, the bot handled complex queries involving multiple products (such as comparisons and summarizations) more effectively.
* However, certain issues from the second version persisted:
* **Keyword Recognition:** The bot struggled with keywords that were commonly used by users but were not prevalent in the database, leading to difficulties in retrieving the correct products.
* **Minor False Responses:** Despite improvements, minor inaccuracies still occurred, particularly when handling multiple intricate entities and requests.

1. Aggregation of Results:

| **Version** | **Accuracy (%)** | **F1-Score (%)** | **Improvements** | **Reasons for Incorrect Answers** | **Common Error Chat Scenarios** | **Analysis of Accuracy Enhancement** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 57.35 | 72.9 | Initial release, basic features | Code configuration errors, leading to prompt and chat history recognition fail | grammatically incorrect queries, incorrect product name, two-entity comparisons, Use of synonyms, indirect references using pronouns like “nó” (it), asking similar items |  |
| 2 | 83.77 | 91.17 | RAG upgrade, Chunk size optimization, Feedback Implementation | Complex comparison issue, retrieval and keyword limitations | curt questions, queries about items similar to those in the database but non-existent, analysis-requiring queries, recomming products, two-entity and three-entity comparisons, uncommon product keywords (“loa thanh E”), recognizable but incorrect item names | Improved understanding of the conversation's premise and history, enhancing flow and relevance in responses.  Applied user feedback to refine responses in similar future queries, increasing reliability and accuracy.  Enhanced chunk size and query rephrasing techniques improved data retrieval and response precision. |
| 3 | 88.73 | 94.03 | Retriever customization | Suboptimal keyword recognition, minor false response | Queries about items similar to those in the database but non-existent, recommending products, two-entity and three-entity comparisons, uncommon product keywords (“loa thanh E”), recognizable but incorrect item names | Expanded the bot’s ability to retrieve detailed product information, enhancing the accuracy and relevance of responses, especially complex queries that involve multiple entities. |



4. Projected Impact

**4.1. Accomplishments and Benefits**

1. **Accomplishments:**

* **Enhanced Accuracy:** Across three versions of our project, we have seen a significant increase in the accuracy rate from 57% to 89%. This improvement has dramatically reduced the occurrence of faulty responses, thereby enhancing the reliability of our bot for users.
* **Optimized Retrieval Methodology:** We transitioned from merely retrieving similar search chunks to extracting all relevant product rows in the database. This development has enabled our bot to handle more complex and detailed user queries with greater precision.
* **User Engagement:** Implementing a feedback mechanism has allowed our bot to better understand users' needs and context. This has fostered increased user satisfaction and trust in the system, contributing to a more engaging user experience.

1. **Benefits:**

* **Improved Customer Care Service:** Our system provides instant, 24/7 support, significantly reducing wait times and enhancing the overall user experience. This is particularly beneficial for customers who may be reluctant to engage directly with sellers, ensuring they receive timely assistance and feel more comfortable interacting with our service.
* **Cost Efficiency:** By automating responses and managing multiple customer interactions simultaneously, our chatbot reduces the need for extensive human customer service resources. This allows us to allocate human talent to address more complex and nuanced customer issues, optimizing our workforce and reducing operational costs.
* **Market Competitiveness:** The implementation of our cutting-edge chatbot solution has not only improved customer satisfaction but also significantly differentiated us from competitors. This strategic advantage should position us as a leader in innovation within our industry, attracting both new customers and potential business partnerships.

**4.2. Future Improvements**

* **Integration with Company Website**: Fully integrate the chatbot into the company's website to ensure seamless, user-friendly, and accessible assistance for a diversity of customers, encouraging more interactions and instant support. What is more, integrating the chatbot into a client-server model will facilitate the tracking of customer behaviors and portraits, setting a foundation for further improvements.
* **Real-time Data Update**: Incorporating real-time data updates into the chatbot can significantly enhance its responsiveness and relevance, especially when our real data on the website are usually dynamic, depending on marketing fluctuation.
* **Reinforcement Learning for Keyword Optimization**: Transition from a static model of using user feedback as CSV inputs to implementing dynamic Reinforcement Learning (RL) techniques, such as Q-Learning and Deep Q-Networks (DQN). This will enable the chatbot to adapt to new queries and responses more effectively over time and manage the increasing volume of data without expanding the prompt size unnecessarily.
* **Sales Pipeline Enhancement**: Incorporate the company’s refined sales pipeline strategies into the chatbot’s functionalities. This will automate key sales processes such as lead generation, lead nurturing, and handling initial pre-sales inquiries. By automating these aspects, the chatbot will not only increase sales efficiency but also optimize the allocation of human resources within sales departments.
* **Product Recommendations Using Association Rules**: Implement association rule learning to generate personalized product recommendations based on extensive user behavior and purchase history data. This method will leverage receipt data to facilitate effective upselling and cross-selling, enhancing revenue potential without relying solely on experienced sales staff.

5. Team Member Review and Comment

|  |
| --- |
| https://lh7-rt.googleusercontent.com/docsz/AD_4nXddrcPUkF-v7eU8RuVAweJW4FpvyhSU9v2GmNXOz-6rE6eu2_X1EzvYcWUMoQvA8qW_40L78HRSHUk4-RN4xVtEtZSuaKoYnoI5TmyjDJpUtjJlw1HiwUhLRKqByOOcsZEz0ULJhnf2Suupa0M7jHKT5KCX?key=Sae-IPGNfwZYkFT_cIH3YgC:\Users\Khanh\AppData\Local\Packages\Microsoft.Windows.Photos_8wekyb3d8bbwe\TempState\ShareServiceTempFolder\123456.jpeg  Tran Thi Quynh Dung – Hua Quang Dat – Vuong Minh Khanh – Phung Trung Hieu |

|  |  |
| --- | --- |
| NAME | REVIEW and COMMENT |
| Tran Thi Quynh Dung | Executes seemingly simple tasks that are actually complex. Significantly improves my leadership ability in team roles. |
| Hua Quang Dat | Actively contributes ideas and engages in discussions. Demonstrates strong responsibility, contributes to model building. |
| Phung Trung Hieu | Supports the group leader with auxiliary tasks and contributes ideas. Reliable and hardworking team member. |
|  |  |
|  |  |

6. Instructor Review and Comment

|  |  |  |
| --- | --- | --- |
| CATEGORY | SCORE | REVIEW and COMMENT |
| IDEA | \_\_/10 |  |
| APPLICATION | \_\_/30 |  |
| RESULT | \_\_/30 |  |
| PROJECT MANAGEMENT | \_\_/10 |  |
| PRESENTATION & REPORT | \_\_/20 |  |
| TOTAL | \_\_/100 |  |