CHATBOT FOR KITCHEN INVENTORY MANAGEMENT

Joshua Muiruri
Department of Computer Science
University of Nottingham

ABSTRACT

This report presents the development of a chatbot named Gordon, designed to assist users in managing a kitchen inventory at a restaurant through natural language interactions. The chatbot performs tasks such as checking inventory levels, adding and removing items, tracking expiration dates, and generating inventory-related reports. It is implemented using Python with NLTK for natural language processing, the system is evaluated based on usability, task completion, and accuracy of intent matching. The report concludes with a discussion of system strengths, challenges faced, and potential future enhancements.

1 INTRODUCTION

1.1 Purpose

The primary purpose of Gordon is to streamline the process of managing a kitchen inventory specifically at a restaurant where stock is constantly changing, but it can also be used in any other kitchen. By interacting with the chatbot, users can easily check stock levels, add or update items, remove items, and generate reports. This seeks to reduce the manual effort needed when handling inventory management on paper, helping to maintain an organised kitchen.

1.2 Motivation

Efficient inventory management is a crucial yet often overlooked aspect of daily life, especially in domestic kitchens or small businesses. Many individuals struggle to keep track of what items they have, how much is left, and when items expire. This can lead to waste, unnecessary restocking, or running out of essential supplies.

Chatbots powered by natural language processing (NLP) present an intuitive solution to this problem. By enabling users to interact in plain language, a kitchen inventory chatbot can bridge the gap between complex inventory systems and non-technical users. The motivation for this project stems from the need for an easy-to-use, conversational interface that simplifies inventory management tasks while enhancing efficiency and reducing waste.

1.3 Overview

This report details the development of a chatbot designed to assist users in managing kitchen inventory through natural language interactions. The chatbot provides functionality such as:

- Checking current inventory levels for specific items or all items.
- Adding, removing, and updating inventory items.
- Generating reports, such as recent inventory changes and usage summaries.

The system is implemented using Python and integrates NLP techniques for intent matching and response generation. Additionally, the chatbot offers conversational features like small talk, question answering, and error handling to improve user engagement and system usability. The report covers the chatbot's architecture, conversational design, evaluation, and potential areas for improvement.

2 SYSTEM ARCHITECTURE

2.1 Functionality

Inventory Check:

Users can inquire about the quantity of specific items in stock or request a complete list of all inventory items. This feature allows users to:

- Check the exact stock level of a specific item, e.g., "How much milk do we have?"
- Request a detailed list of all inventory items along with their current quantities.
- Identify low-stock items by prompting the chatbot to highlight supplies below a predefined threshold.

Add/Update Inventory:

The chatbot allows users to:

- Add new items to the inventory with specified quantities, e.g., "Add 10 apples to inventory."
- When an item is already in the inventory it will instead ask to add to the existing quantity.

Remove Items:

Users can efficiently manage outdated or unnecessary items by:

- Removing a specific quantity of an item from the inventory, e.g., "Remove 2 cans of soup."
- Deleting an item entirely if it’s no longer needed, e.g., "Delete bananas from inventory."

Reports:

The chatbot can generate various types of reports to provide insights into inventory usage and trends:

- **Recent Changes**: Displays a log of recently added, removed, or updated items.
- **Inventory Valuation**: Calculates the total value of all items in stock based on predefined prices. These reports are particularly useful for small businesses or users managing large inventories, as they enable effective tracking and planning.
- **Low Stock items**: Shows a list of items that are flagged with low stock.

Small Talk:

The chatbot is equipped with a library of predefined responses to engage users in casual conversations. Examples include:

- Offering witty replies to questions like "What's up?"
- Providing light-hearted interactions that make the chatbot feel more personable and less transactional.

This feature enhances user experience by adding a layer of friendliness and accessibility to the chatbot.

Help and Guidance:

Users can access a help menu to understand the available commands and features of the chatbot. This includes:

- Explaining how to use key functionalities with examples, e.g., "You can say 'Add 5 apples' to add items to inventory."
- Guiding users within prompts to navigate the chatbot's capabilities, e.g., suggesting valid commands when an invalid input is provided.
  This ensures that even first-time users can easily interact with the chatbot without prior training or technical expertise.

Question and Answer:

The chatbot has access to a dataset of random questions and answers, allowing users to:

- Ask general or domain-specific questions outside inventory management.
- Engage in Q&A sessions to test the chatbot's knowledge or for casual conversation.
  This feature extends the chatbot's utility beyond inventory management, providing additional value to users.

2.2 Implementation

The development of the kitchen inventory chatbot combined multiple technologies, systems, and techniques to deliver an intuitive and efficient solution. This section outlines the technologies used, the key components of the system, and a flowchart illustrating the overall architecture.

Technologies Used

The chatbot leverages the following technologies to achieve its functionality:

Natural Language Toolkit (NLTK): For natural language processing tasks including tokenisation, lemmatisation, and stop words.

Numpy: Used to create vectors of data to parse into similarity models**.**

ScikitLearn: Used to create and train similarity models used for intent matching.

MySQL: Used as a database for inventory management, storing several details about the inventory.

Key Components

To being with I structured the chatbot into three main classes: `` `InventoryBot` ``, `` `TextProcessor` ``, and `` `InventoryDatabase` ``.

The `InventoryBot` class serves as the core of the chatbot, managing user interactions, processing inputs, and executing various inventory-related commands. It utilises the *TextProcessor* class to handle text processing tasks such as tokenisation, lemmatisation, and similarity calculations.

The `InventoryDatabase` class is responsible for managing the inventory database, including adding, updating, and retrieving inventory items. This modular design allows for clear separation of concerns, making the chatbot easier to maintain and extend.

Intent Matching System

The chatbot uses an intent matching system to interpret user input and determine the appropriate action. The system does this by tokenising and lemmatising user input and comparing it to a premade corpus of data stored in a dictionary. Below is a snippet of the code to do this:

```
# Process user input
tokenized_input = self.processor.tokenize(user_input)
lemmatized_input = ' '.join(self.processor.lemmatize(tokenized_input))
non_lemmatized_input = ' '.join(tokenized_input)

# Get the index of the most similar small talk response and its similarity
index, similarity = self.processor.tfidf_cosim(small_talk, user_input)

if similarity > 0.91:
    key = small_talk[index]
    closers = ['bye', 'goodbye', 'see you later', 'stop', 'no', 'nothing']
    # check if the user wants to end the conversation
    if (key or user_input) in closers:
        print(random.choice(self.small_talk_responses[key]).replace("{user_name}",
self.user.get_name()))
        self.running = False
        return

    else:
        response = self.small_talk_responses[key]
        print(random.choice(response).replace("{user_name}", self.user.get_name()))
        self.continued = True
        return
```

As you can see highlighted, user input and small talk corpus are passed into a function that calculates the similarity using a model. Specifically, cosine similarity is used since it is best for larger datasets. The code for the function is shown below:

```
def tfidf_cosim(self, doc, query):
    query = [query]
    tf = TfidfVectorizer(use_idf=True, sublinear_tf=True)
    tf_doc = tf.fit_transform(doc)
    tf_query = tf.transform(query)
    cosine_similarities = cosine_similarity(tf_doc,tf_query).flatten()
    most_similar_index = cosine_similarities.argsort()[:-2:-1]
    # Return the index of the most similar document and the cosine similarity
value
    results = [most_similar_index[0],
cosine_similarities[most_similar_index]]
    return results
```

This `tfidf_cosim` function converts the data (doc) and user input (query) into vectors and then finds the similarity between them. The function then returns two things:

- The **index** of the most similar sentence in the data – used to access the response given to the user.
- The **similarity rating** (between 0 and 1) – used to decide how to respond to the user based on similarity. The higher the rating the more confident the chatbot in its rating.

Also, here's an example corpus for reference:

```
 Small talk = {"how is it going": ["It's going great, thank you
for asking!", "All good here! How can I assist you?"],
  "good morning": ["Good morning! How can I assist you today?",
"Morning! Hope you're well!", "Good morning! Ready to help?"],
  "thank you": ["You're welcome!", "No problem at all!", "Glad I
could help!", "Anytime! Happy to assist."]
}
```

After it gets the results from the similarity function the chatbot does different things depending on the context.

- For small talk and Q&As it accesses the most similar response and prints to the user.
- For kitchen specific commands it the query intent to a unique pin, e.g. 101, which is used by a function called '`execute_intent`' to create the right response.

Shown below is a snippet of the execute intent function:

```
def execute_intent(self, intent, user_input):
    if intent == 101:
        return self.get_item(user_input)
    elif intent == 102:
        return self.get_all_items()
    elif intent == 201:
        return self.add_item(user_input)
```

Database Management

The chatbot uses MySQL for storage of the inventory database since all transactions with this database are saved even after the chatbot is closed. Instead of having to directly interact with the database through an SQL terminal, the class `InventoryDatabase` handles all transactions and queries to the database. The code below initialises the class by creating the database if it doesn't already exist and then creating and filling the tables. There are two tables, the item table for all item details, and the transactions table to store all transactions with the database.

```
class InventoryDatabase:
    def __init__(self, db_path="data/inventory.db"):
        # Initialise the database connection
        self.conn = sqlite3.connect(db_path)
        self.cursor = self.conn.cursor()
        self.create_tables()

    def create_tables(self):
        """Creates the database tables if they don't exist"""
        self.cursor.execute('''
            CREATE TABLE IF NOT EXISTS items (
            item_id INTEGER PRIMARY KEY,
            item_name TEXT NOT NULL,
            category TEXT DEFAULT 'Other',
            unit_price REAL DEFAULT 3.00,
            quantity_in_stock INTEGER DEFAULT 0,
            restock_threshold INTEGER DEFAULT 2
        )''')
        …
        self.conn.commit()
```

Also below is the code for the `add_item` function within the `InventoryDatabase` class. This function is used by an extension of this function the `InventoryBot` class that handles user inputs, prompts and error handling to complete the transaction properly.

```
def add_item(self, item_name, quantity):
    """Insert a new item into the items table"""
    self.cursor.execute('''
        INSERT INTO items (item_name, quantity_in_stock)
        VALUES (?, ?)
    ''', (item_name, quantity))
    self.conn.commit()
```

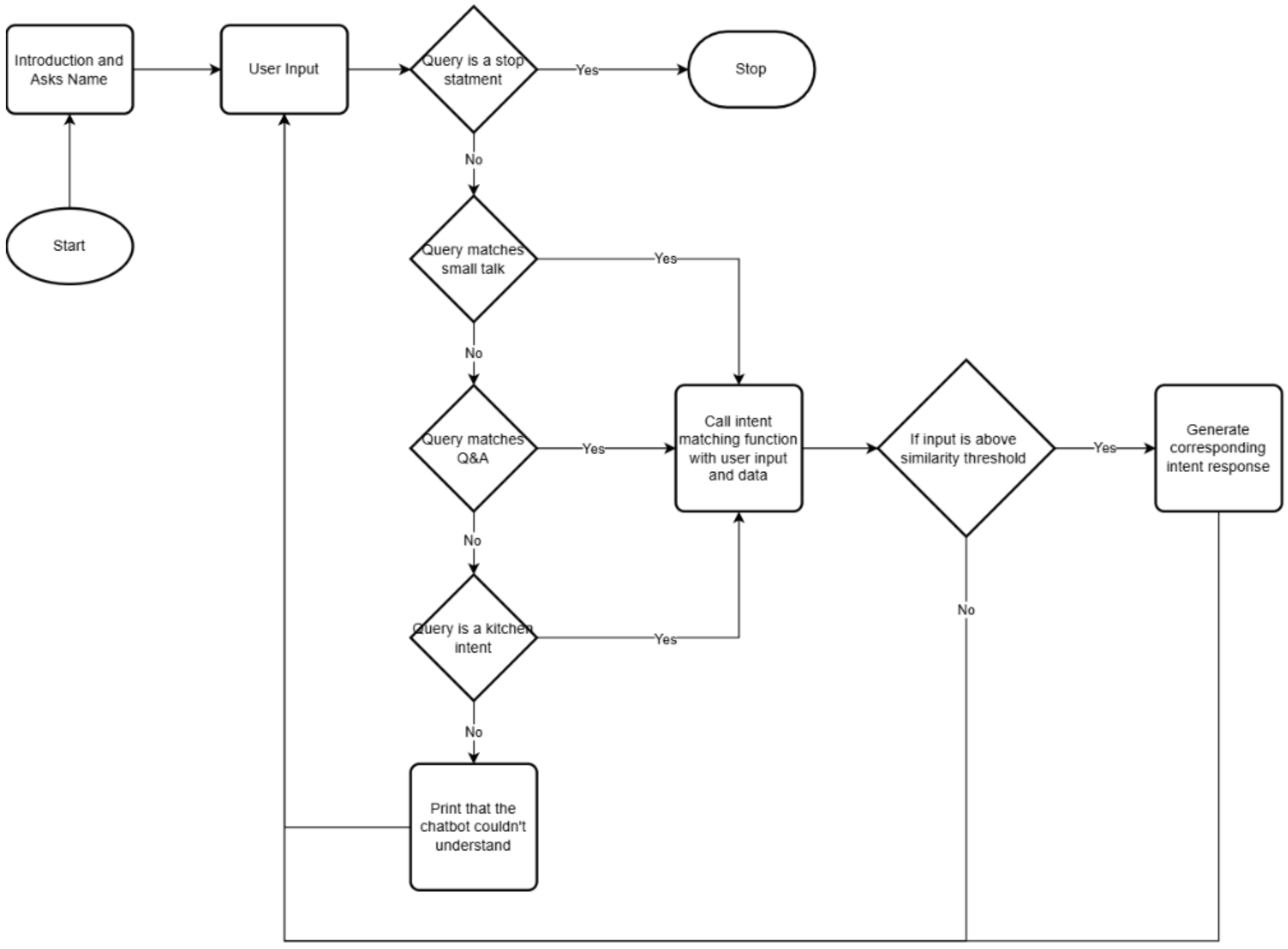
There are several more functions that all query the database for some result like getting an item, showing recent updates etc.

3 CONVERSATIONAL DESIGN

The chatbot's conversational design was crafted to ensure users can interact intuitively while effectively managing their kitchen inventory. This section outlines how prompts, error handling, discoverability, and personalisation were incorporated to enhance user experience.

Flowchart

Here is a flowchart that better illustrates how standard use of the chatbot operates.


Introduction and Asks Name
User Input
Query is a stop statment
Yes
Stop
Start
No
Query matches small talk
Yes
No
Query matches Q&A
Yes
Call intent matching function with user input and data
If input is above similarity threshold
Yes
Generate corresponding intent response
No
Query is a kitchen intent
Yes
No
No
Print that the chatbot couldn't understand

3.1 Prompt Design

Prompts play a critical role in guiding users through the chatbot's functionality. They are designed to be concise, informative, and context-aware, ensuring users always understand their options.

Help Menu Prompts

The chatbot includes a **help command** that lists all available commands and examples for ease of use. This menu is accessible by typing "help" and provides users with a clear overview of the chatbot’s capabilities. For example:

****Help Menu****

Here are some things you can ask me:

1. "How much of [item] do we have?"

2. "Add 10 apples to inventory."

3. "Remove 5 bananas from inventory."

4. "What items are running low?"

By showcasing commonly used commands with placeholders, the help menu simplifies task initiation and encourages users to explore more features.

Discoverability

Discoverability ensures users can easily identify the chatbot's capabilities without prior knowledge. The chatbot achieves this through:

Proactive Suggestions - When users encounter errors, the chatbot suggests valid commands to guide them.

Help Command: The "help" command acts as a centralised resource for discovering the chatbot's full functionality, providing examples for inventory checks, adding or removing items, and more.

Context-Aware Prompts: After completing an action, the chatbot suggests the next logical step. For example:

> User: "Add 5 apples to inventory."
>
> Chatbot: "5 apples added to inventory. Would you like to check your inventory or add more items?"

This approach fosters a seamless and intuitive user experience.

Error Handling

The chatbot's error handling mechanisms are designed to ensure robustness and clarity, even when users deviate from expected inputs.

Handling Invalid Commands

When the chatbot cannot recognise a user's intent, it provides helpful fallback responses. For example:

User Input: "Can you give me apples?"

Chatbot Response: "I didn't understand that. Try saying 'Add 5 apples' or 'How much of [item] do we have?'"

This strategy combines error acknowledgment with actionable guidance, ensuring users can quickly correct their input.

Handling Inventory-Specific Errors

In cases where inventory-related errors occur (e.g., trying to remove more items than available), the chatbot provides specific feedback:

User Input: "Remove 10 bananas" (when only 5 are in stock)

Chatbot Response: "You only have 5 bananas in stock. Try removing a smaller quantity or check your inventory."

These targeted responses improve user understanding and prevent unnecessary confusion.

Personalisation

Personalisation enhances the chatbot's engagement by tailoring interactions to individual users.

Using Names

At the start of the conversation the user is prompted to enter their name. The chatbot then remembers it and incorporates it into subsequent responses. For example:

User: "My name is Alex."

Chatbot: "Nice to meet you, Alex! How can I assist you today?"

Follow-Up Response: "Alex, you have 10 apples in stock. Would you like to add more items?"

Friendly Small Talk

To further personalise the experience, the chatbot supports small talk by responding to casual greetings and questions:

User: "How are you?"

Chatbot: “I’m doing great! Thanks for asking.”

This feature helps establish a friendly and approachable tone, making the chatbot feel more relatable.

4 EVALUATION

4.1 Methodology

The chatbot was evaluated through a combination of user testing and automated mock scenarios to assess its effectiveness and robustness. A group of ten users, including both technical and non-technical participants, interacted with the chatbot to perform core tasks such as adding, removing, and checking inventory. These interactions aimed to gauge the chatbot's usability and ability to handle real-world inputs. Each participant was encouraged to try both straightforward commands (e.g., "Add 5 apples") and less conventional phrasing (e.g., "Can you put five apples into stock?").

In addition to user testing, a series of automated mock scenarios simulated edge cases, including ambiguous commands, invalid inputs, and unexpected sequences. These scenarios were designed to evaluate the chatbot's intent detection and error-handling capabilities. For example, inputs like "Add something" or "Remove oranges but not apples" tested how well the chatbot could interpret incomplete or overly complex commands.

Key Metrics:

- **Task Completion Rate**: The percentage of tasks successfully completed during testing.
- **Intent Detection Accuracy**: The accuracy of identifying user intents, especially with varied phrasing.
- **Error Handling Effectiveness**: The success rate of guiding users to correct errors after receiving chatbot prompts.

4.2 Results

The evaluation demonstrated strong performance, with the chatbot achieving a **task completion rate of 90%**, indicating reliable execution of user commands. Intent detection accuracy was measured at **70%**, showing the chatbot's ability to correctly interpret a wide range of inputs, even those phrased differently from predefined examples. Error handling was similarly effective, with **80% of users successfully resolving mistakes** after receiving guidance from the chatbot.

Users praised the clarity and simplicity of commands for adding and removing inventory, noting that these features were intuitive and straightforward. The help menu, which provided examples of available commands, was particularly well-received and often cited as a key contributor to the chatbot’s usability. Additionally, the small talk feature added a friendly and engaging tone to interactions, making the chatbot feel more approachable.

However, the chatbot struggled with some ambiguous inputs, such as *"Add something"* or *"Check for items we need,"* where the lack of specific details made intent detection more challenging. These cases highlighted limitations in the chatbot's ability to handle open-ended or vague language effectively.

4.3 User Feedback

Participants provided valuable insights into the chatbot’s strengths and areas for improvement. Many highlighted the **help command** as an intuitive way to understand the system’s capabilities. One user commented, *"The help command made it easy to understand what I could ask."* Explicit confirmations for actions, such as adding or removing items, were also well-received, with another user noting, *"I liked how the chatbot confirmed actions like adding items."*

On the other hand, participants suggested improvements to enhance the chatbot's flexibility. Some noted that it struggled with less conventional phrasing and suggested making the chatbot more adaptable to varied user input. Others recommended implementing personalised suggestions based on frequently used commands, which could further streamline interactions and improve user satisfaction.

5 DISCUSSION

5.1 What Worked Well

The help function was one of the chatbot’s strongest features, providing clear instructions and examples that made the system easy to use for both novice and experienced users. Additionally, the confirmation system, combining implicit and explicit confirmations, effectively minimised user confusion and sure users could follow what the chatbot was doing. Error handling also worked quite well, which is evident from the results of testing. The small talk feature added a layer of friendliness, making interactions enjoyable and engaging.

5.2 Challenges

Despite its strengths, the chatbot faced challenges with handling ambiguous inputs. For instance, commands missing specific details, such as "Add some apples," often required additional processing to infer intent. Additionally, the explicit confirmations, while helpful in low-confidence scenarios, occasionally felt repetitive for users performing frequent tasks. Furthermore, it felt as if the confidence threshold could be tweaked a bit more since sometimes it would result in the wrong intent being matched.

5.3 Future Enhancements

To address challenges and improve user experience, several enhancements could be made to the chatbot. Integrating more advanced natural language processing techniques, such as spaCy or pre-trained transformer models, would allow it to handle complex and ambiguous inputs more effectively. For instance, commands like "Add whatever’s missing for making pancakes" could trigger a recipe analysis feature to suggest or add the necessary ingredients to the inventory. Adding voice input support would further enhance accessibility, allowing users to issue commands hands-free.

Expanding the chatbot’s functionality into broader kitchen tasks could make it a more comprehensive assistant. For example, meal planning could allow the chatbot to suggest recipes based on available ingredients, while grocery list management could generate shopping lists or integrate with online grocery services for direct ordering. Features like calorie tracking and expiration notifications would provide additional value, reminding users of expiring items or offering health insights about their inventory. Organising inventory by categories, such as pantry or freezer, would also improve usability, allowing users to quickly retrieve relevant information.

Finally, introducing personalised suggestions based on user behaviour could make the chatbot more adaptive, such as recommending frequently used items during restocking. These enhancements would transform the chatbot into a versatile tool, combining inventory management with meal planning and other kitchen-centric features to create a truly robust user experience.

6 CONCLUSION

The chatbot developed in this project demonstrates how natural language processing and conversational design can simplify kitchen inventory management. Through features such as inventory tracking, adding and removing items, and robust error handling, the chatbot offers a user-friendly and efficient solution for managing supplies. Testing showed strong performance, with high task completion rates and effective intent detection, while user feedback highlighted its ease of use and engaging interaction style.

However, challenges like handling ambiguous inputs and low-confidence scenarios revealed areas for improvement. Future enhancements, such as integrating advanced NLP techniques, voice input, and expanded kitchen-related features like meal planning, would elevate the chatbot from a task-oriented tool to a comprehensive kitchen assistant. With these developments, the chatbot could significantly enhance the way users interact with and manage their kitchens.

REFERENCES

1. Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly Media, Inc.
2. SpaCy. (2024). Industrial-Strength Natural Language Processing in Python. Retrieved from https://spacy.io
3. scikit-learn Developers. (2024). *scikit-learn: Machine Learning in Python*. Retrieved from https://scikit-learn.org
4. Diagrams.net. (2024). Online Diagram Software and Flowchart Maker. Retrieved from https://www.diagrams.net
5. OpenAI. (2024). *Transformer Models for NLP*. Retrieved from https://openai.com