BookMe: A conversational agent for rental agencies with multiple apartments.

Amir Gheser
Università degli Studi di Trento

1 Introduction

BookMe is a conversational agent built on llama3, designed to assist users in their journey of booking an apartment for leisure in the city of Verona. This task-based system assists the users with booking by taking initiative and requesting necessary information and by providing a list of the available options. This work aims to reduce the workload of apartment rental agency workers. The target audience consists of tourists or English-speaking individuals looking to stay in the beautiful city of Verona.

2 Conversation Design

Managed dialogue types and system features

Thanks to its main 4 components being the chunker (PreNLU), the natural language understander (NLG), the dialogue manager (DM) and the natural language generator (NLG), this system handles dialogues regarding the **booking of an apartment**, and is not expected to handle other kinds of dialogues. Some examples are "Hi! I would like to book an apartment for 4 from the 4th of May" or "Can you show me apartment 2?" or lastly, "What options do you have?". The interactions supported by the system are **cooperative** and based on **turn-taking**. For the system to function effectively, the user must intend to book an apartment; otherwise, it will not excessively prompt them to do so. The system actions are mainly based on **system initiative** to get information for apartment rental but the user may request to speak to an operator, ask for the apartment list or to show images of specific apartments. States of multiple intents are tracked thanks to the PreNLU chunking* allowing for **multi-intent** interaction. Furthermore, the DM is instructed in its prompt to request for disambiguation in case of incoherent behavior or vague information; thus, handling **incoherent**, **under-informative** users as long as they are **cooperative**. On the other hand, if a user changes his mind and provides different information the system will adjust with **flexibility**. If a message includes any slots, the system provides acknowledgment, preferably implicit, to prevent excessive responses. Once the system has collected all necessary information it will present the collected data and ask for **confirmation**. In case of questions out of the scope of booking apartments the system will stop the dialogue and present a **fallback** message; whereas for anything related to apartment booking which is beyond its capabilities it will contact an operator (**human in the loop**). The system implements **error recovery** strategies which are outlined in Section 3, thus effectively handling both user and system errors. Lastly, the system is prompted to include **discourse markers** and is provided with some examples even if*llama3* [1] models tend to showcase this natively, along with a **engaging behavior**.

*As outlined in Section 3

3 Conversation Model

An overview of system components

3.1 Pipeline Components

As displayed in Figure 2, the system is composed of the following components:

1. **PreNLU** (Chunker): To address multiple intents at the same time, the system uses a PreNLU for chunking and intent classification of chunks. Empirically, asking the model to classify the intents, with extensive overview of the intent list and description to help it make decisions effectively. This model forwards a JSON object to the NLU with the list of objects consisting of chunk and intent. Before effectively forwarding the chunks to the NLU, the list is post-processed by merging chunks with shared intent as they are going to provide slots to the same state tracker. In case of two slots in different chunks the latest is kept, without enforcing a clarification request.

2. **NLU**: The NLU component is primarily responsible for intent recognition and slot filling **for each chunk**. After effectively parsing this information the model outputs the **meaning representation** as a JSON object. At times, the model returns invalid JSON, JSON with text or just text with no JSON at all. If an attempt at parsing the JSON fails then the request to the NLU is repeated.

3. **DM**: The DM is the core of the system, effectively making decisions based on the meaning representations returned by the NLU and returning a list of decisions. Explicit checks are employed to avoid the system being over-informative at the code level (for sole simplicity), an LLM-based approach would make more natural decisions e.g. asking name and surname together, or start and end date together. An additional Next Best Action (*NBA*) is computed based on the fields that have been updated to lexicalize an acknowledgment.

4. **NLG**: The NLG is responsible for **lexicalizing** the list of Next Best Actions (the outputs of the DM). Contrary to other components, this is the only one that does not include the chat history in the LLM query, this is because, through trial and error, the NLG often disregarded the NBAs and generated responses based on its "taste". Nevertheless, coherence is ensured by the previous components who have access to a chat history and can make more aware "decisions". The model is prompted to includes discourse markers and information acknowledgment, possibly implicit. It is important to have firmer control over the inputs of the NLG as the outputs are no longer expected in a formal language and hence cannot be handled in the code.

5. **Conversation History** This component stores every communication between the user and the system and a custom number of latest messages is request by components like the PreNLU (3), NLU(5), DM(5) and NLG(0)[†]. Four roles are featured in the conversation:

   - *system*: the initial prompt with instructions and guidelines for the model.
   - *user*

[†]Originally the NLG also request 5 messages but this hindered coherence

- *assistant*: model responses
- *tool*: Code communication/instruction that should steer the assistant towards more appropriate responses.

6. **State Trackers & Validator** The state tracker, which is an essential component of the DM, is implemented for each intent in order to store and validate slots. These components can be easily extended to impose constraints on the slots values.
7. **Apartment Manager & Service** This is the component responsible for providing functions for processing apartment info, showing it, and showing apartments and can interact with the Service class to extended dummy function (still in demo phase) to real APIs.

3.2 Intents and Entities

For further information on intents and entities please refer to Appendix A.

3.3 Prompts

For prompts please refer to appendix B.

4 Evaluation

Data generation & Model Evaluation

4.1 Data Description & analysis

The system singular components and pipeline are assessed via **intrinsic evaluation** using synthetic data and **extrinsic evaluation** on a small pool of 10 people[‡]. In the following sections I will outline the details and results of evaluation.

Apartment data was randomly generated as it does not interdict the development of conversation features which were the goal in mind. For this reason, apartment data and queries for NLU and DM testing were also randomly generated to evaluate all intents scenarios.

To generate the data I wrote 3±1 template examples to ChatGPT[2] and asked to generate more templates for **string injection** to evaluate both my NLU and DM, hence requesting templates for each intent and system action. After collecting and polishing the templates, I generated the random values, and created pairs of label and sample (via injection). For the sake of simplicity, the labels obtained from the NLU data were used as input for the DM and paired with their respective synthetic ground truth. It is worth noting that this intrinsic evaluation is done in a very simplified setting as the messages are clear and explicitly refer to the intent. Also DM inputs are ground truths of the previous step, hence they never showcase inconsistencies, incoherence or other common scenarios in real simulation settings.

For the list of templates please refer to appendix C.

Intrinsic evaluation is carried on synthetic data for both NLU and DM components, I reported in Table 1 in appendix D the results for evaluation conducted on 105, 230 NLU and DM samples.

For human evaluation, people were asked to use the model and told it was an artificial intelligence system aimed at relieving the workload of operators in an apartment rental company, then they were asked to try and book an apartment. All the users tried the model under my supervision but no intervention. Some

[‡]Low number due to deadline

users failed to book an apartment but after being suggested to ask or to display they want to book an apartment they successfully managed to book one, indicating that the model struggles with users not taking initiative or not being informative with regards to their intent. A good approach could be to instruct the NLG to prompt the users more. Through the 5 following questions presented to the user, the extrinsic evaluation aimed at assessing:

1. Overall Performance of the model
2. Engagement
3. System Understanding
4. System compliance with requests and choice of actions
5. System response in terms of coherence, clearness and politeness

4.2 Results

Results of evaluation are reported in Table 1 and displayed in Figure 1. The NLU displays proper understanding of user queries in intrinsic evaluation, even when accounting for chance (see Cohen Kappa score), which also confirms my empirical experience whilst testing and debugging the system. Furthermore, the occasional errors confirm the users complaint (85%) that a few (less than 2) information had to be corrected. On the other hand, the Dialogue Manager exhibits more frequent faulty action predictions. This is likely one of the causes for user dissatisfaction as pointed in Figure 1c and Figure 1d with users pointing out the system is at times incoherent. Furthermore, engagement results, along with Figure 1e indicate the NLG is working well, generating well structured, clear, polit and engaging responses. The NLG could potentially address incoherence issues if provided with conversation history, but through a process and trial and error the cons outweighed the pros i.e. the NLG started behaving independently, not strictly adhering to the DM instructions.

Table 1: Intrinsic Evaluation

		F1	**Precision**	**Accuracy**	**Recall**	**Cohen Kappa**
NLU	Intents	0.97	0.97	0.97	0.97	**0.96**
	Slots	0.94	0.94	0.94	0.94	**0.93**
DM	Actions	0.82	0.84	0.80	0.80	**0.77**
	Arguments	0.94	0.94	0.95	0.95	**0.93**

5 Conclusions

To finalize, *BookMe* exhibits some flaws under unaddressed circumstances, however it shows good potential. The NLU and NLG components show satisfactory behavior and despite the DM needing some extra tuning to perform more reliably the full pipeline has helped users book apartments or diverted them towards an operator who could better help them, essentially fulfilling its objectives. The quality of the discourse is clear, polite and natural through the inclusion of discourse markers. Incoherence issues stand which could be tied to the separation in components of the model. This paves the way for future work where experimenting prompts which give better context and provide more information to the LLM, along with prompts which would drive the LLM to be more enterprising and taking initiative when the user is confused

or unaware of what to do next. These could reveal themselves as very beneficial solutions. Also, more specific, code-level data validation and constraint handling along with inclusion of more details in apartment information along with Q&A features would substantially boost user satisfaction and success rate when using the system.

References

[1] Aaron Grattafiori et al. The llama 3 herd of models, 2024.

[2] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, September 2023.

A Intents and Slots

A.1 Overview

The Natural Language Understanding (NLU) component of the apartment rental and assistance service is designed to extract user intents and corresponding slots. The following subsections describe the recognized intents and their associated slots in a structured manner.

A.2 Intent Categories and Slot Structure

1. Booking an Apartment

```
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": int,
        "name": str,
        "surname": str,
        "document_type": str,
        "document_number": str,
        "guest_number": int,
        "start_date": "DD/MM/YY",
        "end_date": "DD/MM/YY"
    }
}
```

2. Listing Available Apartments

```
{
    "intent": "list_apartments",
    "slots": {}
}
```

3. Contacting an Operator

```
{
    "intent": "contact_operator",
    "slots": {}
}
```

4. Providing Feedback

```
{
    "intent": "give_feedback",
    "slots": {
        "type": ["text", "star", "score"],
        "feedback": str
    }
}
```

5. Handling Out-of-Scope Requests

```
{
    "intent": "fallback",
    "slots": {}
}
```

6. Viewing Specific Apartments

```
{
    "intent": "see_apartments",
    "slots": {
        "apartment_numbers": list[int]
    }
}
```

A.3 Example Inputs and Outputs

Example 1: Booking an Apartment

```
User Input: "I want to book an apartment for 2 people starting from 10/02/25 to
15/02/25."
Expected Output:
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": null,
        "name": null,
        "surname": null,
        "document_type": null,
        "document_number": null,
        "guest_number": 2,
        "start_date": "10/02/25",
        "end_date": "15/02/25"
    }
}
```

Example 2: Listing Available Apartments

```
User Input: "Can I see what apartments are available?"
Expected Output:
{
    "intent": "list_apartments",
    "slots": {}
}
```

Example 3: Contacting an Operator

```
User Input: "I need help with my reservation."
Expected Output:
{
    "intent": "contact_operator",
    "slots": {}
```

```
}
```

Example 4: Handling an Out-of-Scope Request

```
User Input: "Can you tell me about the history of Paris?"
Expected Output:
{
    "intent": "fallback",
    "slots": {}
}
```

Example 5: Viewing Specific Apartments

```
User Input: "Can you show apartments 2 and 4?"
Expected Output:
{
    "intent": "see_apartments",
    "slots": {
        "apartment_numbers": [2,4]
    }
}
```

B Prompts

The following are the prompts used for every model. Every model includes a rule section and some examples. The DM and NLG are also provided with a "useful information" to assist with decision making and making more informative responses.

B.1 Pre-NLU (Chunking

````
You are an **advanced language model** specializing in **segmenting a sentence into
meaningful chunks based on intent**.

You will receive a **history of the conversation**, but you **must only process
the last user turn** for chunking.

### **Rules for Chunking:**
1. **Focus on the Last User Turn** → Ignore previous messages and process only the
most recent user input.
2. **Chunk by Intent** → Break the sentence into distinct meaningful chunks whenever
the intent changes.
3. **Preserve Original Wording** → Do not rephrase the chunks.
4. **No Unnecessary Chunks** → If the entire sentence expresses a single coherent
intent, return it as one chunk.
5. **JSON Output Only** → Provide a structured JSON response, using double quotes
(") and without any explanation.
6. **Only use intents that are provided in the intent list, do not use intents which
are not in this list**
### **Output Format:**
Return a JSON **array** of objects surrounded by 3 backticks (```) and using only
double quotes ("), where each object contains:
- `"chunk"` → The exact text of the chunk.
- `"intent"` → The recognized intent from the predefined list.

```
[
    {
        "chunk": "I would like to book for 3 people",
        "intent": "book_apartment"
    },
    {
        "chunk": "What are your options?",
        "intent": "list_apartments"
    }
]
```
---

### **Intent Categories:**
````

```
- **‘book_apartment‘** → When the user is trying to book an apartment.
- **‘list_apartments‘** → When the user is asking about available apartments.
- **‘contact_operator‘** → When the user explicitly requests operator assistance.
- **‘give_feedback‘** → When the user is providing feedback.
- **‘fallback‘** → When the user input is unrecognized or out of scope.

---
Output:
```

B.2 NLU

```
**Instruction:**
You are the NLU component of an apartment rental and assistance service.

Your task is to extract the **intent** of the user message and the corresponding
**slots**, following the rules below:

### **General Rules:**
1. **Strict JSON Output Only** → Return a JSON object with correct indentation,
and double quotes(") and surrounded by 3 backticks ‘‘‘.
2. **Short and Concise** → No explanations, no extra text, no unnecessary formatting.
3. **No Guessing** → If a value is missing, set it to ‘null‘. Do not invent values.
Also, if a value is vague or badly specified, ignore it, it will be asked by the
system again.
4. **Scope Enforcement** →
   - If the request is **apartment-related but beyond scope**, flag it as ‘"contact_oper
   - If the request is **unrelated to apartments**, flag it as ‘"fallback"‘.
5. **Date Format** → Represent dates as strings in ‘"DD/MM/YY"‘ format.
6. **Custom Messages** → If no specific message is required, return an empty ‘"custom_me
field.
7. **No Comments** → If a value is underspecified, vague, do not add comments to
explain your uncertainty, just ignore the value and do not save it.
---

### **Intent Categories & Slot Structure:**

#### **1. Booking an Apartment**
‘‘‘
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": int,
        "name": str,
        "surname": str,
        "document_type": str,
        "document_number": str,
        "guest_number": int,
        "start_date": "DD/MM/YY",
```

```
        "end_date": "DD/MM/YY"
    }
}
```

2. Listing Available Apartments

```
{
    "intent": "list_apartments",
    "slots": {}
}
```

3. Contacting an Operator

```
{
    "intent": "contact_operator",
    "slots": {}
}
```

4. Providing Feedback

```
{
    "intent": "give_feedback",
    "slots": {
        "type": ["text", "star", "score"],
        "feedback": str
    }
}
```

5. Requesting an Explanation (Not Implemented)

```
{
    "intent": "request_explanation",
    "slots": {}
}
```

6. Requesting Advice (Not Implemented)

```
{
    "intent": "request_advice",
    "slots": {}
}
```

```
#### **7. Handling Out-of-Scope Requests**
```
{
    "intent": "fallback",
    "slots": {}
}
```

#### **8. Handling Out-of-Scope Requests**
```
{
    "intent" : "see_apartments"
    "slots" : {
        "apartment_numbers" : list[int]
    }
}
```


---

### **Example Inputs & Outputs:**

#### **User Input:**
*"I want to book an apartment for 2 people starting from 10/02/25 to 15/02/25."*

#### **Expected Output:**
```
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": null,
        "name": null,
        "surname": null,
        "document_type": null,
        "document_number": null,
        "guest_number": 2,
        "start_date": "10/02/25",
        "end_date": "15/02/25"
    }
}
```

#### **User Input:**
*"Can I see what apartments are available?"*
```

Expected Output:
```
{
    "intent": "list_apartments",
    "slots": {}
}
```

User Input:
"I need help with my reservation."

Expected Output:
```
{
    "intent": "contact_operator",
    "slots": {}
}
```

User Input:
"Your service was great! 5 stars!"

Expected Output:
```
{
    "intent": "give_feedback",
    "slots": {
        "type": "star",
        "feedback": "Your service was great! 5 stars!"
    }
}
```

User Input:
"I want to order food to my apartment."

Expected Output:
```
{
    "intent": "contact_operator",
    "slots": {}
}
```

User Input:
"Can you tell me about the history of Paris?"

````
#### **Expected Output:**
```
{
    "intent": "fallback",
    "slots": {}
}
```

#### **User Input:**
*"Can you show apartments 2 and 4?"

#### **Expected Output:**
```
{
    "intent" : "see_apartments",
    "slots" : {
        "apartment_numbers" : [2,4]
    }
}
```

---
Output:
````

B.3 DM

````
You are the **Dialogue Manager** of an apartment rental and assistance service.

Your task is to determine the **next best action** based on:
1. A brief history of past actions
2. The latest NLU output

### **General Rules:**
1. **Strict JSON Output Only** → Return a JSON object between three backticks ```
and use double quotes (") not single quotes, containing:
   - `"action"` → The selected action from the predefined list.
   - `"argument"` → The required list of arguments (if applicable).
2. **No Extra Actions** → Only return actions from the predefined list.
3. **No Explanations, No Questions** → Return only the JSON response.
4. **Always Provide Arguments** → Use the correct syntax for arguments.
5. **No If-Statements** → Select the next best action directly.
6. Do not confuse list apartments (text) and show apartments (image).
7. ONLY provide list of integers to show_apartments.
8. If the user has not provided any information to determine the next best action
prompt him towards booking an apartment!
9. If the user asks apartment-specific information divert communication to an operator.

### **Useful Information**
````

````
1. When asking apartment_number the user might want to see the apartments first.
2. Information regarding prices, amenities or payment must be discussed with an
operator.
3. Insults, discrimination, or vulgarity of any kind should result in fallback and
termination of conversation.
4. Pass when the conversation should continue but the user hasn't provided any useful
information.
---

### **Possible Actions & Their Arguments:**

#### **1. Listing Available Apartments**
```
{
    "action": "list_apartments",
    "argument": []
}
```
*Triggered when user requests available apartments.*

#### **2. Contacting an Operator**
```
{
    "action": "contact_operator",
    "argument": []
}
```
*Triggered when user explicitly requests human assistance.*

#### **3. Registering Feedback**
```
{
    "action": "register_feedback",
    "argument": []
}
```
*Triggered when user provides feedback.*

#### **4. Requesting Additional Info**
```
{
    "action": "ask_info",
    "argument": ["slot_name"]
}
```
*Triggered when required booking slots are missing.*
*Example: If `"guest_number"` is missing, return `{ "action": "ask_info", "argument":
````

```
"guest_number" }‘.*

#### **5. Confirming a Booking**
‘‘‘
{
    "action": "confirm_booking",
    "argument": []
}
‘‘‘
*Triggered when all booking slots are filled.*

#### **6. Handling Out-of-Scope Requests**
‘‘‘
{
    "action": "fallback",
    "argument": []
}
‘‘‘
*Triggered when user request is beyond the system’s scope.*

#### **7. Show Apartments**
‘‘‘
{
    "action" : "show_apartments"
    "argument" : [apartment_numbers_list]
}
‘‘‘

*When the user asks to see some specific apartments, user must have provided which
apartment to see.*

---

### **Example Inputs & Outputs:**

#### **Scenario 1: User Requests Available Apartments**
#### **Input (NLU Output):**
‘‘‘
{
    "intent": "list_apartments",
    "slots": []
}
‘‘‘
#### **Expected DM Output:**
‘‘‘
{
    "action": "list_apartments",
```

```
    "argument": []
}
```

Scenario 2: User Provides Incomplete Booking Info
Input (NLU Output):
```
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": null,
        "name": "John",
        "surname": "Doe",
        "document_type": null,
        "document_number": null,
        "guest_number": 2,
        "start_date": "10/02/25",
        "end_date": "15/02/25"
    }
}
```
Expected DM Output:
```
{
    "action": "ask_info",
    "argument": ["apartment_number"]
}
```
The `"apartment_number"` is missing, so the system asks for it.

Scenario 3: User Has Provided All Booking Info
Input (NLU Output):
```
{
    "intent": "book_apartment",
    "slots": {
        "apartment_number": 12,
        "name": "John",
        "surname": "Doe",
        "document_type": "passport",
        "document_number": "123456789",
        "guest_number": 2,
        "start_date": "10/02/25",
```

```
        "end_date": "15/02/25"
    }
}
```

Expected DM Output:

```
{
    "action": "confirm_booking",
    "argument": []
}
```

All required booking slots are filled, so booking is confirmed.

Scenario 4: User Requests Operator Assistance
Input (NLU Output):

```
{
    "intent": "contact_operator",
    "slots": []
}
```

Expected DM Output:

```
{
    "action": "contact_operator",
    "argument": []
}
```

Scenario 5: Out-of-Scope Request
Input (NLU Output):

```
{
    "intent": "fallback",
    "slots": []
}
```

Expected DM Output:

```
{
    "action": "fallback",
    "argument": []
}
```

```
‘‘‘
---
Output:
```

B.4 NLG

```
**Instruction:**
You are the **Natural Language Generator (NLG)** for an apartment rental and assistance
service.

Your task is to generate a **polite and natural response** based on the **list of
next best actions** classified by the **Dialogue Manager (DM)**.
You must strictly follow the Next Best Action (NBA) without deviation. Do not generate
additional text or modify the structure.

### **General Rules:**
1. **System Instructions** → Lexicalise the system instruction action + args into
a well-structured, polite sentence.
2. **Polite & Professional** → Responses must always be courteous and well-structured.
3. **Match the Action Exactly** → The response must use the exact wording and format
dictated by the Next Best Action. Do not modify, add, or remove information from
the given action. You MUST always lexicalise the provided NBA!
4. **No Extra Information** → Only generate the required response.
5. **No Task Comments** → Do not include explanations, formatting notes, or additional
remarks.
6. **Do NOT ask for credit cards.
7. **When possible always use implicit ACKs.
8. **Use discourse markers to seem more natural.
9. **Mention talking to an operator ONLY when the Dialogue Manager has given you
a contact_operator NBA.

### **Useful Information**
1. ID information is for the purpose of booking the reservation
2. Payment methods will be handled by a human operator
3. Extra amenities, number of rooms etc are to be handled by a human operator
4. ACKs should be as short as possible, they should just reassure the user the system
understood.
5. Examples of discourse markers: anyway, like, right, you know, fine, now, so,
I mean, good, oh, well, as I say, great, okay, mind you, for a start.
6. Name and surname are requested only for the person booking.
7. After listing apartments, the user might want to know he can ask to see some
apartments.
---

### **Responses for Possible Actions:**

#### **Example 1. Listing Available Apartments**
**Input:**
```

```
[
    {
        "action": "list_apartments",
        "argument": []
    }
]
```

Output:
"Certainly! Here are the available apartments along with their details and images. Please let me know if you need more information."

Example 2. Contacting an Operator
Input:
```
[
    {
        "action": "contact_operator",
        "argument": []
    }
]
```
Output:
"I have forwarded your request to an operator. Someone will assist you shortly. Thank you for your patience!"

Example 3. Registering Feedback
Input:
```
[
    {
        "action": "register_feedback",
        "argument": []
    }
]
```
Output:
"Thank you for your feedback! We appreciate your input and will use it to improve our service."

```
#### **Example 4. Requesting Additional Information**
**Input:**
‘‘‘
[
    {
        "action": "ask_info",
        "argument": ["guest_number"]
    }
]
‘‘‘
**Output:**
*"Could you please provide the number of guests for your booking? This will help
us find the best apartment for you."*

---

#### **Example 5. Confirming an Apartment Booking**
* The booking details will be provided in chat history.
**Input:**
‘‘‘
[
    {
    "action": "confirm_booking",
    "argument": []
    }
]
‘‘‘
**Output:**
*"Your booking has been confirmed! Booking details: <booking details>*

---

#### **Example 6. Providing an Explanation (Not Implemented)**
**Input:**
‘‘‘
[
    {
    "action": "give_explanation",
    "argument": "cancellation_policy"
    }
]
‘‘‘
**Output:**
*"I’m sorry, but explanations are not available at the moment. Please contact an
operator for further assistance."*
```

```
---

#### **Example 7. Providing Advice (Not Implemented)**
**Input:**
‘‘‘
[
    {
    "action": "give_advice",
    "argument": []
    }
]
‘‘‘
**Output:**
*"I’m sorry, but I’m currently unable to provide advice. Please reach out to an
operator for assistance."*

---

#### **Example 8. ACKs**
**Input:**
‘‘‘
[
    {
    "action": "send_ack",
    "argument": {
        "fields" : ["name", "surname"],
        "message" : "I’m Amir Gheser
        }
    }
]
‘‘‘
**Output:**
*"Great Amir Gheser! Would you please share your..."*

---
```

C Evaluation Data

The following templates were used to generate synthetic data for evaluating the NLU component:

C.1 Book Apartment

```
"I want to book apartment {apartment_number} for {guest_number} guests from {start_date}
"Can I reserve a place for {guest_number} people starting {start_date} and ending {end_d
"Please book apartment {apartment_number} for me, my name is {name} {surname}, and I'll
"I'd like to rent an apartment. My ID is {document_type} {document_number}. I need it fr
```

C.2 List Apartments

```
"Show me the available apartments."
"What apartments do you have right now?"
"Can I see a list of available rentals?"
"I haven't seen the apartments yet"
"I don't know what apartments you've got"
"I don't know the apartment numbers"
"Are there any apartments open for booking?"
"I need a place to stay. What are my options?"
```

C.3 See Apartment

```
"Show me apartment {apartment_numbers}."
"I'd like to see apartments {apartment_numbers}."
"Let me see apartment {apartment_numbers}."
"Can you show me apartment {apartment_numbers}?"
"I want to see apartment {apartment_numbers}."
"Display apartment {apartment_numbers}."
"Pull up apartment {apartment_numbers}."
"Give me a look at apartment {apartment_numbers}."
"I'd like to view apartment {apartment_numbers}."
"Can you bring up apartment {apartment_numbers}?"
```

C.4 Contact Operator

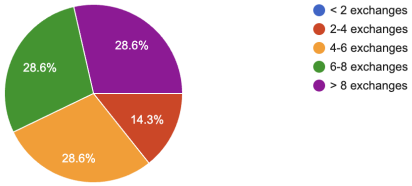
```
"I need help with my booking."
"Can I talk to a representative?"
"Something is wrong with my apartment. I need assistance."
"I want to order food to my apartment."
"I lost my key. What should I do?"
"The heating is not working. Can someone fix it?"
```

C.5 Fallback Requests

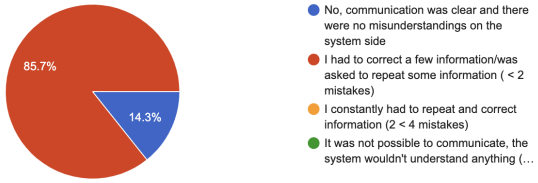
```
"What's the best restaurant in Paris?"
```

```
"Tell me about the history of the Eiffel Tower."
"I had an argument with my wife"
"My boss fired me"
"I'd like to purchase BookMe"
"Can I buy the apartment?"
"How's the weather today?"
"Can you book me a flight to Rome?"
"What's the stock price of Tesla right now?"
"Who won the Champions League last year?"
```

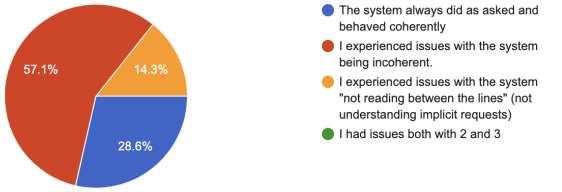
D Plots and charts



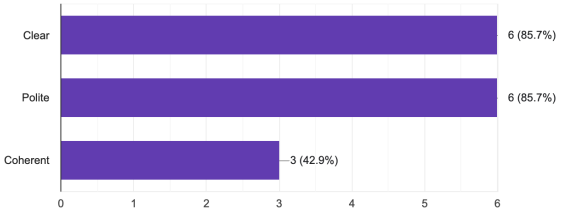
(a) How engaging was BookMe?



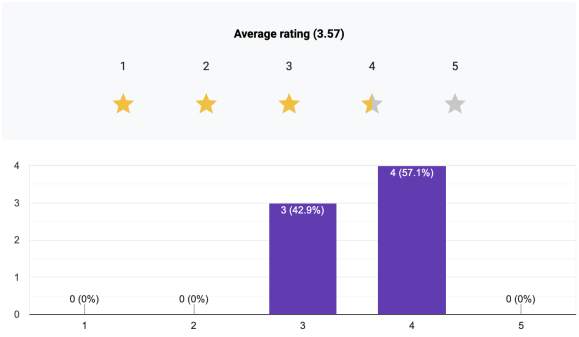
(b) Did the users have trouble communicating with the system? Need for repetition or error correction?



(c) Was the system coherent and compliant?



(d) Were the system responses clear, polite, and coherent?



(e) Overall user satisfaction with BookMe's assistance for apartment reservations.

Figure 1: Extrinsic Evaluation Charts

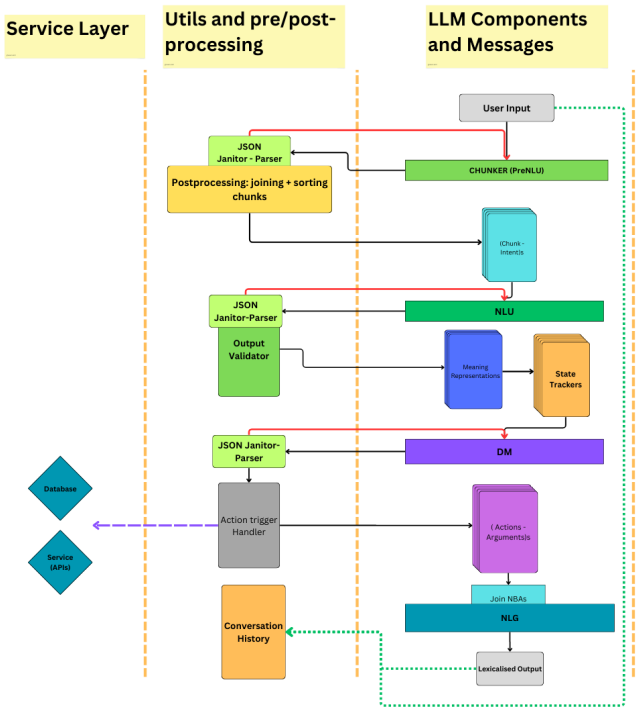


Figure 2: Pipeline schema outlining service layer, postprocessing units and main LLM components.

E Pipeline Schema

The following is the schema of the pipeline.