LLAMusica - Human Machine Dialogue Project Report

Stefano Bonetto (247179)

University of Trento

stefano.bonetto@studenti.unitn.it

1. Introduction

Music is more than just sound. It's an experience that connects listeners with emotions, memories, and cultural narratives. Enthusiasts often desire more than just passive listening, they wants a deeper understanding of the music they enjoy.
LLAMusica is an intelligent music assistant designed for these passionate listeners who seek real-time, in-depth insights into their favorite tracks, albums, and artists. By integrating with Spotify's APIs, LLAMusica delivers accurate and up-to-date music-related information, analyzes user listening habits, and provides genre-based recommendations, enriching the listening experience through contextual details and continuous discovery.

1.1. Dialogue System Description

This project is powered by `LLaMa 3.2`[1], a state-of-the-art large language model (LLM) that provides advanced natural language understanding and generation capabilities.
Specifically, I have chosen to use the `ollama` framework[2], which provides an efficient solution for querying the LLM in a relatively faster and streamlined manner compared to other alternatives.
LLAMusica seamlessly integrates with Spotify's API[3] to retrieve relevant music data, ensuring that users receive real-time and contextually rich responses.
However, despite the extensive database provided by Spotify, its APIs has certain limitations in the depth of information it exposes compared to the mobile application. Some interesting metadata (such as composer and songwriter credits, detailed per-track listener counts, and additional album and tracks insights) remain inaccessible through programmatic queries. This creates a discrepancy between the information available to Spotify users and the data retrievable by external applications.

1.2. Type of information it can retrieve

Currently, LLAMusica can provide answers based on a well-defined set of musical attributes, categorized as follows:

- **Tracks**: LLAMusica is able to retrieve a track's popularity score, release date, duration, its associated album, its artist, and genre classification.
- **Artists**: it can extract an artist's total number of followers, popularity rating, and primary music genres.
- **Albums**: it can provides details such as the album's genre, release date, and total number of tracks.
- **User Habits**: The system can display user's top artists and tracks based on their listening history over different time frames, including the past month, the last six months, and the past year.
- **Recommendations**: given a seed genre, some suggestions can be given to the user.[4]

Beyond these fundamental attributes, LLAMusica is designed with scalability in mind. As Spotify expands its API capabilities and provides access to richer metadata, the system will evolve accordingly.

2. Conversation Design

The system is designed to sustain continuous dialogues within the same query while also identifying new queries that deviate from the previous one. This functionality is enabled by the Change of Topic (CoT) detector, as illustrated in Figure 1.
The system is structured around three principal categories of topics:

- **Music Information:** queries related to specific tracks, artists, or albums, including details such as release dates, genres, followers, etc...
- **User Preferences:** retrieval of personalized listening data, such as the user's top tracks or top artists over different time frames.
- **Music Recommendations:** suggestions based on a seed genre (e.g. pop, rock, etc...).

Users can freely switch between different topics, but once a topic is completed, the system deliberately clears its memory of that conversation.
This decision is based on practical observations: retaining past interactions often introduced bias, making it harder for the system to fully focus on new queries. When previous conversations were remembered, the system tended to associate new questions with past ones rather than treating them as independent requests. This sometimes led to confusion, especially when the user wanted to shift to a completely different topic.
To prevent this, every new query is processed as a standalone request. This ensures that the system remains objective and accurately interprets the user's intent. Even if a user asks the same question again at a later time, the system will treat it as a fresh request, unaffected by prior interactions.
While this approach may seems like a limitation, it is not a significant drawback in my use case. If a user repeats a query at different times, it is likely because they genuinely want to retrieve that information again, rather than expecting the system to recall previous conversations.
LLAMusica's pipeline is structured as follows:

1. The user initiates interactions by providing input.
2. The system takes initiative when information is missing, prompting the user to clarify or provide details (handled by the "`request_info`" action in DM).

[1] https://arxiv.org/pdf/2407.21783
[2] https://github.com/ollama/ollama
[3] https://github.com/spotipy-dev/spotipy/tree/2.24.0

[4] The idea initially was to use the recommendations of Spotify's API based on given songs, however it doesn't work I opt to use a set of predefined tracks for each genre to suggest to the users.

3. When sufficient data is available (required data to query the Spotify database), the system provides confirmations ("`confirmation`" action) and the NLG component give to the user the answer.
4. The Change of Topic (CoT) component determines whether the ongoing dialogue is related to the previous query. If the conversation remains on the same topic, it updates the state dictionary using the User State Dictionary (USD) component. However, if a new topic is introduced, the system initializes a new state dictionary to ensure proper context management.

Engagement is maintained through responses generated by the NLG component, ensuring the conversation remains interactive and contextually relevant. The system continuously prompts the user with follow-up questions, asking whether they would like to explore more about their current query. This approach effectively preserves the mixed-initiative dialogue framework, keeping the user engaged and actively involved in the conversation.

Even if a user input is out-of-domain the engagement is preserved by providing a polite response, guiding the user back to relevant topics.

The system retains short-term preferences, such as the current query context, but does not store long-term history once a topic is completed, as it does not save past interactions. This approach eliminates bias from previous queries, ensuring that each new question is treated independently. However, this design limits the system's ability to build a detailed user profile over time.

Nevertheless, a long-term profiling of the user's musical habits is achieved through Spotify APIs, which provide insights into the user's top tracks and artists over different time frames.

Finally, the system has a built-in error handling procedure. It starts in the slot extractor where some errors are checked (empty slots, under-informative user, typos, etc...).

However, in the initial stage of the pipeline, the system does not verify whether the requested entity (song, album, or artist) exists. Instead, this verification occurs when the `Ground Knowledge Builder` queries the Spotify APIs. If no matching entity is found, the `Ground Knowledge` remains empty. Consequently, the `NLG` component recognizes this absence and informs the user that the requested song, album, or artist could not be found, while also prompting them to retry with a different input if desired.

3. Conversation Model

LLAMusica's pipeline is built upon a core architecture consisting of three main components: the Natural Language Understanding (NLU) component, the Dialogue Manager (DM), and the Natural Language Generator (NLG).

However, several refinements have been introduced, along with the addition of new components, as illustrated in Figure 1.

The effectiveness of the conversational system relies on well-structured prompts designed to guide the behavior of different components. Each prompt ensures that its module operate efficiently, handling user interactions in a context-aware manner.

3.1. State dictionary

The fundamental entity in the pipeline is the state dictionary, which is a simple structure representing the state of the conversation.

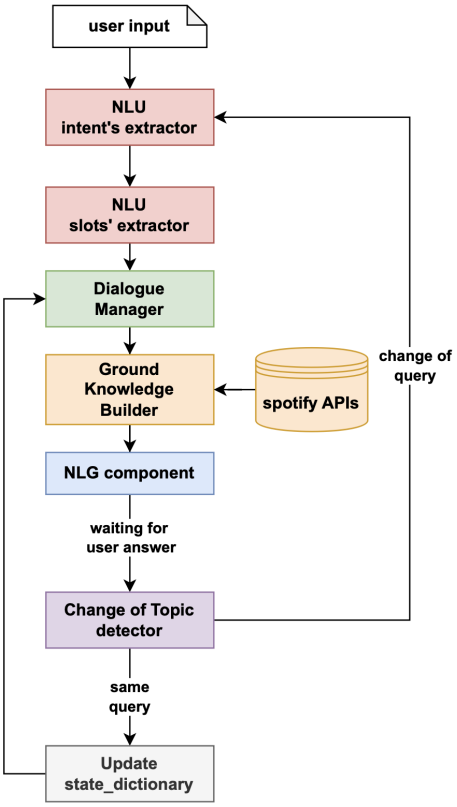

Figure 1: *Complete pipeline for processing user input.*

In particular, in my configuration, I decide to use the state dictionary to collect all the contributions of the different components:

```
{
    "NLU": {
        "<intent1>": {
            "slots": {
                "<slots1>": "<val1>"
                ...
            }
        }
    },
    "DM": [{
            "next_best_action": ...,
            args: {
                ...
            }
        },
        ...
        ],
    "GK": { ... },
    "NLG": "<system_answer>"
}
```

Figure 2: *Structure of the state dictionary.*

3.2. NLU - Intent Extractor

The **intent extraction** component identifies the user intent based on predefined categories reported in the Table 1.

Notice that if a query contains multiple intents of the same type, these are numbered (e.g. `song_info1`, `song_info2`).

Intent categories
song_info
artist_info
album_info
user_top_tracks
user_top_artists
get_recommendations
out_of_domain

Table 1: *Intents recognized by the system.*

3.3. NLU - Slots Extractor

The next component extracts intent-specific slots from the user query, ensuring structured data representation.

Intent	Slots
song_info	song_name, artist_name, details
album_info	album_name, artist_name, details
artist_info	artist_name, details
user_top_tracks	time_frame, limit
user_top_artists	time_frame, limit
get_recommendations	genre, limit
out_of_domain	sentence_intent

Table 2: *Slots recognized by the system.*

The supported slots for each intent are reported in Table 2. Notice that:

- The `details` list can contain one or more of the following elements:
  - **song_info**: ["popularity", "release_date", "duration", "album", "artist", "genres", "all"]
  - **artist_info**: ["followers", "popularity", "genres", "all"]
  - **album_info**: ["genres", "release_date", "total_tracks", "all"]
- time_frame can be "short_term" (last month), "medium_term" (last 6 months), or "long_term" (last year)
- The `genre` slot in **get_recommendations** can have one of the following values: ["pop", "rock", "jazz", "electronic", "hiphop", "r&b", "soul", "classical", "folk", "reggae"].

3.4. Change of Topic (CoT) component

The CoT component analyzes whether the input remains relevant to the previously discussed entity or introduces a new topic:

- If the new user input asks for details about the previously mentioned entity, the system returns `same_query`.
- If the input refers to a completely new entity (e.g., a different song, artist, or topic), the system returns `change_of_query`.

3.5. Dialogue Manager (DM)

The Dialogue Manager determines the next best action based on extracted intents and slots. The system decides whether to confirm the request or ask for missing details

- If required slots are missing, the system requests additional information ("`request_info(intent)`").
- If all necessary slots are provided, the system confirms the request ("`confirmation(intent)`") and retrieves relevant data via Ground Knowledge Builder.

3.6. Ground Knowledge (GK) builder

This component follows the instructions provided by the DM, whether it is `confirmation(intent)` or `request_info(intent)`. If necessary, it retrieves the required information from Spotify.
Once the data is obtained, the component updates the state dictionary, making the information accessible to the NLG component, which then utilizes it to generate a response to the user's query.
Notice that even if the Dialogue Manager (DM) determines the `next_best_action` as `request_info`, the system still retrieve information in some cases. For instance, if the requested slot is `artist_name` for a given song, the General Knowledge (GK) component searches Spotify for the **first associated artist** of that song. This allows the Natural Language Generation (NLG) component to formulate a response asking the user for confirmation, ensuring that the retrieved information aligns with their intended query.

3.7. Natural Language Generator

The Natural Language Generator ensures that system responses are clear, engaging, and strictly based on available information in the Ground Knowledge (GK).

- If `confirmation` is required, the system provides relevant details.
- If `request_info` is needed, it prompts the user to provide missing data.
- If an `out-of-domain` query is detected, the system politely informs the user that it cannot answer.

This component plays a crucial role in maintaining engagement throughout the dialogue. For this reason, prompt strongly recommend that responses conclude with an engaging question related to the current query, also reinforcing the mixed-initiative framework.

3.8. Update Slot Detection (USD) Prompt

The USD module dynamically updates missing slot values based on user input and previous interactions:

- If the system prompts the user to fill a missing slot and the user provides the required information, the module updates the corresponding slot in the state dictionary.
- If the system is confirming information and the user requests additional details about the same entity, the USD module updates the state dictionary by adding the new request to the `details` list of the corresponding intent and the retrieved information used in the previous query.

3.9. Errors in `llama3.2` output.

The primary challenge in managing data flow across these components has been ensuring that `llama3.2` consistently adheres to the specified output formats. Despite explicit formatting instructions embedded within the prompts, the LLM often exhibits variability in its responses, making it difficult to enforce strict structural consistency.

To address this issue, an extensive analysis of `llama3.2`'s outputs was conducted to take into account all possible variations. In particular, the outputs generated by various components have been systematically utilized to "mechanically" build, update, and populate the state dictionary, ensuring coherence across dialogue stages.
By leveraging libraries such as `regex`, I have been able to efficiently process the LLM's diverse output formats, extract relevant information, and seamlessly integrate it into the state dictionary, thereby maintaining structured and accurate data management throughout the pipeline.

4. Evaluation

During the evaluation process, each component (Intent Extractor, Slots Extractor, and Dialogue Manager) was tested independently by providing the correct input, ensuring that the assessment focused intrinsically on the performance of the specific component.

4.1. Intrinsic evaluation

The testing phase was conducted using a test set of 50 user inputs generated with ChatGPT-4o, ensuring variability in the data. These inputs comprehensively cover all application use cases, including single-intent, multi-intent, out-of-domain scenarios and all the seven supported intents.
The results obtained are reported in the Table 3. A prediction is considered correct if it does not interfere with the overall system's functionality, ensuring a seamless and coherent NLG response.

Component	Correct predictions	Accuracy
Intents extractor	43/50	0.86
Slots extractor	41/50	0.82
DM	45/50	0.9

Table 3: *Quantitative validation of different pipeline components.*

The primary weakness of the pipeline lies in the **intent extractor**, as its accuracy directly impacts the entire system. Since there is no mechanism to verify or correct LLaMA's output before passing it downstream, errors in intent extraction propagate through subsequent components without any additional check, potentially leading to incorrect responses.
Despite this limitation, extensive **prompt engineering** has improved its performance, achieving 86% accuracy in intent recognition.
The **Slots Extractor** and **Dialogue Manager (DM)** offer greater control and verification in the next steps.
The **slots extraction** process is validated by ensuring extracted slots conform to the expected format for each intent and verifying that fundamental slot are filled. Since intents follow predefined structures, discrepancies trigger a re-query to LLaMA until a correct response is obtained.
Similarly, the **DM** is validated by checking whether the extracted slots contain null values, ensuring the next-best-action aligns with expected logic (e.g., triggering a `request_info` action when required slott are empty).
However, **semantic errors in individual components remain uncorrectable**, making them the primary source of validation test failures.

4.2. Extrinsic evaluation

For what concerns the overall performance of the system and the user experience, I can confidently say that I am satisfied with the results.
However, it is evident that the system is not flawless. In particular, the Natural Language Generator exhibits a tendency to remain biased toward specific behaviors, which, in certain contexts, may lead to inaccurate or off-topic responses. This issue is particularly noticeable in the final sentences of system responses, which were specifically included to reinforce mixed-initiative dialogue.
For instance, in some cases (one of which is also demonstrated in the presentation video[5]), the system erroneously appends a closing phrase such as:

> *"Would you like to know more about these tracks?"*

even when the ongoing conversation is actually focused on albums or artists. This behavior is likely due to:

- **Confusion in entity tracking** after multiple changes of topic, leading to mismatches between the current conversational focus and the generated response.
- Potential **overfitting** to specific training examples or patterns in the prompt examples, which could be reinforcing the inappropriate response structure.

While these limitations do not significantly hinder the functionality of the system, they highlight areas for improvement.

5. Conclusion

My primary goal was to design a system that encourages mixed-initiative dialogues, ensuring user engagement even when queries are incorrect, ambiguous, or out of context. LLAMusica successfully maintains an interactive conversational flow, guiding the user through the dialogue even in challenging scenarios.
Obviously, there are numerous areas for improvement:

- `ollama` is still slow, especially for complicate queries; exploring more efficient models could enhance responsiveness and improve user experience.
- The **intent extractor** remains the weakest link, as it lacks a verification mechanism. Using a more reliable model could enhance the performances.
- The **NLG component** sometimes generates biased or redundant responses. Incorporating adaptive response mechanisms may improve contextual accuracy.
- **User profilation** could be integrated by leveraging stored user preferences and historical interactions to provide more tailored recommendations.

While LLAMusica has achieved its core objective of enabling structured and engaging conversations around music discovery, addressing these limitations would further enhance its robustness, efficiency, and overall user experience.

[5] https://youtu.be/YRIkrx3f7iA?si=0u4iDPtFt6vnHgoF