Project Report
HeartSync

Pavitra Malhotra
UIN: 634005665

Problem Statement

Dating apps have become an integral part of modern social interaction, yet they lack intelligent systems that truly enhance the user experience. Existing apps often rely on basic filtering and matching algorithms, which fail to incorporate deeper insights into user compatibility. Moreover, users frequently struggle to initiate or sustain meaningful conversations with their matches, leading to frustration and suboptimal user engagement.

The lack of integration with large language models (LLMs) in dating apps creates a gap in both match selection and conversational assistance. LLMs have the potential to revolutionize this space by providing personalized match recommendations and generating tailored conversation suggestions. This project, HeartSync, aims to bridge this gap by leveraging cutting-edge AI techniques to create a smarter, more engaging dating app experience.

Objective

The primary goal of HeartSync is to incorporate LLMs into the matchmaking and communication process to provide users with:

1. **Enhanced Match Selection**: Use LLMs to analyze profiles deeply and recommend matches based on nuanced compatibility metrics, not just basic filters.
2. **Conversation Assistance**: Generate context-aware suggestions for initiating and continuing meaningful conversations with matches.
3. **Searching Profiles Through LLM**: This allows users to enter a search criteria which LLM will follow while searching the profile by giving them a score. Higher score profiles will be recommended to the user.
4. **Personalized Preferences**: Allow users to customize their matchmaking criteria dynamically, using weighted attributes to prioritize what matters most to them.
5. **Streamlined User Experience**: Make the entire process intuitive, reducing the cognitive load for users when finding and interacting with potential matches.

Methodology

Approach

HeartSync utilizes an AI-powered pipeline combining LLMs with classical recommendation techniques:

1. Collect user input during registration, including preferences and personality traits.
2. Employ BM25-based ranking for filtering profiles based on textual similarity and feature alignment.
3. Use an LLM (Ollama - local) for:
   - Compatibility analysis.
   - Generating conversational suggestions tailored to user profiles and chat history.
   - Search profiles using LLM based on criteria given by the user.
4. Enable customization of recommendations via a preference-weighting system.

Data

The dataset for this project is the publicly available **OkCupid Dataset**. This dataset contains rich user profiles, including personal attributes, preferences, and text responses, making it an excellent source for training and testing the matchmaking system.

Data Preprocessing

Based on the code:

- **Filtering and Normalization**:
    - Extracted relevant columns like age, status, orientation, essays, and other attributes.
    - Converted categorical data into tokenized text for BM25-based similarity calculations.
    - Combined multiple essay columns into a single text corpus for holistic profile analysis.
- **Stopword Removal and Tokenization**:
    - Removed stopwords using NLTK.
    - Tokenized text data to prepare it for ranking and matching.
- **Missing Data Handling**:
    - Replaced null values with defaults or placeholders.
    - Serialized list-based fields, such as pets and preferences, into JSON-compatible formats.

Architecture

The system architecture is divided into three layers:

1. **Frontend**:
    - React-based interface for user interaction, including registration, profile viewing, and matchmaking.
2. **Backend**:
    - Flask API to handle user requests, manage database queries, and provide AI-powered functionalities.
3. **AI Layer**:
    - Ollama 3.2 3 billion parameters for LLM-based tasks.
    - BM25 for classical text-ranking of user profiles.

Evaluation Process

- **Profile Matching**:
    - Evaluate how well the LLM recommends matches by manually verifying the relevance of suggested profiles.
- **Conversational Assistance**:
    - Assess the quality of conversation suggestions generated by the LLM during chats.
- **Profile Searching:**
    - Searching the profiles using LLM based on user's criteria

Software Requirements

Backend Requirements

- **Python**:
    - Version: 3.7 or higher
    - **Installation**:
        - Download from Python Official Website.
        - Install and ensure pip is included in the installation.
- **Python Libraries**:
    - Install the required Python packages:

pip install flask flask-sqlalchemy flask-cors flask-socketio requests g4f pandas rank-bm25 nltk numpy

-
    - **Special Notes**:
        - Install NLTK dependencies:

python -m nltk.downloader punkt stopwords

- **SQLite**:
    - Version: Any version compatible with Python.
    - **Installation**:
        - SQLite is bundled with Python. Verify installation:

sqlite3 --version

        - No additional installation is required unless advanced database management is needed.

Frontend Requirements

- **Node.js and npm**:
    - Version: Node.js 14+ and npm 6+.
    - **Installation**:
        - Download from Node.js Official Website.
        - Install and verify
- **React**:
    - Set up React project dependencies using npm

Third-party API Keys

- **Pexels API**:
    - API key required for fetching random profile images.
    - **Setup**:
        - Sign up at Pexels API.
        - Replace the placeholder API key in the PEXELS_API_KEY variable in chat_backend.py.

3. Database Requirements

- **SQLite Databases**:
    - Two SQLite databases are used:
        - users.db: Stores user profile data.
        - conversations.db: Stores user conversations and matches.
    - **Setup**:
        - Ensure both databases are present in the directory specified in ShowProfileServer.py and chat_backend.py.

4. Running the Application

- Start the backend servers:
    - ShowProfileServer.py, chat_backend.py and login_server.py.
- Run the frontend development server (npm start).
- Access the application at http://localhost:3000.

Components

HeartSync is composed of several interrelated components, each playing a vital role in creating an intuitive and AI-driven dating app experience. Below is a breakdown of the components based on the provided code, detailing their functionalities:

Frontend Components

1. **JoinScreen.js**
    - **Purpose**: Handles user registration.
    - **Features**:

        - Collects extensive user details such as username, password, personal attributes, preferences, and essays.
        - Supports image uploads, converting images to Base64 format for backend storage.
        - Uses Axios to send user data to the backend for registration.
        - Performs client-side validation (e.g., required fields, minimum age).
    - **User Flow**:
        - The user fills in the registration form and submits their data.
        - On successful registration, the user is redirected to the preferences screen.
2. **ProfileSearch.js**
    - **Purpose**: Allows users to search for profiles based on custom criteria and displays the results.
    - **Features**:
        - A text input box for entering search criteria.
        - Fetches profiles from the backend using Axios.
        - Displays profile cards with user details, including:
            1. Username
            2. Age
            3. Compatibility score
            4. Score analysis
            5. Bio
        - Includes a circular profile picture for each profile, sourced dynamically from a local folder.
    - **User Flow**:
        - The user enters their search criteria in the text box and submits it.
        - The system fetches matching profiles from the backend and displays them as scrollable cards.
        - Each card includes a circular image and user details.

3. **Login.js**
    - **Purpose**: Manages user login.
    - **Features**:
        - Allows users to log in using their credentials.
        - Displays error messages for invalid credentials.
        - Uses Axios to authenticate the user via the backend.
        - Provides options to navigate to the registration page (JoinScreen.js) or log in to access the main application.
    - **User Flow**:
        - The user inputs their username and password.
        - On successful login, they are redirected to the main dashboard (MainScreen.js).
4. **MainScreen.js**
    - **Purpose**: Acts as the central dashboard for user navigation.
    - **Features**:
        - Provides navigation to key features like profile view, matchmaking, preferences, inbox, and AI-driven matchmaker choices.
        - Resets certain state counters on initial load for a fresh user experience.
    - **User Flow**:

        - The user chooses from the available features (e.g., view profile, show profiles) to proceed to different app sections.

5. **Profile.js**
    - **Purpose**: Displays the user's profile details.
    - **Features**:
        - Fetches and displays detailed user information, including personal attributes and essays.
        - Dynamically loads a random profile image using the backend API.
        - Allows users to view their profile in a scrollable container.
    - **User Flow**:
        - The user views their personal information and bio within the profile page.
6. **ShowProfiles.js**
    - **Purpose**: Provides a Tinder-like swipe interface for browsing profiles.
    - **Features**:
        - Displays detailed profiles fetched from the backend, including personal attributes and bios.
        - Integrates buttons for liking, disliking, and AI-driven compatibility analysis.
        - Uses Axios to send likes/dislikes to the backend and checks for mutual matches.
        - Displays match notifications and fetches random images for profiles.
    - **User Flow**:
        - The user swipes through profiles, likes/dislikes them, or analyzes compatibility using the AI assistant.
7. **Choice.js**
    - **Purpose**: Allows users to customize matchmaking preferences.
    - **Features**:
        - Provides sliders to adjust weights for various attributes (e.g., ethnicity, job, bio).
        - Sends user-defined weights to the backend for preference-based matchmaking.
        - Displays loading indicators and confirmation messages for preference updates.
    - **User Flow**:
        - The user sets weights for preferred attributes and submits preferences to refine their recommendations.

Backend Components

1. **ShowProfileServer.py**

- **Purpose**: Serves as the backend for managing profiles, preferences, and match analysis.
- **Features**:
    - Handles CRUD operations for user data via a SQLite database.
    - Implements APIs for:
        - User registration (/join) and login (/login).
        - Fetching recommended profiles (/profiles).
        - Searching for profiles based on user-provided criteria (/profiles_search/<username>).
        - Analyzing profile compatibility using AI (/analyze_profile).
        - Managing likes (/like), dislikes (/dislike), and matches (/match/<twousers>).
    - Leverages a local LLM (e.g., Ollama) for AI-driven compatibility analysis.
    - Records execution times for profile analysis and searches in log files.
- **Data Flow**:

    - User preferences are filtered and scored using AI for compatibility analysis.
    - Profiles are dynamically ranked based on relevance to the search criteria or user preferences.
    - Matching information, such as likes and mutual matches, is updated in real time.
- **Profile Search Flow**:
    - User enters search criteria on the frontend.
    - Frontend sends the search criteria to the backend via a POST request to /profiles_search/<username>.
    - Backend fetches user profiles and scores them using AI based on strict adherence to the search criteria.
    - Matching profiles with a score greater than 5 are returned to the frontend, along with compatibility analysis.
    - Execution time for the search is logged for performance tracking.
- **Additional Functionalities**:
    - Compatibility Analysis:
        - Profiles are compared and scored using detailed prompts sent to the local LLM.
        - Analysis results include compatibility scores and an explanation in JSON format.
    - Real-time Matchmaking:
        - The /match/<twousers> endpoint checks for mutual likes and, if matched, initiates a conversation in the conversations.db database.
        - Users are notified of successful matches with a predefined message.
    - Logging:
        - Execution times for AI responses and profile searches are logged for debugging and performance optimization.
    - User Interaction:
        - APIs such as /like and /dislike track user interactions and update the database to manage preferences effectively.

2. **login_server.py**
    - **Purpose**: Manages user authentication, registration, preference updates, and profile retrieval.
    - **Features**:
        - Provides endpoints for:
            1. Validating user credentials (/login).
            2. Adding new users with Base64-encoded profile images (/join).
            3. Dynamically updating user preferences (/preference).
            4. Fetching complete user profiles (/user/<username>).
        - Handles image uploads by decoding Base64 data and saving images locally.
        - Interacts with an SQLite database (users.db) for CRUD operations on user data.

    User Flow:
    - Users log in, register, update preferences, or retrieve profile details via dedicated endpoints.
    - The backend processes requests, updates the database, and sends responses to the frontend.

3. **chat_backend.py**
    - **Purpose**: Manages chat interactions and AI-driven conversation suggestions.
    - **Features**:

        - Implements WebSocket communication via Flask-SocketIO for real-time chat.
        - Provides endpoints for:
            1. Saving and retrieving chat messages.
            2. Generating conversation suggestions using Ollama based on chat history and profile details.
            3. Updating feature weights for matchmaking.
        - Filters and ranks profiles based on combined BM25 scores and user-defined weights.
    - **User Flow**:
        - The backend suggests conversational messages tailored to the user's current match and chat context.

Database Components

1. **users.db**
    - **Purpose**: Stores user profiles and preferences.
    - **Schema**:
        - Includes tables for user attributes (e.g., age, status, pets), essays, preferences, and login credentials.
        - Contains serialized JSON fields for list-based preferences (e.g., pets).
    - **Data Flow**:
        - Stores user data during registration.
        - Retrieves data for matchmaking and compatibility analysis.
2. **conversations.db**
    - **Purpose**: Manages chat histories and match interactions.
    - **Schema**:
        - Includes tables for storing user conversations and match notifications.
    - **Data Flow**:
        - Tracks user interactions and conversation timelines.
        - Enables AI to analyze chat contexts for tailored suggestions.

Primary Features

HeartSync incorporates advanced AI functionalities, leveraging Large Language Models (LLMs) to enhance user experience across three core features. These features aim to streamline matchmaking, optimize search results, and foster meaningful interactions between users. Below is an overview of the primary functionalities:

1. Profile Analysis During Browsing

- **Objective**: Provide users with a detailed compatibility analysis while viewing profiles.
- **Functionality**:
    - When a user browses potential matches, HeartSync evaluates each profile using an LLM-powered compatibility scoring system.
    - A detailed analysis is generated, highlighting the strengths and potential alignment of the viewed profile with the user's preferences and traits.
- **Workflow**:
    - The backend sends the current user's and the potential match's profiles to the LLM for evaluation.
    - The LLM assesses attributes like age, interests, personal traits, and lifestyle habits to compute a compatibility score.

    - Results are displayed as a score out of 10 alongside a brief justification for the score, giving users valuable insights to make informed decisions.

2. Searching for Profiles Based on Criteria

- **Objective**: Enable users to find profiles that closely align with specific search criteria.
- **Functionality**:
    - Users can input free-text queries (e.g., "gym enthusiasts in Houston") to search for potential matches.
    - HeartSync uses BM25 for initial ranking and sends the filtered profiles to the LLM for further analysis against the provided search criteria.
- **Workflow**:
    - The user’s search input is tokenized and matched against the database to shortlist relevant profiles.
    - Shortlisted profiles are passed to the LLM, which evaluates their adherence to the search criteria and provides a score.
    - Execution times for these searches are logged in a file (search_time.log) to monitor and optimize performance.
    - Results are presented to the user in ranked order, complete with scores and explanatory analyses.

3. LLM-Assisted Conversations

- **Objective**: Facilitate engaging and context-aware conversations between users.
- **Functionality**:
    - HeartSync employs the LLM to suggest conversation starters and responses based on user profiles, preferences, and chat history.
    - The suggestions are personalized, ensuring they are relevant to the context of the ongoing interaction.
- **Workflow**:
    - The LLM analyzes the user's and their match's profiles and retrieves past conversation history (if any).
    - Based on this context, it generates tailored conversation suggestions.
    - This feature reduces the cognitive load on users and helps them maintain meaningful conversations, particularly useful for individuals who struggle with initiating or continuing dialogues.

Prompting

Different prompting strategies were used to prompt the LLM for valuable responses.

Prompt for Chat Suggestion

Here, current_user is the user who is currently using the app and prompting the LLM. otherUser is the profile that current_user is checking.

```
    prompt = f"""I am {current_user} and I am talking to {otherUser}. This
is {otherUser}'s profile {user_profile}.
    Based on {otherUser}'s profile and following latest
conversation:\n{conversation_text}\n give me 10 suggestions for my next
      message based on our latest conversation and {otherUser}'s whole
profile.
```

```
      Just give the 10 things that i can reply with, dont write any extra
text.
      Emphasise more on the last message from the person and based on our
profiles and whole conversation.
      Also, if last message from {otherUser} is a question, tell me what
replies i can give. Dont give me another questions."""
```

Prompt for Analyzing Profile
Here, username is the current user prompting the LLM and profile_details are the details of the profile that the current user is checking.

```
        prompt = f"""
        You are an AI assistant helping users on a matchmaking platform.
Analyze the compatibility of the following two profiles.:

        **User Profile (Analyzer)**
        Name: {username}
        Age: {user_profile_dict.get("age")}
        Location: {profile_details.get("location")}  # Assuming the user's
location might also be needed
        Status: {user_profile_dict.get("status")}
        Sex: {user_profile_dict.get("sex")}
        Orientation: {user_profile_dict.get("orientation")}
        Body Type: {user_profile_dict.get("body_type")}
        Height: {user_profile_dict.get("height")}
        Diet: {user_profile_dict.get("diet")}
        Drinks: {user_profile_dict.get("drinks")}
        Drugs: {user_profile_dict.get("drugs")}
        Smokes: {user_profile_dict.get("smokes")}
        Education: {user_profile_dict.get("education")}
        Job: {user_profile_dict.get("job")}
        Ethnicity: {user_profile_dict.get("ethnicity")}
        Religion: {user_profile_dict.get("religion")}
        Offspring: {user_profile_dict.get("offspring")}
        Pets: {user_profile_dict.get("pets")}
        Languages Spoken: {user_profile_dict.get("speaks")}
        Zodiac Sign: {user_profile_dict.get("sign")}
        Bio: {user_profile_dict.get("essay0")}

        **Potential Match Profile**
        Name: {profile_details.get("name")}
        Age: {profile_details.get("age")}
        Location: {profile_details.get("location")}
        Status: {profile_details.get("status")}
        Sex: {profile_details.get("sex")}
        Orientation: {profile_details.get("orientation")}
        Body Type: {profile_details.get("body_type")}
        Height: {profile_details.get("height")}
        Diet: {profile_details.get("diet")}
        Drinks: {profile_details.get("drinks")}
        Drugs: {profile_details.get("drugs")}
        Smokes: {profile_details.get("smokes")}
        Education: {profile_details.get("education")}
        Job: {profile_details.get("job")}
        Ethnicity: {profile_details.get("ethnicity")}
        Religion: {profile_details.get("religion")}
        Offspring: {profile_details.get("offspring")}
        Pets: {profile_details.get("pets")}
        Languages Spoken: {profile_details.get("speaks")}
        Zodiac Sign: {profile_details.get("sign")}
        Bio: {profile_details.get("essay0")}

        Analyze the compatibility of these profiles and explain your
reasoning. Also give the output in English language only, not in any other
language.
        """
```

Prompt for Searching Profile

Here, username is the current user prompting the LLM and profile_details are the details of the profile that the current user is checking.

```
            prompt = f"""
            You are an AI assistant helping users on a matchmaking
platform.

            **Potential Match Profile**
            Name: {other_profile['username']}
            Age: {other_profile['age']}
            Status: {other_profile['status']}
            Sex: {other_profile['sex']}
            Orientation: {other_profile['orientation']}
            Body Type: {other_profile['body_type']}
            Height: {other_profile['height']}
            Diet: {other_profile['diet']}
            Drinks: {other_profile['drinks']}
            Drugs: {other_profile['drugs']}
            Smokes: {other_profile['smokes']}
            Education: {other_profile['education']}
            Job: {other_profile['job']}
            Ethnicity: {other_profile['ethnicity']}
            Religion: {other_profile['religion']}
            Offspring: {other_profile['offspring']}
            Pets: {other_profile['pets']}
            Languages Spoken: {other_profile['speaks']}
            Zodiac Sign: {other_profile['sign']}
            Bio: {other_profile['essay0']}

            Search Criteria: {search_criteria}

            Check if the Potential Match Profile strictly satisfies the
Search Criteria. Based STRICTLY on the Search Criteria, give a score out
of 10.
            Also, generate the score in JSON format, like if you give a
score of 5, give output as {{ "Score": 5 }}.
            Along with this, give a brief analysis of your score in json
format as well like {{ "Score Analysis": "(your analysis)" }}.
            """
```

Challenges and Risks

- **Lack of Suitable Data:**
  The OkCupid dataset, while comprehensive and rich in features, may not fully capture all the nuances and variables required for real-world deployment. For instance, factors such as cultural preferences, regional diversity, or niche interests might not be adequately represented in the dataset. This limitation can lead to gaps in the recommendations, resulting in a mismatch between user expectations and the actual outcomes. Such inadequacies could hinder the system's ability to generalize effectively across diverse user bases, impacting the app's overall appeal and usability.

- **Computational Resources:**
  The simultaneous operation of large language models (LLMs) like Ollama and BM25-based ranking algorithms for matchmaking imposes significant computational demands. These processes require substantial memory and processing power, which can escalate costs, particularly for cloud-based solutions. Additionally, during periods of high user activity, the system might experience latency issues, causing delays in generating matches or chat suggestions. These resource-intensive operations could negatively affect user experience and system scalability if not managed efficiently.

- **Model Complexity:**
  Adapting general-purpose LLMs such as Ollama for specific dating tasks presents unique challenges. These models are not inherently designed to understand the subtleties of compatibility or the tone required for chat suggestions in a dating context. Tailoring these models to align with matchmaking requirements involves careful crafting of prompts and, potentially, fine-tuning. The inherent complexity of such modifications increases the risk of generating irrelevant or overly generic outputs, which could fail to resonate with users and reduce their trust in the system.

- **Subjective Evaluation:**
  The success of a dating application is highly subjective, relying heavily on individual preferences, emotional connections, and real-world interactions. Unlike traditional applications with objective performance metrics, a "successful match" is defined differently by each user. This subjectivity complicates the evaluation process, making it challenging to measure the app's effectiveness quantitatively. Despite accurate recommendations, users might still experience dissatisfaction due to unmet personal expectations, creating a disconnect between the system's performance and perceived success.

- **Ethical and Privacy Concerns:**
  Handling sensitive user data such as personal profiles, preferences, and private conversations necessitates strict adherence to ethical and privacy standards. The integration of AI in matchmaking and conversations raises concerns about data security and user autonomy. Users might feel uncomfortable with the extent of AI involvement, particularly if transparency about its operations is lacking. Moreover, any breach of privacy or perceived misuse of data could damage the app's reputation and erode user trust, posing significant risks to its sustainability.

Proposed Solutions

- **Data Augmentation:**
  To address the limitations of the OkCupid dataset, additional data sources can be incorporated. Publicly available datasets or user-simulated data can supplement the existing data to fill gaps and enhance its diversity. Data generation techniques like synthetic data creation can also be employed to expand the dataset's coverage, ensuring that the recommendation engine considers a broader range of user attributes and preferences. This approach can improve the robustness and inclusiveness of the matchmaking system.

- **Optimization:**
  Performance optimization is crucial to managing computational demands. Techniques such as caching frequently accessed data, implementing efficient querying mechanisms, and batching API requests can significantly reduce processing times. These optimizations ensure that the system remains responsive and scalable even during peak usage. Additionally, leveraging cloud-based infrastructure for dynamic resource allocation can help maintain smooth operations without incurring excessive costs.

- **Pretrained Models:**
  Instead of fine-tuning large LLMs, the project can rely on pretrained models like Ollama, which offer state-of-the-art capabilities out of the box. By focusing on prompt engineering, the system can extract highly relevant and context-aware outputs tailored to the dating domain without incurring the overhead of training. This strategy not only saves computational resources but also accelerates development, enabling faster iterations and deployment.

- **Continuous Feedback Loop:**
  Implementing a feedback mechanism allows users to rate their matches, chat suggestions, and overall satisfaction. This feedback can be systematically integrated into the recommendation engine to refine its accuracy and relevance over time. By creating a user-centric approach, the system evolves to better meet user expectations, ensuring continuous improvement in matchmaking quality and user engagement.

- **Ethical Safeguards:**
  To mitigate privacy concerns, robust data encryption and anonymization methods must be employed. Transparency about AI operations is essential, and users should have control over the extent of AI involvement in their interactions. Regular audits of the system can help identify and rectify any biases, ensuring fairness in recommendations and preserving user trust. Adopting these ethical safeguards builds a foundation for long-term credibility and success.

- **User Personalization:**
  Providing customization options allows users to tailor their experience to their preferences. For example, users can choose the level of AI intervention or specify the tone of chat suggestions. This personalization empowers users and ensures that the system aligns closely with their comfort levels. By fostering a sense of control and adaptability, user satisfaction and trust in the platform can be significantly enhanced.
  By proactively addressing these challenges through innovative solutions, the HeartSync project can overcome potential hurdles and deliver a seamless, user-centric experience that leverages cutting-edge AI technologies to redefine matchmaking and interactions in the digital dating space.

Evaluation

For evaluation, I logged the time taken by LLM to give certain responses

Profile Analysis Time

- LLM took 8.8 seconds to response to a prompt of length 3141 words
- LLM took 3.7 seconds to response to a prompt of length 1925 words
- LLM took 4.6 seconds to response to a prompt of length 2469 words
- LLM took 4.9 seconds to response to a prompt of length 1957 words
- LLM took 4.5 seconds to response to a prompt of length 1957 words

Chat Suggestions Time

- Execution time: 2.383156 seconds
- Execution time: 2.983280 seconds
- Execution time: 2.859001 seconds
- Execution time: 2.753487 seconds
- Execution time: 2.139371 seconds

On an average, LLM took 2.5 seconds to suggest chat messages

Profile Search Suggestion Time

- Execution time: 36.194247 seconds for query: someone who likes adventure
- Execution time: 39.999693 seconds for query: someone who likes diverse conversations
- Execution time: 47.295540 seconds for query: someone who likes diverse conversations and are below age 40
- Execution time: 35.375854 seconds for query: someone who likes going to concerts and like books
- Execution time: 30.888538 seconds for query: someone who likes nature

The evaluation done above is highly dependent on the length of the chat history and the details of the users profiles.

For further evaluation, app has to be used by several people and get their suggestions on how the app is, given the subjective nature of the dating apps.

Some evaluation challenges that can be faced for a dating app are as follows:

- **The Subjectivity of Dating App Success**
  Evaluating the success of a dating app like HeartSync is inherently complex due to the highly subjective nature of relationships and personal preferences. Unlike traditional applications where metrics such as precision, recall, or accuracy provide concrete indicators of performance, the success of a dating app depends on deeply personal and often intangible factors. These include emotional compatibility, mutual interest, shared values, and communication styles—all of which vary significantly from one individual to another. Consequently, traditional quantitative measures fall short of capturing the app's effectiveness in creating meaningful matches or fostering genuine connections.

- **Challenges in Traditional Evaluation Metrics**
  In the context of HeartSync, the subjectivity of evaluation poses unique challenges. For instance, two users might have seemingly compatible profiles based on their attributes and preferences, yet fail to connect on a personal level during interactions. Conversely, an unconventional match that appears incompatible on paper could lead to a meaningful relationship. These nuances make it difficult to assess the app's success purely through algorithmic outputs or system-generated metrics.

- **LLMs Enhancing Match Recommendations**
  Despite these challenges, the integration of Large Language Models (LLMs) such as Ollama provides a valuable advantage in enhancing the user experience. In the realm of match recommendations, the LLM analyzes user profiles, preferences, and personality traits to provide more nuanced and thoughtful suggestions. By leveraging natural language understanding and compatibility analysis, the AI introduces a level of depth that traditional algorithms struggle to achieve. This ensures that users are matched based on a holistic understanding of their individual and relational needs.

- **Facilitating User Conversations**
  Additionally, the LLM plays a crucial role in easing conversations, which is often a significant hurdle in online dating. Initiating and maintaining engaging conversations with matches can be daunting for many users, especially those who struggle with social anxiety or are unsure of what to say. By providing tailored conversation suggestions based on the context of user profiles and previous interactions, the AI reduces the cognitive load on users. This feature empowers them to communicate more confidently, fostering smoother and more enjoyable interactions.

- **Improving Overall User Experience**
  Moreover, LLMs enhance the overall user experience by streamlining matchmaking and interaction processes. The system automates complex analyses of user data, freeing users from manually sifting through profiles or crafting the perfect message. This automation not only saves

time but also makes the app more engaging and accessible, particularly for individuals who are new to online dating or find the process overwhelming.

- **Alternative Evaluation Methods**
  To navigate the subjectivity of evaluation, HeartSync relies on alternative assessment methods such as user feedback and qualitative analysis. Users are encouraged to rate their matches, provide input on the relevance of recommendations, and share their satisfaction with conversation suggestions. These insights are invaluable for iteratively refining the app's algorithms and improving the quality of its services. Additionally, measures like user retention rates, the frequency of successful matches, and the quality of conversations serve as indirect indicators of the app's effectiveness.

Conclusion
HeartSync represents a significant step forward in leveraging advanced AI technologies, particularly Large Language Models (LLMs), to enhance the online dating experience. By integrating AI-driven features such as profile analysis, tailored profile searches, and conversational assistance, the application addresses key challenges faced by traditional dating platforms. The system not only simplifies matchmaking but also empowers users to engage in meaningful and contextually aware interactions, fostering connections on a deeper level.

Through extensive evaluation, HeartSync demonstrated its potential in delivering accurate profile recommendations and engaging conversation suggestions. The system's ability to process complex queries, rank profiles based on nuanced compatibility metrics, and provide real-time conversational insights positions it as a user-friendly and intelligent solution in the dating app market. However, the inherent subjectivity of dating success underscores the importance of continuous user feedback and iterative refinement to meet diverse user needs.

The project also highlighted challenges such as computational demands, data limitations, and ethical considerations, emphasizing the need for robust optimization strategies, data augmentation, and privacy safeguards. By addressing these challenges, HeartSync can continue to evolve into a scalable, inclusive, and ethically sound platform that redefines how users approach online dating.
Overall, HeartSync showcases the transformative power of AI in solving real-world problems, offering a more intuitive, personalized, and impactful dating experience that aligns with the emotional and practical needs of its users.