Afya-Smart-ML Project

Objectives:

The primary goal of the Smart-ML Project is to **substantially improve both the efficiency and effectiveness of the e-consultation process on the Afya-Chat platform**.
Beyond enhancing the current system, the project seeks to **significantly reduce the time it takes for Primary Care Providers (PCPs) to receive referrals or recommendations from specialists**. This reduction in turnaround time means PCPs can provide patients with faster responses to consultations regarding illnesses or diseases. Achieving this involves leveraging the capabilities of advanced Large Language Models, specifically GPT-3.5 Turbo, to streamline and fast-track the consultation process with state-of-the-art AI-driven insights.

Scope:

The Project Smart-ML leverages GPT-3.5 Turbo to enhance the e-consultation process on the AfyaChat platform. This is achieved through the intelligent automation of several key components, designed to streamline consultations, improve the accuracy of medical advice, and ensure compliance with established best-practice guidelines. The project is meticulously structured into three phases, each targeting a specific aspect of the e-consultation enhancement:

David Chuang, in the initial stages of the development of the project, found "*COMMUNITY FIRST HEALTH PLANS PCP MEDICAL RECORD DOCUMENTATION AND CONTINUITY GUIDELINES*", a doc laying out key components of what makes a medical consultation complete for review. It has been corroborated and maintained by the Texas Human and Health Services corporation extensively for many years, making it a best-practise standard for our project too.

Phase I: Identifying Missing Information and displaying as a Checklist to the PCP.

Integration of Best-Practice Guidelines: Utilizing the Community First Health Plans PCP Medical Record Documentation and Continuity Guidelines as a foundation, we developed an ML algorithm powered by GPT-3.5 Turbo. This algorithm analyses e-consultation notes to identify missing critical information that is paramount for a comprehensive medical record.

Real-Time Notification System: Upon identification of missing information, the system automatically generates a checklist and alerts the PCP through a real-time notification via text message, ensuring immediate awareness and action.

Interactive Interface for Completing Missing Information: A user-friendly interface allows PCPs to view the generated checklist and efficiently complete any identified gaps in the consultation notes.

Phase II: Dynamic Generation of Targeted Questions

Feedback-storing, Prioritised Targeted Questions: By feeding GPT-3.5 Turbo with best-practice guidelines and current consultation notes, the model acts as a PCP advisor. Upon providing it content to generate 4-5 MOST IMPORTANT questions for the particular

disease, it dynamically generates targeted questions enhancing the consultation's depth and ensuring adherence to medical standards.These questions and recommendations are integrated into the AfyaChat platform, prioritized to guide the PCP through the most critical consultation aspects first.

Feedback-Driven Response Mechanism: PCPs can easily respond to targeted questions through AfyaChat, with their interactions contributing to the ongoing improvement of the consultation process.

Phase III: Provision of Validated Recommendations and Continuous Learning

Evidence-Based Recommendations: Leveraging the power of GPT-3.5 Turbo, the system analyzes the e-consultation data against the latest clinical guidelines to generate validated, evidence-based recommendations for patient care.

Feedback Loop for Model Refinement: All interactions with the system, including the PCP's responses to targeted questions and feedback on recommendations, are captured and reincorporated into the GPT-3.5 Turbo model. This continuous learning loop ensures the recommendations are refined and remain aligned with current medical practices and guidelines.

Documentation Enhancement: Functionally, the system enhances the documentation process by storing all interactions (checklists, responses to targeted questions, feedback on recommendations) and adding them back as text in the consultation file. This ensures a comprehensive and evolving record of the consultation process, enhancing both the quality of care and the educational value for PCPs.

Team Member: Roles

Smart ML is a team of 3 members: Debalina Chowdhury, Saksham Rai and Cheng-Yun Chuang.

Debalina's role: Debalina contributed on the following prospects of the project:

1. Brainstormed with teammates and designed workflow diagram of the API
2. Prepared authorization mechanism for the PCP login page
3. Researched state-of-the-art LLMs for medical text generation
4. Developed frontend and backend for user story 3 i.e. displaying evidence based recommendations given eConsult data
5. Ensured correct navigation/routing between API endpoints
6. Performed unit tests for routing and user story 3
7. Completed documentation for development tools, project management tools and API design and testing(user story 3)

Cheng-Yun's role: Cheng-Yun contributed on the following prospects of the project:

1. Developed frontend and backend components for user story 1, including the missing information and import data page.
2. Ensured seamless data flow from the backend, where GPT model-generated data is processed, to each of the three frontend pages that display the results.
3. Conducted thorough unit tests for routing and the user story 1 functionalities.

4. Completed comprehensive documentation for ML Models, Security Measures, and Testing related to user story 1.
5. Researched and compared results using the OpenAI API, evaluating its performance against other existing ML models such as Med-Palm2 and ClinicalBert.
6. Collaborated with teammates in brainstorming sessions, providing valuable suggestions, and working as an integral part of the team to refine the implementation.

Saksham Rai's role:

1. Contributed in preliminary brainstorming sessions to come up with innovative solutions for each user-story and possible tradeoffs of different types of implementations.
2. Implemented designs for front-end for all three user stories. In addition, I wrote backend code for User Story 2: to generate and display Targeted Questions.
3. Implemented the Twillio real-time notification service to send the missing information checklist straight to the specialist phone number.
4. Wrote unit-tests for correct routing and responses for User Story 2 and front-end tests for the service page of the user story.
5. Created the powerpoint presentation of Smart_ML project for Demo Day.
6. Wrote Objective, Scope and System Architecture sections of the Engineering Doc.
7. Researched and compared results from the GPT 3.5 turbo model, evaluating its performance against other existing ML models such as Med-Palm2, ClinicalBert and also Llama Index.

Development Tools:

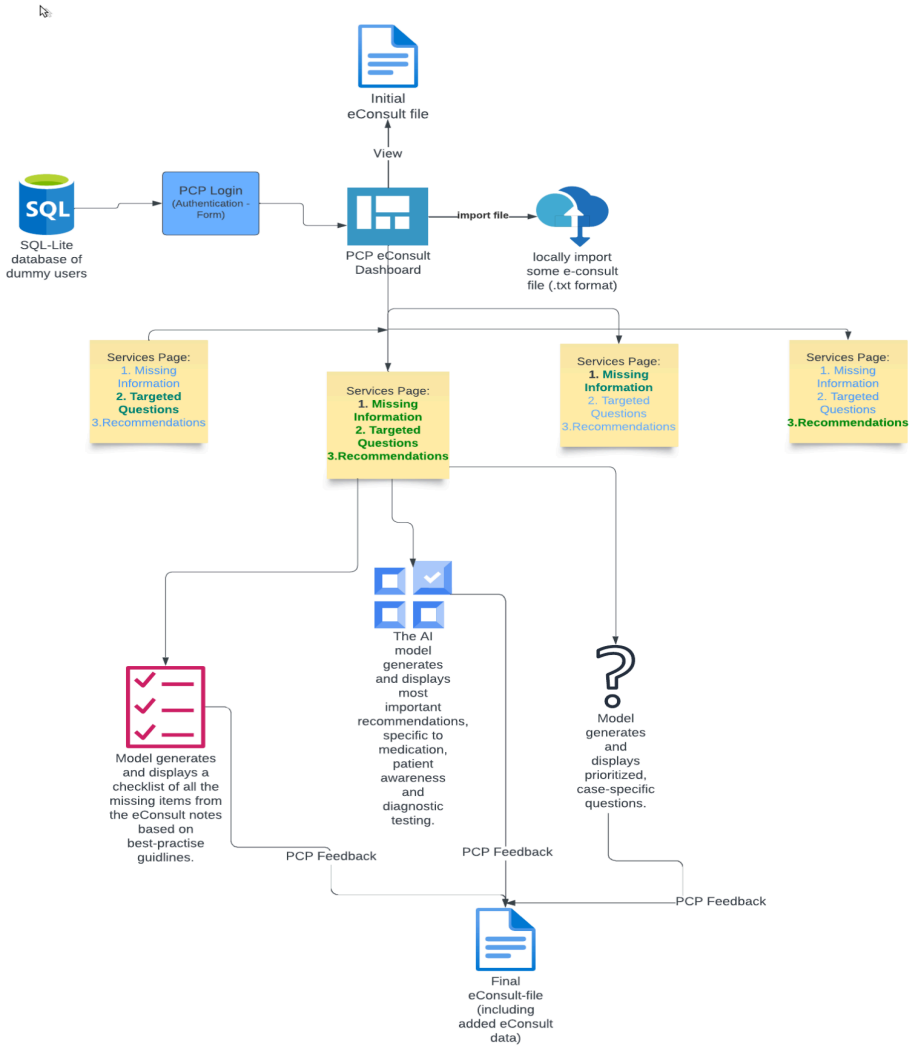
List the software, frameworks, libraries, and tools used for development, specifying versions where applicable.

1) **Frontend**: HTML, CSS and javascript files have been used to develop the frontend
2) **Backend**: Sqlite has been used to build the PCP database and python programming has been used for text generation required for the three user stories.
3) **Frontend-backend integration**: Flask framework has been leveraged to integrate frontend and backend.
4) **Testing**: Pytest has been utilized to perform unit testing on frontend as well as backend.
5) **Python libraries**: The following python libraries have been installed via requirements.txt for the smooth workflow:
    a) flask==3.0.2
    b) requests==2.31.0
    c) pandas==2.2.1
    d) pysqlite3==0.5.2
    e) openai==1.16.2
    f) twilio==9.0.4
    g) pytest==8.1.1

Project Management Tools:

Google doc has been used for managing tasks and daily updates. Github issues have been used for collaboration and tracking progress.

System Architecture:



System Architecture and Data Flow:

• **User Authentication:** PCPs log in to the system via an authentication form, interacting with an SQL-Lite database that validates credentials using a stored list of dummy users.

• **Consultation File Import:** PCPs upload initial eConsult files (text format) into the PCP eConsult Dashboard. This import initializes the consultation process.

• **Service Page Interaction:** Upon file import, when the PCP clicks on the imported eConsult data, the system directs PCPs to the Services Page (for that particular consultation), where the following interactions occur:

• I**dentification of Missing Information:** The AI model scans the consultation notes, cross-references with best practice guidelines, and highlights any missing critical information.

• **Generation of Targeted Questions:** The model analyzes the case details and generates prioritized, case-specific questions for the PCP to address.

• **Provision of Recommendations**: The system delivers AI-driven, evidence-based recommendations for treatment options and further diagnostic tests.

• **Feedback Loop**: PCPs can provide feedback on each of the AI model's outputs, which is used for continuous learning and system improvement.

• **Final eConsult File Generation**: The PCP completes the consultation by adding the missing information, answering targeted questions, and considering the recommendations. A final eConsult file, now enriched with comprehensive data, is generated for submission.

API Design

API design for Smart ML aims at facilitating Primary Care Physicians (PCPs) to bridge common gaps between patient eConsult data and specialist requirements in a standard sequence. The API endpoints along with their descriptions are as follows:

1. **app/**: This directory contains the main application code.
    - 1.1. **templates/**: Contains HTML templates used for rendering web pages.
        - 1.1.1. **base.html**: Base template for other HTML files.
        - 1.1.2. **index.html**: GET: Render login page, POST: Authentication of user
        - 1.1.3. **import_page.html**: Webpage to show the dashboard with the functionality of importing data and working on the three operations and also for logout
        - 1.1.4. **operation1.html**: GET: Render operation 1 page, POST: navigate to the webpage of missing information list
        - 1.1.5. **missing_info.html**: Webpage to show missing information list and provide additional feedback
        - 1.1.6. **operation2.html**: GET: Render operation 2 page, POST: navigate to the webpage of targeted questions list
        - 1.1.7. **targeted_questions.html**: Webpage to display targeted questions with input fields for PCPs to provide responses
        - 1.1.8. **operation3.html**: GET: Render operation 3 page, POST: navigate to the webpage of evidence-based recommendations
        - 1.1.9. **recommendations.html**: GET: Webpage to exhibit recommendations for the diagnostic testing, medication and patient education/management. It will contain criteria with dropdown for clinical review of recommendations and input field for overall feedback
        - 1.1.10. **operation4.html**: GET: Render operation 4 page showing all 3 operations have been completed and PCP can click on “Go to dashboard” to return to dashboard
    - 1.2. **static/**: Holds static folder for CSS and javascript files.
        - 1.2.1. **css/**: Contains css files for HTML files in templates folder
            - 1.2.1.1. **style.css**: CSS file for styling the web pages.
            - 1.2.1.2. **missing_info.css**: CSS file for missing info webpage
            - 1.2.1.3. **targeted_questions.css**: CSS file for targeted questions webpage
            - 1.2.1.4. **recommendations.css**: CSS file for recommendations webpage
        - 1.2.2. **js/**: Contains javascript files for HTML files in template folder
            - 1.2.2.1. **done.js**: Javascript file to navigate between operations sequentially

        1.2.2.2. **click_operation.js**: Javascript file to navigate from operation clickable page to the respective operation page
        1.2.2.3. **options.js**: Javascript to show the different options for the criteria of clinical reviews in the recommendation webpage
        1.2.2.4. **all_done.js**: Javascript to navigate from operation4 page to dashboard
    1.3. **data/**: Stores data files used by the application.
        1.3.1. **database.db**: Dummy database of PCP names and corresponding passwords
        1.3.2. **database_creation.py**: Module used to create PCP database
    1.4. **routes.py**: Module defining Flask routes for the web application.
    1.5. **userstory1.py**: Module using OPEN AI 3.5 to generate a list of missing information given eConsult data file
    1.6. **userstory2.py**: Module using OPEN AI 3.5 to generate a list of targeted questions given eConsult data file
    1.7. **userstory3.py**: Module using OPEN AI 3.5 to generate 3 evidence based recommendations given eConsult data file
2. **config.py**: Configuration file for API IDs to be used for the application.
3. **run.py**: Script to run the Flask application.

Machine Learning Models

Model Selection

ChatGPT 3.5

After conducting extensive trials with various models, including BERT, Bio-BERT, Clinical BERT, and Med-Palm2, we found that these models did not meet our requirements. They either lacked the precision needed for our task or required access to training data that we did not have. In contrast, ChatGPT 3.5 emerged as the most suitable choice for the following reasons:

- **Versatility**: ChatGPT is a versatile language model capable of handling a wide range of natural language processing tasks.
- **Large Knowledge Base**: ChatGPT has been trained on a vast amount of text data, providing a broad knowledge base to draw from.
- **Ease of Use**: ChatGPT is easy to use and integrate into our project, requiring minimal setup.
- **No Training Data Required**: ChatGPT can be used out of the box without the need for additional training data.
- **Fast Response Times**: ChatGPT is optimized for speed, providing quick responses.
- **Availability**: ChatGPT is readily available for use, making it accessible for our hackathon project.

ML Model Integration: Explain how machine learning models are integrated and interact with the back-end.

Model Integration on Backend

We used Python Flask as the framework to develop our application. We designed distinct functions for three user stories, with the model's responses based on the guidelines we provided. The GPT model was assigned the roles of PCP and healthcare professional. We integrated the ChatGPT API to process user input, which consists of patient E-consult data combined with guidelines for different user stories. The model generates responses in a bullet-point format, allowing us to split each line for display on the frontend. Each response from the model is converted into a completion object, which we then transform into a list of dictionaries containing each individual answer.

Model Interaction between Frontend and Backend

In our project, on the frontend, we import consultation data in .txt file format and fetch the JSON data to the backend, where we process it using different modules we've implemented. The generated and well-formatted data is then sent back through the routing function to the HTML file for display. Additionally, we store the responses of PCPs in a separate .txt file for future model training.

The future model training phase aims to enhance the performance and accuracy of the ChatGPT model by leveraging the responses of PCPs collected during consultations. This data is valuable for training the model to provide more accurate and relevant suggestions and recommendations based on real-world clinical expertise.

Llama Index

LlamaIndex is an orchestration framework designed to facilitate the integration of private data with public data for building applications using Large Language Models (LLMs). It addresses the challenge of integrating unstructured, siloed private data with publicly available data used to train LLMs. By providing tools for data ingestion, indexing, and querying, LlamaIndex enables the augmentation of public data with private data, allowing for the development of more robust and contextually aware applications.

We implemented a system using the LlamaIndex framework to ingest and index pre-processed healthcare data, including message body of healthcare messages and specialty names. We then compared the responses generated by a model using this pre-processed data with the responses generated by a model without using the private data.

ML Model Evaluation:

As we do not train any ML models, our approach involves comparing the performance of GPT-3.5 models with and without pre-processed private data from Afya Company. We evaluate the models based on their precision and logic, using our own data examples as benchmarks.

Testing

Unit Testing: Describe the approach to unit testing for both front-end and back-end components.
(just mention for what components are you writing unit tests for and what are you checking)

1. **tests/**: Contains unit tests for the application.
    1.1. **test_routes.py**: Unit tests for Flask routes.
    1.2. **test_login_page.py:** Unit tests for presence of username and password input fields and authorization.
    1.3. **test_operation1.py:** Unit tests for presence of all operations buttons and active missing list button and go to dashboard button and other buttons disabled.
    1.4. **test_userstory1.py:** Unit tests for proper response type of missing lists
    1.5. **test_missing_info.py:** Unit tests for proper display of missing list
    1.6. **test_operation2.py:** Unit tests for presence of all operations buttons and active targeted questions list button and go to dashboard button and other buttons disabled.
    1.7. **test_userstory2.py:** Unit tests for proper response type of targeted questions
    1.8. **test_targeted_questions.py:** Unit tests for proper display of targeted questions
    1.9. **test_operation3.py:** Unit tests for presence of all operations buttons and active recommendations button and go to dashboard button and other buttons disabled.
    1.10. **test_userstory3.py:** Unit tests for
    1.11. **test_recommendations.py:** Unit tests for proper display of recommendations

Security Measures

Assumptions and Dependencies:

- **Assumption**: PCPs have access to some form of e-consultation notes when they interact with our application.
  Dependency: The application's functionality relies on the availability of e-consultation notes to generate insights and recommendations.
- **Assumption**: The e-consultation notes provided by PCPs are in a structured format that can be processed by the application.
  Dependency: The application's algorithms require structured data to function accurately.

- **Assumption**: Users interacting with the application have a basic understanding of medical terminology and procedures.
  Dependency: The application's interfaces and outputs are designed with the assumption that users have a certain level of medical knowledge.

- **Assumption**: The application will be used in a controlled environment where data privacy and security measures are in place.
  Dependency: The application's security measures rely on the underlying infrastructure and environment in which it is deployed.

Risks and Mitigations:

1. **Secure Data Transmission:** Consultation communications were transmitted over secure channels, such as HTTPS, to prevent eavesdropping and data tampering.
2. **Authentication and Authorization:** A robust authentication and authorization mechanism was implemented to ensure that only authorized users had access to the consultation communications.
3. **Data Minimization:** The project minimized the collection and storage of sensitive information such as api keys to reduce the risk of data breaches.
4. **Application Security:** These security measures ensured that consultation communications between PCPs and specialists were protected from unauthorized access and tampering, maintaining the confidentiality and integrity of the data.