AMERICAN UNIVERSITY OF BEIRUT

MAROUN SEMAAN FACULTY OF ENGINEERING & ARCHITECTURE

EECE 490 – Project MLops

AI Home Doctor Chatbot MLops

MLops for course EECE 490

Fall 2024

Group Members:

Dayeh, Hadi
Shammaa, Nour
El Kari, Riwa
Mazraani, Zeinab

Dr. Ammar Mohanna

Date: December 30, 2024

Contents

1. MLops and Overview

1.1. Hierarchical Diagram:

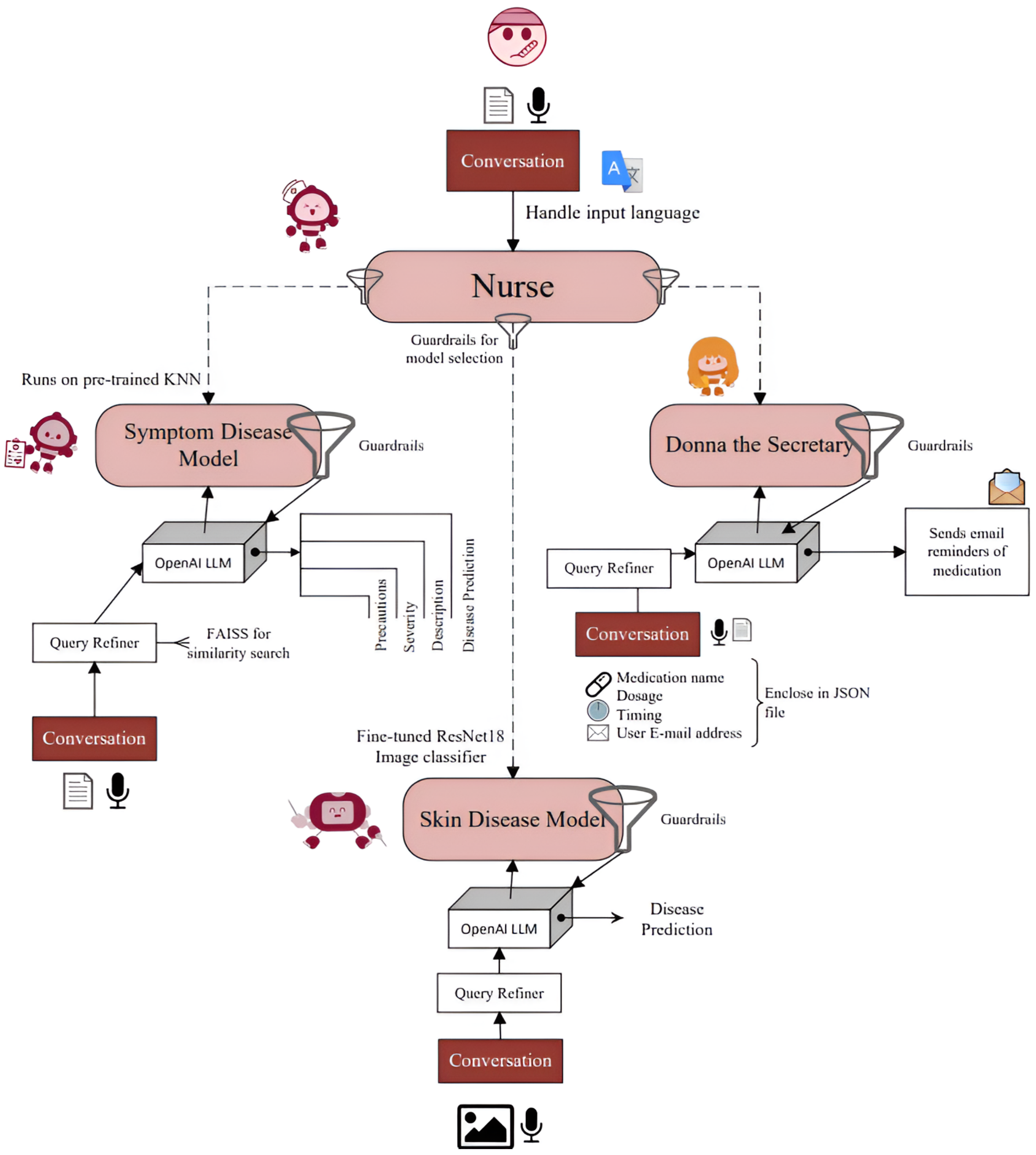


Figure 1: Hierarchical Diagram of the Models

The figure above presents a hierarchical flow diagram outlining the internal workings of the code. The primary interface, represented by the "Nurse" component, acts as a central orchestrator for processing user input and delegating tasks to specialized models. The overall workflow can be summarized as follows:

1. **User Interaction (Conversation Layer):** The user provides input, either through speech or text. This input may describe symptoms, request medication reminders, or include images of skin conditions. All user input initially arrives at the "Nurse" component.
2. **Language Handling and Intent Determination (Nurse):** The Nurse module uses natural language understanding

to interpret the user's request. It applies predefined guardrails (rules and constraints) to ensure the request is safe and falls within the supported domain. Based on the content of the request, the Nurse decides which specialized subsystem is best suited to handle the query.

3. **Routing to Specialized Models:** Depending on the user's intent, the Nurse forwards the request to one of the following specialized models:

   - **Symptom Disease Model:** For symptom-related inquiries, the input is passed to the Symptom Disease Model. This model employs a FAISS-based similarity search to retrieve relevant medical information from a pre-trained knowledge base and uses a dataset with symptoms, description of disease, precautions, and severity.and uses an OpenAI Large Language Model (LLM) to provide a preliminary diagnosis or suggested conditions. A query refiner ensures that the user's input is transformed into a structured, model-friendly query.
   - **Donna the Secretary:** If the user requests assistance with medication scheduling or reminders, the Nurse routes the query to "Donna the Secretary." This subsystem refines the query to extract key details such as medication name, dosage, timing, and the user's e-mail address. An OpenAI LLM then converts this information into a structured JSON file, which is subsequently used to send automated e-mail reminders to the user.
   - **Skin Disease Model:** For image-based queries (e.g., when the user uploads a photo of a skin condition), the Nurse sends the image and related context to the Skin Disease Model. A fine-tuned ResNet18 image classifier processes the image to identify potential skin conditions, and an OpenAI LLM generates a human-readable explanation and disease prediction.

4. **Output and Response Generation:** After the specialized model processes the request, a structured, contextually relevant response is returned to the user. For symptom inquiries, the user might receive a list of possible conditions and next steps. For medication reminders, a confirmation that reminders have been scheduled and e-mails sent is provided. For image-based queries, the user receives a probable diagnosis and recommended follow-up actions.

5. **Guardrails and Safe Completion:** Throughout the process, guardrails ensure that the models operate within intended parameters. These guardrails help maintain patient privacy, prevent disallowed content, and ensure that the models' outputs remain medically relevant and safe.

In essence, the code implements a layered, conversational healthcare assistant. The Nurse module serves as a central node that interprets user input, routes requests to the appropriate specialized model (Symptom Disease, Donna the Secretary, or Skin Disease Model), and returns a comprehensive and well-structured response to the user.

1.2. MLOps

Below are the MLOps pipeline of the different chains:

1.2.1 Nurse (Base Chain):

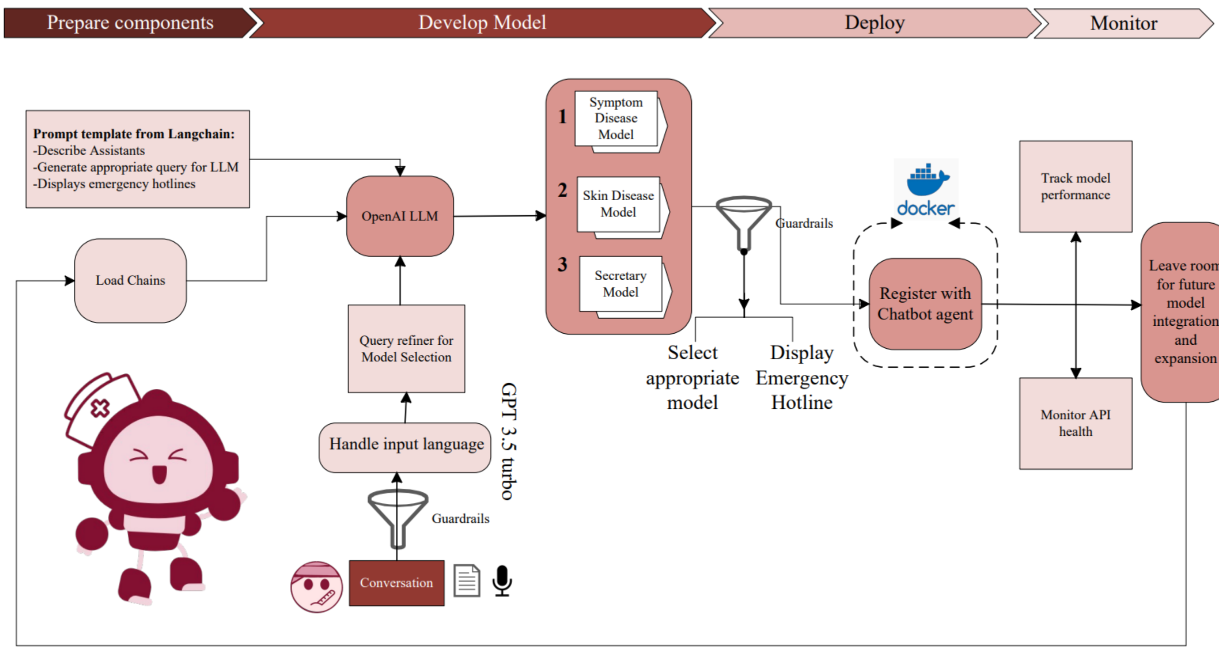


Figure 2: Nurse MLOps cycle

Nurse MLOps Cycle: The Nurse component of the home doctor chatbot system provides users with a reliable and guided medical assistant experience. It processes health-related inquiries through a carefully orchestrated pipeline of large language models (LLMs), domain-specific models, and safety mechanisms. Below is a detailed step-by-step explanation of how the system operates.

1. Overall Structure and Workflow

The system is divided into four major phases: **Prepare Components**, **Develop Model**, **Deploy**, and **Monitor**. Each phase is critical to ensuring that the chatbot functions correctly, provides safe and accurate health information, and remains adaptable to future enhancements.

- **Prepare Components:** This phase assembles fundamental building blocks such as prompt templates, toolchains, specialized models, and guardrails.
- **Develop Model:** This phase integrates the primary language model and specialized domain models. It refines user queries, selects the appropriate model, and ensures that the output is safe, medically appropriate, and contextually relevant.
- **Deploy:** This phase packages the finished model, typically using containerization (Docker), and connects it to the chatbot agent that end-users interact with.
- **Monitor:** In this final phase, the system tracks performance, reliability, and scalability. The architecture is designed to allow for the seamless integration of new models and features as needs evolve.

2. Prompt Template from LangChain

At the core of the system, **LangChain** is used to handle prompt engineering. The prompt template:

- **Describes Assistants:** It clearly defines the role and persona of the nurse assistant, ensuring a tone of empathy, professionalism, and helpfulness.
- **Generates Appropriate Queries for the LLM:** Given a user's natural language question, the prompt template guides the LLM to produce structured, context-rich queries for downstream models.
- **Displays Emergency Hotlines:** When the user's situation calls for immediate attention, the prompt includes instructions for the LLM to display emergency hotline numbers or direct the user toward seeking urgent medical help.

3. Load Chains

Predefined ”chains” from LangChain are loaded at this stage. These chains are sequences of LLM calls, transformations, and tools that run in order. They handle:

- **Pre-Processing:** Chains clean and format the user's input, ensuring it fits the model's requirements.
- **Query Transformation:** Chains break down the initial user query into more specific sub-queries suitable for selecting and running the appropriate healthcare models.

4. Handling User Input and Applying Guardrails

Before invoking the main logic, the user's query is passed through content safety filters known as ”guardrails.” These guardrails:

- **Ensure Content Safety:** They filter out harmful or disallowed content.
- **Maintain Medical Appropriateness:** They guide the conversation so that the system provides relevant and safe health-related assistance.

If the user's query is in Arabic, the system uses GPT-3.5 Turbo to translate it into English. This ensures the entire pipeline, including the downstream domain models, can work effectively on the standardized input. Once translated and verified, GPT-3.5 Turbo interprets the user's needs, acting as a language interpretation layer to convert a user's natural query into a well-defined internal representation.

5. Query Refinement for Model Selection

After the query is understood (and translated if necessary), a specialized ”Query Refiner” prepares it for selecting the most suitable model. The refiner examines the user's request to determine which domain model is best suited—whether it involves symptoms, skin-related issues, or administrative tasks.

6. Multiple Domain-Specific Models

The system includes three distinct domain models tailored to different types of medical inquiries:

- **Symptom Disease Model:** Focused on general health symptoms. When users mention symptoms like headaches, fatigue, or persistent coughs, this model identifies potential conditions and provides appropriate guidance.
- **Skin Disease Model:** Dedicated to prediction of skin diseases based on images provided by the user
- **Secretary Model:** Handles administrative or organizational tasks, such as booking appointments, providing healthcare facility contact details, or assisting with schedule management.

7. Registering the Result with the Chatbot Agent

Once the response is finalized and verified, it is registered with the chatbot platform. The entire pipeline is containerized—ensuring it runs reliably within a **Docker** environment—and made accessible through a defined API endpoint. Containerization guarantees consistent performance and simplifies updates, scaling, and integration.

8. Deployment and Integration

The entire chatbot system, including the Donna Chain, is packaged into a single Docker image. This Dockerized environment ensures consistency across different deployments and simplifies the setup process. While the current focus is on running the system as a Docker container, this setup allows for potential integration with various interfaces, such as web pages, mobile apps, or voice-activated devices, in future iterations.

9. Monitoring and Future Expansion

The system includes continuous monitoring capabilities:

- **Model Performance Tracking:** Performance metrics are recorded and evaluated for accuracy, response time, and user satisfaction.
- **API Health Monitoring:** System health, uptime, and request handling are continuously supervised to ensure reliability.

Feedback loops provide insights for improvement, allowing the integration of new models or features in the future. If new types of inquiries arise, additional specialized models can be added to the pipeline. The architecture's modular design leaves room for ongoing expansion, ensuring the chatbot can adapt to emerging healthcare needs and user requirements.

In summary, the nurse component of the home doctor chatbot is a sophisticated orchestrator. It receives raw user queries—whether in English or Arabic—applies safety and quality filters, leverages both large language and specialized disease models, translates language as needed, delivers medically appropriate guidance (with options for emergency escalation), and continuously evolves based on performance monitoring and user feedback.

1.2.2 Symptom Disease Chain MLOps Cycle:

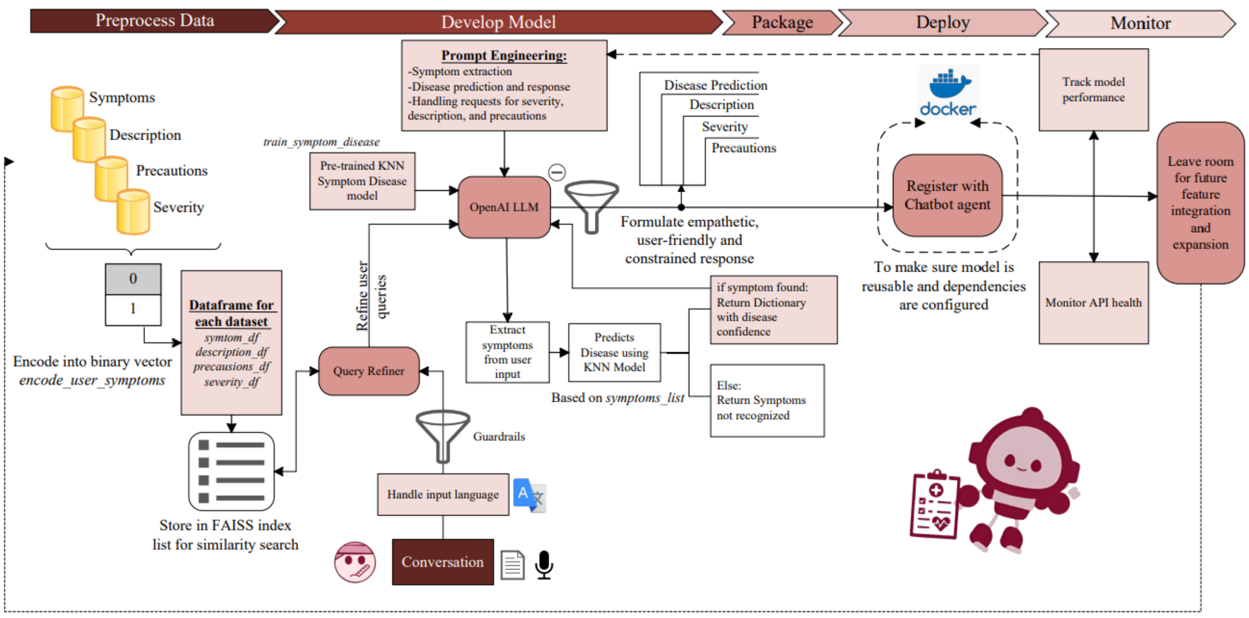

Preprocess Data
Develop Model
Package
Deploy
Monitor
Symptoms
Description
Precautions
Severity
0
1
Encode into binary vector
encode_user_symptoms
Dataframe for each dataset
symtom_df
description_df
precautions_df
severity_df
Store in FAISS index list for similarity search
train_symptom_disease
Pre-trained KNN Symptom Disease model
Prompt Engineering:
-Symptom extraction
-Disease prediction and response
-Handling requests for severity, description, and precautions
OpenAI LLM
Refine user queries
Query Refiner
Guardrails
Handle input language
Conversation
Extract symptoms from user input
Predicts Disease using KNN Model
Based on symptoms_list
if symptom found: Return Dictionary with disease confidence
Else: Return Symptoms not recognized
Disease Prediction
Description
Severity
Precautions
Formulate empathetic, user-friendly and constrained response
docker
Register with Chatbot agent
To make sure model is reusable and dependencies are configured
Track model performance
Monitor API health
Leave room for future feature integration and expansion

Figure 3: Symptom Disease Model MLOps cycle

The Symptom Disease chain system (Figure 5) is designed to handle user inquiries about symptoms and predict possible diseases. It relies on a single dataset base, which is composed of four separate files covering symptoms, description, precautions, and severity information. The pipeline identifies the user's symptoms, predicts the most likely disease, and—only upon explicit user request—provides additional information such as severity, precautions, or description. The retrieval of these additional details is performed using a FAISS index designed for efficient similarity searches on textual embeddings.

1. Single Dataset with Four Files

The system relies on one integrated dataset base, which is split into four separate files for modularity:

- **Symptoms File:** Contains symptom information and mappings used for disease prediction.
- **Description File:** Contains textual descriptions of diseases.
- **Precautions File:** Lists preventive measures associated with various diseases.
- **Severity File:** Provides severity levels or seriousness indicators for different conditions.

These files together form the knowledge base. The code treats them as a unified dataset, ensuring all relevant information about symptoms and diseases is easily accessible when needed.

2. Preprocessing and Encoding

Before predictions occur, the symptom data is processed and encoded:

- **Symptom Encoding:** The user's symptoms are represented in a structured format (e.g., as a binary vector or a standardized list of symptoms present or absent). This representation allows the KNN model to efficiently predict the most likely disease.
- **FAISS Index for Additional Details:** A separate FAISS index is built to enable fast similarity-based retrieval of descriptions, precautions, and severity information.

3. Query Refiner

The Query Refiner is a crucial component that interprets the user's input. It does more than just clarify the user's symptoms:

- **Determining User Intent:** The Query Refiner identifies the user's primary goal. Is the user specifically requesting severity, precautions, or a description of the disease?

- **Symptom Extraction:** If the user provides raw input mentioning certain ailments (e.g., ”I have a headache and a rash”), the symptom extraction prompt is used to extract these symptoms in a clean format that the KNN model can understand.
- **Request for Additional Information:** The Query Refiner checks if the user asked about severity, precautions, or description. If not, it does not provide these details by default. However, it can inform the user that they may ask for these details if needed.

4. KNN Disease Prediction (No FAISS for Symptoms)
After refining the query to a structured symptom representation, the system uses a pre-trained K-Nearest Neighbors (KNN) model to predict the disease:

- **Disease Identification:** The KNN model uses the encoded symptom vector (or structured representation) to find the closest known symptom patterns in its training data and selects the most probable disease.
- **No Additional Info from KNN:** The KNN model returns only the predicted disease. It does not provide severity, precautions, or description. Those details are fetched separately if the user requests them.

5. Controlled Response Generation Using LLM
The OpenAI LLM (e.g., GPT-based) works with the Query Refiner and the KNN output to produce a user-facing response:

- **Primary Focus on Disease Name:** Upon initial inquiry, the LLM confirms the predicted disease. It does not automatically mention severity, precautions, or description unless the user has asked for them.
- **Encouraging Further Inquiry:** The LLM may inform the user that they can ask for more details (severity, precautions, description) if they want to. This ensures users retain control over the level of detail they receive.
- **Empathetic and User-Friendly Tone:** The prompt engineering ensures the LLM’s response is not only accurate but also empathetic and easy to understand. It provides medically appropriate language without overwhelming the user.

6. On-Demand Retrieval with FAISS
If the user subsequently asks, “What is the severity of this disease?” or “Can you give me precautions?” the system then:

- **FAISS Lookups:** Uses the FAISS index to quickly retrieve the requested information (severity, precautions, or a detailed description) based on the predicted disease.
- **Contextual Integration:** The LLM integrates the retrieved information into a cohesive, user-friendly message, thereby answering follow-up questions with relevant details from the unified dataset.

7. Packaging, Deployment, and Monitoring
Once the pipeline is tested and verified:

- **Containerization:** The entire system—query refinement logic, KNN model, LLM integrations, and FAISS index—is containerized using a tool like Docker. This encapsulation simplifies deployment and environment consistency.
- **Registering with Chatbot Agent:** The containerized solution is registered with a chatbot platform, making it accessible to users through APIs or user interfaces on websites, mobile apps, or other platforms.
- **Ongoing Monitoring and Scalability:** The system’s performance, accuracy, and uptime are continuously monitored. The architecture allows for easy integration of improved models, updated data files, or new features as user needs evolve.

In summary, this Symptom Disease chain pipeline takes a user’s symptom inquiry, refines it, predicts the disease using a KNN model, and provides additional severity, precautions, and descriptive information on-demand by querying a FAISS index. The Query Refiner ensures that users receive exactly what they ask for—just the disease initially, and additional details only if requested—while the LLM ensures the communication remains empathetic and clear. This modular and flexible design enables the system to be both efficient and user-centric.

1.2.3 Skin Disease Chain MLOps Cycle:

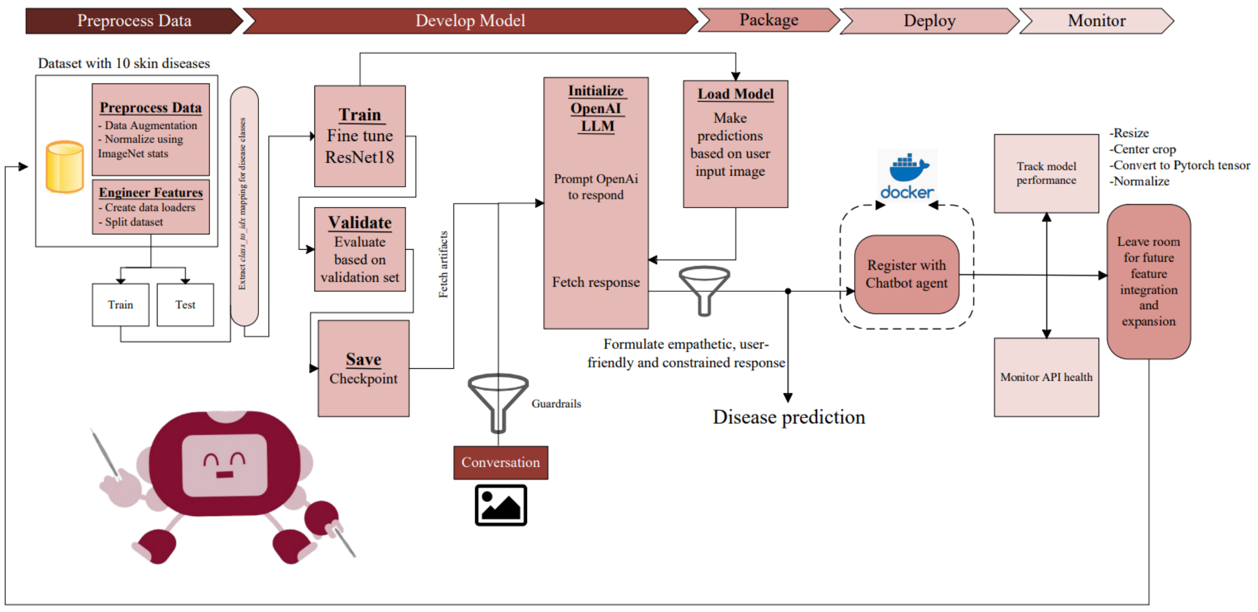


Figure 4: Skin Disease Model MLOps cycle

The Skin Disease chain is designed to identify and communicate potential skin diseases based on user-provided images. This pipeline integrates image preprocessing, model training and validation, and a language model (LLM) to produce empathetic, user-friendly responses. The code (in `skin_disease_chains.py`) outlines how the system receives user input, prompts for images, processes images through a fine-tuned model, and returns disease predictions. Below is a detailed, step-by-step explanation aligned with the provided diagram.

1. Preprocess Data

The starting point is a curated dataset containing images of 10 different skin diseases. The preprocessing steps include:

- **Data Augmentation:** Applying transformations like random crops, flips, or rotations to increase the dataset's diversity and improve the model's generalizability.
- **Normalization Using ImageNet Stats:** Adjusting the dataset's pixel values to match ImageNet means and standard deviations, aligning with common deep learning best practices.
- **Engineer Features and Split Dataset:** Creating data loaders for training and testing sets, ensuring the data is appropriately divided for model development and evaluation.

The result of this stage is a well-prepared dataset ready for model training, ensuring robust input quality.

2. Training and Validation of the Disease Classification Model

A **ResNet18** model (or a similar pre-trained architecture) is fine-tuned on the processed dataset:

- **Train:** The model learns to distinguish among the 10 skin diseases from the training subset. The code updates the ResNet18 weights through backpropagation, optimizing the model's parameters.
- **Validate:** A validation subset is used to evaluate the model's performance mid-training. This step ensures the model is generalizing well and not overfitting to the training data.
- **Save Checkpoint:** The best-performing model weights (based on validation metrics) are saved as a checkpoint. This makes it easy to revert to a known good state and facilitates deployment.

3. Model Artifacts and LLM Initialization

Once the model is trained and validated, relevant artifacts (e.g., class-to-index mappings, model weights) are available. The next phase involves:

- **Initialize OpenAI LLM:** The language model (e.g., GPT-based) is configured with a custom prompt to respond empathetically to the user. This LLM:

  - Encourages the user to provide an image if one is not already provided.
  - Adapts responses to the user's language preferences (e.g., if the user is speaking Arabic, it responds in English and suggests switching to Arabic mode).
  - Does not diagnose without an image, ensuring users are guided to supply visual evidence before receiving a prediction.

4. Loading the Model and Predicting the Disease
At runtime, the saved ResNet18 model is loaded for inference:

- **User Input Image:** When the user provides an image of the affected skin area, it passes through the same normalization and feature extraction steps used during training.
- **Disease Prediction:** The loaded model runs a forward pass on the processed image and outputs a predicted disease class. This class is then interpreted by the LLM and integrated into the response.

5. LLM-Driven Conversational Response
The LLM uses the predicted disease and the conversation history to produce a user-friendly, empathetic response. Key aspects include:

- **Empathetic Language:** The LLM avoids technical jargon and speaks naturally, aiming to reassure users.
- **Adherence to Prompt Guidelines:** The LLM does not mention internal metrics or confidence scores, does not provide unauthorized medical advice, and avoids diagnosing without an image. The response encourages professional consultation if the prediction is severe, referencing available emergency contacts such as the Lebanese Red Cross number (140).

6. Packaging and Deployment
After ensuring the pipeline works end-to-end, the model and code are packaged for deployment:

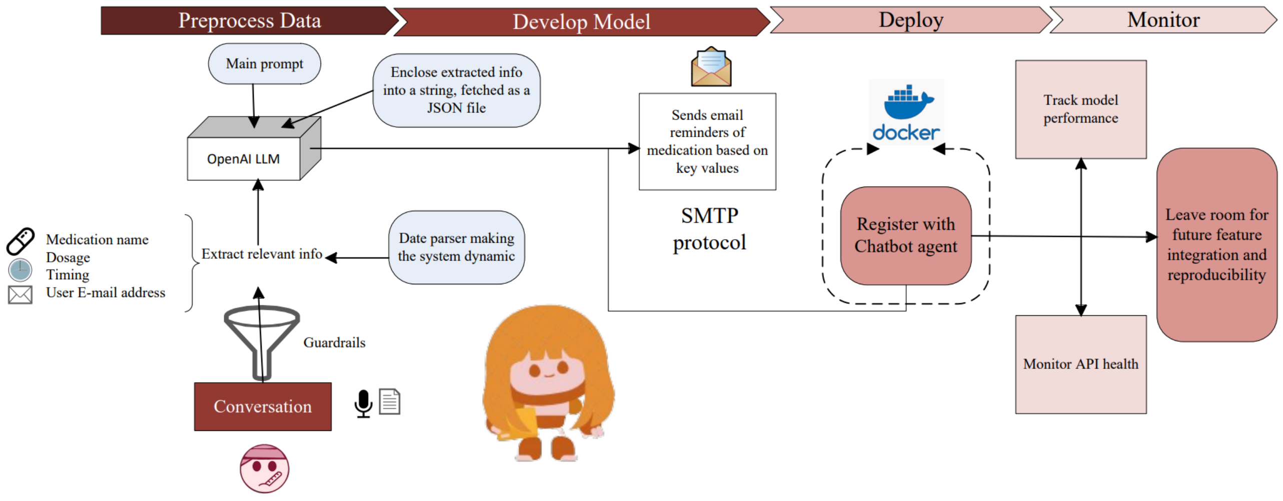
- **Containerization with Docker:** The entire stack—preprocessing, model inference, LLM integration—is containerized for consistency and portability.
- **Registering with Chatbot Agent:** The containerized model is integrated into a chatbot platform. Users can now interact with the model through web or mobile interfaces.

7. Monitoring and Future Expansion
The final step involves continuous monitoring and maintenance:

- **Track Model Performance:** Logs and metrics assess accuracy, response time, user satisfaction, and system stability.
- **Monitor API Health:** The deployed service is monitored for uptime, latency, and errors, ensuring consistent reliability.
- **Future Feature Integration:** The architecture is designed with modularity in mind, leaving room to add new diseases, improved models, or enhanced features as user needs evolve.

In summary, the Skin Disease chain pipeline transforms raw image data into a valuable diagnostic tool, combining advanced image classification (ResNet18) with an empathetic, language-driven interface (LLM). By guiding users through image submission, ensuring medically responsible communication, and maintaining a scalable, monitored infrastructure, the pipeline provides a safe, informative, and flexible health support system.

1.2.4 Donna Chain MLOps Cycle:



Preprocess Data
Develop Model
Deploy
Monitor
Main prompt
Enclose extracted info into a string, fetched as a JSON file
OpenAI LLM
Sends email reminders of medication based on key values
SMTP protocol
docker
Register with Chatbot agent
Track model performance
Leave room for future feature integration and reproducibility
Monitor API health
Medication name
Dosage
Timing
User E-mail address
Extract relevant info
Date parser making the system dynamic
Guardrails
Conversation

Figure 5: Donna the Secretary MLOps cycle

The Donna Chain is a specialized component of the medical assistant chatbot designed to help users schedule and manage medication reminders. It extracts relevant information from conversations, dynamically schedules reminders, and sends them via email using the SMTP protocol. Below is a step-by-step breakdown of the Donna Chain workflow.

1. Preprocess Data

The first stage focuses on extracting and organizing relevant user inputs:

- **User Inputs:** The system gathers four key pieces of information:
  - Medication name.
  - Dosage (e.g., ”1 tablet”).
  - Timing (e.g., ”every day at 8 AM” or ”in 5 minutes”).
  - User email address for sending reminders.
- **OpenAI LLM:** The language model processes the user’s conversation to extract this information using a structured prompt. Guardrails are applied to ensure privacy, safety, and compliance with usage rules.
- **Dynamic Parsing:** Once the extracted information is formatted, the system employs a date parser (via the `dateparser` library) to convert the timing into specific schedules, making the system dynamic and adaptable to different time zones and user preferences.

2. Develop Model

This stage involves handling user conversations, extracting structured data, and preparing it for further processing:

- **Main Prompt:** The LLM prompt ensures that Donna:
  - Greets the user and offers help in setting reminders.
  - Collects the necessary data (medication name, dosage, timing, and email).
  - Confirms the schedule with the user in a friendly tone.
  - Avoids providing medical advice or unnecessary details.
- **Information Extraction:**
  - The extracted data is enclosed in a JSON format to ensure consistency and reusability.
  - The data includes fields like medication, dosage, timing, and email.
  - Unique IDs are generated for each prescription to prevent duplication.
- **Date Parsing and Validation:**

    - Timing is parsed using `dateparser` to compute the next reminder time.
    - The system ensures that reminders are scheduled for future times only, validating that the timing makes sense in the context of the current date and time.
    - Each prescription is tagged with a `frequency_seconds` value to define the interval for recurring reminders (e.g., daily).

3. Deploy
Once the reminders are set, the system moves to the deployment phase:

- **SMTP Protocol for Emails:**
    - Donna sends reminders to users via email using the SMTP protocol.
    - Emails contain the medication name, dosage, and instructions for the user to take their medication on time.
    - The SMTP credentials (e.g., Gmail username and password) are securely loaded from environment variables to ensure confidentiality.
- **Single Docker Container for the Entire Project:**
    - The entire chatbot project, including the Donna Chain, is packaged as a single Docker image.
    - This container encapsulates all components, ensuring consistency across environments and simplifying deployment.

4. Monitor
To ensure the system's reliability and scalability, monitoring tools are implemented:

- **Track Model Performance:**
    - The system evaluates how accurately the LLM extracts relevant information.
    - Any issues with parsing or user interaction are logged for future improvements.
- **Monitor API Health:**
    - The health and uptime of the chatbot API are continuously monitored to ensure uninterrupted service.
- **Leave Room for Feature Expansion:**
    - The architecture is designed to accommodate future enhancements, such as supporting recurring reminders at custom intervals or integrating with other notification methods (e.g., SMS or app notifications).

Key Technical Features

- **Singleton Pattern:** The Donna Chain is implemented as a singleton, ensuring that only one instance is active at any time. This avoids redundancy and ensures efficient resource usage.
- **Background Thread for Email Reminders:**
    - A dedicated thread checks pending reminders every minute.
    - Emails are sent automatically at the scheduled times, and the next reminder is computed for recurring prescriptions.
- **Guardrails for Safety and Privacy:**
    - Inputs are passed through a guard function to ensure compliance with the system's safety protocols.
    - Personal information is kept private, and reminders are scheduled only after explicit user confirmation.

In Summary
The Donna Chain provides a user-friendly and efficient solution for managing medication reminders. By leveraging OpenAI's LLM for natural language understanding, `dateparser` for dynamic scheduling, and SMTP for email delivery, the system ensures accurate and timely reminders while maintaining user privacy. By packaging the entire chatbot project, including the Donna Chain, as a single Docker image, the system ensures streamlined deployment and portability. The use of a modular and scalable architecture allows Donna to integrate seamlessly into the broader chatbot system and adapt to future needs.