华南师范大学
SOUTH CHINA NORMAL UNIVERSITY

Software Design and Architecture Assignment

Paradox AI Studio Tongue Diagnosis Final Report

Group： 1 **2025 / 12 / 22**

Topic	**Paradox AI Studio Tongue Diagnosis Final Report**			
	Name	**Student Number**	**Major**	**Email**
Member 1	闫弘宇	20223803065	Software Engineering	hongyu.yan@163.com
Member 2	盖乐蕾	20233802023	Software Engineering	3641526296@qq.com
Member 3	陈思凡	20223803053	Software Engineering	2652212759@qq.com
Member 4	刘思远	20223803045	Software Engineering	1354828493@qq.com
Member 5	王思蕴	20233802021	Software Engineering	1051158053@qq.com
Supervisor	Faizan Khan		**Words**	15000+

Abstract: This report presents a preliminary machine-learning-based traditional Chinese medicine (TCM) tongue diagnosis system, establishing an end-to-end intelligent analysis and application pipeline from problem background to technical feasibility. We developed a prototype system that adopts a cascaded pipeline consisting of YOLOv5 for localization, SAM for segmentation, and SE-ResNet for classification, combined with a FastAPI backend and an Electron + Vue client. Structured diagnostic reports are generated using a locally deployed large language model via Ollama. To ensure engineering quality, the system implements a front-end/back-end separated architecture, JWT-based authentication with TLS encryption, data persistence using SQLite and SQLAlchemy, public access via Nginx/Caddy and FRP, CI/CD driven by Jenkins, and multi-level testing using pytest, Cypress, and Apifox. Experimental results show that the system is functional, while improvements are still needed in data quality, cross-device robustness, and medical validation. Future work will focus on knowledge distillation and semi-supervised learning, on-device acceleration, enhanced image acquisition guidance and historical analysis, and the construction of a “feature–symptom” knowledge base.

Acknowledgments

This report represents a staged outcome of the course project and has benefited greatly from the guidance and timely feedback provided by the course instructor, Faizan Khan, in terms of topic selection, methodology, and engineering practice. During the processes of requirement analysis, technical route selection, implementation, and testing, the instructor's suggestions and annotations significantly improved both the technical depth and the writing quality of this project. We also thank our classmates for their cooperation and constructive feedback during discussions and system integration. Any remaining shortcomings in this report are solely the responsibility of our team.

This project relies heavily on the open-source ecosystem and community resources. We sincerely thank the Vue community, as well as Element Plus and Tailwind, for their support in front-end development; FastAPI, Pydantic, Uvicorn, SQLAlchemy, and SQLite for their stable and efficient backend and persistence support; and PyTorch for providing the training and inference framework. Special thanks are extended to YOLOv5, the Segment Anything Model (SAM), ResNet, and the Squeeze-and-Excitation (SE) mechanism for their contributions to computer vision and classification performance. For deployment and operations, we thank FRP, Nginx, Caddy, and Jenkins; for testing and collaboration, pytest, Cypress, Apifox, Postman, Git, and GitHub. We also acknowledge Ollama and deepseek-r1-14b for their support in report generation and interactive explanations.

Regarding data and domain knowledge, we thank the open-source community, publicly available literature, and the students and volunteers who assisted with tongue image annotation and quality assessment. As the team does not have a formal medical background, we are grateful to professionals in traditional Chinese medicine for their informal guidance, which helped us better understand the importance of a validated "feature–symptom/constitution" mapping and expert verification loop. In future work, we will continue to improve data compliance, privacy protection, and ethical review, while strictly adhering to the licenses and usage boundaries of all open-source component.

Contents

Part01: Introduction

1.1 Problem Statement

1.1.1 Reailty

At present, traditional Chinese medicine tongue diagnosis, as a valuable non-invasive diagnostic method, faces two major challenges in modern applications. First, tongue diagnosis relies heavily on the long-term clinical experience of practitioners, resulting in strong subjectivity and difficulties in standardization and large-scale deployment, which significantly limits service availability. Second, transitioning from manual interpretation to automated intelligent analysis introduces substantial technical challenges, including robust tongue region segmentation, effective modeling of fine-grained features, and the encapsulation of complex algorithms into user-friendly applications. To date, no unified or mature solution has been established.

1.1.2 Ideal

The ideal solution is to develop an intelligent system capable of bridging these gaps by transforming traditional tongue diagnosis knowledge into computable and executable algorithmic models, enabling automation and standardization of the diagnostic process. Users, regardless of their medical background, should be able to upload tongue images through an intuitive application and receive objective and reliable preliminary health assessments. Such a system can serve as an assistive tool for practitioners to improve diagnostic efficiency, as well as an intelligent health management assistant for the general public, thereby significantly enhancing the accessibility and inclusiveness of traditional Chinese medicine services.

1.1.3 Proposal

This project aims to deliver an end-to-end machine-learning-based tongue diagnosis solution through systematic software engineering and machine learning research. The core proposal is to construct a multi-model collaborative intelligent analysis pipeline and encapsulate it within a user-friendly, cross-platform application. Specifically, a hybrid strategy combining YOLOv5 and the Segment Anything Model is adopted for high-quality tongue localization and segmentation. A ResNet-based classification network enhanced with Squeeze-and-Excitation attention mechanisms is employed for multi-dimensional feature classification. The complete workflow, from data input to result presentation, is realized through a FastAPI backend and an Electron + Vue

front-end.

1.1.4 Consequences

The successful implementation of this project addresses key challenges in the modernization of traditional tongue diagnosis. From a technical perspective, it validates the effectiveness of a cascaded multi-model architecture for complex medical image tasks and provides valuable practical insights into model training and optimization. From an application perspective, it transforms sophisticated AI technologies into accessible public services, bridging the gap between professional medical expertise and everyday health needs, and demonstrating the significant social value of applying artificial intelligence to traditional medicine.

1.2 Background & Literature Review

1.2.1 Current Situation of Overseas Research

At present, there are relatively few overseas studies that are strongly related to the research direction of this project. Most existing research focuses more on the application of computer vision techniques in various modern medical fields, with limited efforts to directly associate these techniques with tongue image analysis or apply them to the domain of traditional Chinese medicine. Jianpeng Zhang et al.[1] proposed the DoDNet model to address the problem of partially annotated multi-organ and tumor segmentation in abdominal CT scans. DoDNet is a single encoder‐decoder network with dynamic heads. Experimental results show that, benefiting from task encoding and dynamic filter learning, DoDNet not only achieves the best overall performance on seven organ and tumor segmentation tasks, but also demonstrates faster inference speed than other competing methods.

Wei Ji et al.[2] pointed out that in medical image analysis, a typical practice is to collect multiple annotations, each provided by different clinical experts or raters, in order to reduce potential diagnostic errors. Meanwhile, from the perspective of computer vision practitioners, the common approach is to obtain ground-truth labels through majority voting or by simply selecting annotations from a preferred rater. However, this process often overlooks the rich information about agreement or disagreement contained in the original multi-rater annotations.

To address this issue, they proposed a new model, MRNet. First, an expertise-aware inferring module (EIM) is designed to embed the expertise levels of individual raters as prior knowledge, thereby forming high-level semantic features. Second, the method is able to reconstruct multi-rater rankings from coarse predictions and further exploit agreement or disagreement cues among raters to improve segmentation performance.

Experimental results show that MRNet achieves superior performance compared with state-of-the-art methods, and also verify its effectiveness and applicability across a wide range of medical segmentation tasks.

Qichao Tang et al.[3] investigated a deep-learning-based tongue detection method, using the MobileNetV2[4] for feature extraction, which is capable of accurately detecting the tongue from facial images. However, the proposed tongue detection model was not further extended or integrated with real-world clinical cases.

In the above studies, the focus is primarily on the application of computer vision techniques in the field of medical image analysis. More generally, the current state of overseas research tends to emphasize the proposal and construction of more general-purpose and efficient deep learning models, followed by analysis and comparison of these models to identify those with superior performance and efficiency under the context of traditional Chinese medicine tongue diagnosis. This trend is of significant importance to the present research project.

In the early 21st century, feature extraction was largely based on researchers' domain knowledge, such as the Scale-Invariant Feature Transform (SIFT)[5]. Features designed in this manner are referred to as handcrafted features. However, handcrafted features are not always optimal, as they are extracted using algorithms designed according to researchers' prior knowledge, and such feature extraction methods are inherently limited by the designers' knowledge systems.

In the 2010s, deep learning emerged and underwent rapid development. Deep learning algorithms greatly simplified the process of manual feature extraction, enabling models to automatically perceive and extract features. In 2014, Girshick et al. [6] proposed the Region-based Convolutional Neural Network (R-CNN) for object detection. This method uses selective search to generate region proposals, extracts features from each proposal using a CNN, and performs classification using a Support Vector Machine (SVM)[7].

In 2015, Kaiming He et al.[8] proposed the ResNet residual network, which enables the training of very deep neural networks and addresses the problems of vanishing and exploding gradients in traditional deep networks. ResNet can be used to learn abstract features from tongue coating images. By stacking deep residual blocks, the network is able to capture image features at different levels, thereby better representing subtle variations in tongue coating characteristics. In the same year, Olaf Ronneberger et al.[9] proposed the U-Net image segmentation model. U-Net was specifically designed for biomedical image segmentation tasks, particularly for precise segmentation of cells and organs in medical images. Compared with other segmentation models, U-Net is more suitable for small datasets and medical image segmentation tasks. Although U-Net provides strong segmentation capabilities, it still requires a more accurate image localization model beforehand to assist in achieving

more precise segmentation results.

In 2016, Joseph Redmon et al.[10] proposed the YOLO model, which stands for “You Only Look Once.” This name reflects the core design philosophy of the algorithm: object detection and localization can be completed through a single forward pass. YOLO formulates object detection as a regression problem, directly outputting object categories and bounding box coordinates in one step. Compared with traditional two-stage object detection methods (such as R-CNN), YOLO adopts a simpler single-network structure. In addition, YOLO introduces the concept of Anchor Boxes to handle objects of different sizes and aspect ratios, enabling the model to better adapt to targets with varying shapes and scales.

In 2023,Kirillov, A et al.[11] proposed the Segment Anything Model (SAM), as the name suggests, aiming to “segment anything.” The design of SAM mainly relies on three key components: (1) Task: promptable segmentation tasks that enable zero-shot generalization; (2) Model: the model architecture; and (3) Data: datasets that support both the tasks and the model. Trained on millions of images and over one billion masks, SAM is capable of producing valid segmentation masks for any given prompt. This represents an important introduction of self-attention mechanisms into the field of computer vision.

Through the review of overseas research, it can be observed that the field of computer vision has undergone continuous innovation and development, providing advanced solutions for medical image analysis and object detection. From Scale-Invariant Feature Transform (SIFT) to the rise of deep learning, models such as R-CNN and ResNet have introduced more efficient approaches to object detection and image classification tasks. Image segmentation has also been significantly enhanced by powerful models such as U-Net and YOLO. These overseas research achievements provide rich technical support for intelligent tongue diagnosis in traditional Chinese medicine and lay a solid foundation for the development of this field. As a result, in recent years, an increasing number of studies have focused on applying deep learning techniques to modern medical domains, particularly medical imaging. By drawing on and integrating these advanced computer vision techniques, it is expected that more significant progress can be achieved in the automated analysis and intelligent diagnosis of traditional Chinese medicine tongue images.

1.2.2 Current Situation of Domestic Research

Compared with overseas studies, domestic research places greater emphasis on the application of these models, with a stronger tendency to integrate deep learning models with traditional Chinese medicine theory and apply deep learning to specific tongue diagnosis scenarios. The goal is to train models that can be practically used in concrete traditional Chinese medicine tongue diagnosis tasks.

黄恩铭 et al.[12] focused on investigating the relationship between “constitution types and tongue images.” They employed TensorFlow-based deep learning techniques to perform precise segmentation of tongue images and extract color, shape, and texture features of the tongue body, enabling automatic identification of potential traditional Chinese medicine constitution types, including balanced constitution, qi-deficiency, yin-deficiency, yang-deficiency, phlegm-dampness, damp-heat, blood stasis, and qi-stagnation. Ultimately, they developed a WeChat mini-program that allows users to upload tongue images, with the application automatically identifying constitution types and generating corresponding reports and recommendations.

江涛[13] aimed to establish an intelligent tongue diagnosis technique based on deep learning and to promote its application in clinical tongue diagnosis, thereby providing technical support for traditional Chinese medicine diagnostic systems centered on tongue examination.

Specifically, Mask R-CNN was first employed to automatically segment tongue image boundaries and perform pixel-level precise segmentation of the tongue region, establishing a high-quality automatic tongue region recognition and segmentation model. Subsequently, models such as ResNet were used to efficiently classify four tongue body colors (light red, pale white, red, and purplish) and three tongue coating colors (white, yellow, and gray-black). Faster R-CNN was then applied to detect seven types of tongue morphology and texture features, including tooth marks, fissures, prickles, petechiae, greasy coating, peeled coating, and rotten coating, achieving lightweight multi-task fused intelligent batch analysis of tongue images. Finally, Spearman analysis and complex network analysis were used to explore the relationships between tongue image indicators and clinical disease indicators. For common conditions in health examination populations, such as metabolic syndrome, non-alcoholic fatty liver disease, and sub-health status, multiple linear regression, logistic regression, and various machine learning methods (SVM, Naive Bayes, Random Forest, ResNet, etc.) were applied to investigate the relationships between tongue image features and disease diagnosis or health status. The results indicate that deep learning methods enable quality control of tongue images, pixel-level segmentation of tongue regions, and classification of tongue color, morphology, and texture, thereby achieving preliminary intelligent tongue image diagnosis.

Changzheng Ma et al.[14] proposed a machine-learning-based screening model for precancerous gastric lesions (PLGC). They applied machine learning methods by using the YOLOv5 model for image segmentation and the ResNet50 model to construct an image classification framework. Statistical analysis was then conducted using the Python Scikit-learn package, revealing potential correlations between tongue image features and PLGC. This study also presented the first deep learning screening model for PLGC based on tongue images, named AITongue.

Lintai Wu[15] utilized convolutional neural networks to extract features from tongue

images for the diagnosis of diabetes mellitus (DM). To address the issue of limited data, they first applied parameter transfer methods from transfer learning. In order to embed the model into portable diagnostic devices, the SqueezeNet Architecture[16] was adopted for diabetes image detection. Model performance was evaluated in terms of accuracy, sensitivity, and specificity. The results demonstrate that this method performs well in DM diagnosis.

牛富泉[17] proposed a deep-learning-based traditional Chinese medicine tongue image classification model and further developed a web-based auxiliary tongue diagnosis platform based on this model, providing remote diagnostic assistance for physicians and patients.

To address issues such as low segmentation accuracy, poor color correction, and low tongue feature recognition accuracy in traditional models, this study proposed a pyramid pooling tongue image segmentation algorithm based on a pyramid scene parsing network and ResNet101, a two-stage tongue image color correction algorithm combining subjective and objective methods based on AlexNet, and a tongue feature recognition algorithm based on a squeeze-and-expansion network. Comparative experiments with other algorithms showed that the proposed methods achieved superior performance. However, the authors also noted that the study did not fully consider multi-scale feature pooling capabilities, which may result in extracted tongue features that do not fully represent global characteristics.

Zongrun Li et al.[18] aimed to establish an end-to-end deep learning network for intelligent tongue image analysis based on traditional Chinese medicine tongue diagnosis images. Based on the data characteristics of clinical trial indicators dominated by continuous variables, a dual-stream architecture was adopted to construct the intelligent tongue analysis network.

In this study, a U-Net-based tongue segmentation model at the network front end was used to segment tongue target regions from original images. After segmentation, a ResNet network was employed to extract feature vectors from the tongue regions. On the day of image acquisition, convolutional operations in the ResNet network were used to extract fused feature vectors from blood pressure data. Through this process, two sets of tongue features and fused features were successfully obtained. Based on the analysis of blood pressure data, tongue images, and their fused data at the network output, four regression methods—Random Forest (RF), AdaBoost, Gradient Boosting Regression Trees (GBRT), and Support Vector Regression (SVR)—were applied to predict the Mean Absolute Error (MAE). The results indicate that multimodal data fusion is an important approach for uncovering the clinical value of traditional Chinese medicine tongue images.

Overall, compared with overseas research, domestic studies place greater emphasis on the application of deep learning models in real-world traditional Chinese medicine

tongue diagnosis scenarios. By exploring specific disease diagnosis tasks such as diabetes and precancerous gastric lesions, as well as tongue image classification and auxiliary remote diagnosis, these studies demonstrate the potential value of deep learning in traditional Chinese medicine tongue diagnosis. Domestic researchers actively adapt and improve existing models to better align with traditional Chinese medicine tongue diagnosis requirements, thereby enhancing model performance. These studies provide strong support for the development of intelligent tongue coating diagnosis projects.

1.3 Stakeholders

The stakeholders of this project include the following categories.

First are end-users, including general users who hope to obtain convenient and low-cost preliminary health assessments, as well as traditional Chinese medicine practitioners and health consultants with some knowledge of traditional Chinese medicine but who wish to improve efficiency with the help of intelligent tools; they focus on usage barriers, result readability, privacy, and data security.

Next are the parties related to medical and health services, such as traditional Chinese medicine clinics, health management institutions, and education and training institutions. They expect the system to bring value in terms of standardization, reusability, and decision support, and emphasize compliance boundaries and risk warnings.

In addition, there are internal stakeholders within the project team, including Product Managers and Architects, Front-End and Back-End Developers and Algorithm Engineers, Testing and Documentation Staff, as well as Operations and Security Engineers. Meanwhile, Data Annotation and Research Partners are also crucial, as they provide data and method support for Model Training and focus on academic achievements and reproducibility.

Finally, the platform, open-source communities (such as Ultralytics, SAM, FastAPI, Ollama, etc.), and their license maintainers are key external stakeholders. They focus on our compliance with open-source licenses, secondary distribution and commercial use compliance, as well as continuous follow-up on community norms and adoption of feedback. These requirements directly impact component selection, source code open strategy, dependency version management, and the usage boundaries of third-party assets.

1.4 Scope of the Software

The scope of this software is positioned as a "preliminary health assessment tool for

traditional Chinese medicine tongue image based on Machine Learning", emphasizing the combination of intelligent analysis and popular science explanations, and clearly stating that it does not replace professional clinical diagnosis.

The system supports desktop clients (Electron, Windows x86-64 and macOS arm64) and web clients, providing functions such as registration and login, creation of diagnostic sessions, tongue image upload, automated analysis pipeline (YOLOv5 localization → SAM segmentation → SE-ResNet50 classification → four-dimensional feature vector generation), local LLM (invoking deepseek-r1-14b via Ollama) to generate readable diagnostic reports and continuous conversations, as well as persistence and retrieval of historical sessions and chat records.

The backend uses FastAPI as its core, adopts JWT authentication and TLS encrypted transmission (terminated by Nginx/Caddy), uses SQLite as the database and abstracts it through SQLAlchemy to support future migrations; the public network entry implements a secure access link through reverse proxy and FRP intranet penetration. It includes basic exception handling and retry mechanisms, end-to-end performance optimization, basic front-end interaction experience and status prompts, as well as a pluggable architecture design for algorithm modules.

Clinical diagnosis and treatment recommendations, mobile native applications, video/multi-frame tongue image analysis, real-world medical data networking and third-party medical system integration, doctor work flow management, fee settlement and risk control, strong compliance review and medical device registration, as well as high-level privacy enhancement technologies (such as FL and differential privacy, to be further enhanced in the future) are explicitly excluded from the scope.

1.5 Technologies and Tools Used

Frontend: Electron, Vue3, Element Plus, Tailwind, Pinia, Axios

Backend: FastAPI, Uvicorn, Pydantic, SQLAlchemy, SQLite

AI Model Inference: PyTorch, YOLOv5, SAM, SE-ResNet, Ollama, deepseek-r1-14b

Operation and Maintenance: Nginx/Caddy, FRP, Docker

Testing: pytest, Cypress, Apifox, Postman

Development and Collaboration: Git, GitHub, Jenkins

Part02: Requirement Documents

This document aims to clearly define the functional and non-functional requirements of the Paradox AI Intelligent Tongue Diagnosis System. The system is a desktop application that integrates traditional Chinese medicine (TCM), deep learning techniques, and large language models (LLMs), and is designed to provide users with a convenient and intelligent preliminary tongue-based health assessment tool.

2.1 Functional Requirements

☐ **Use Case Overview and Participants:**

■ **Main Participants**: User (End User)

■ **System Participants**: Frontend Client (Electron/Web), Backend Service (FastAPI), Authentication Service (JWT/Refresh Token), Model Pipeline (YOLO → SAM → SE-ResNet), LLM Service (Ollama/deepseek-r1-14b), Database (SQLite/SQLAlchemy), Reverse Proxy (Nginx/Caddy)

☐ **Use Case List**:

☐ Register Account (FR-UAM-01)
☐ Login Account (FR-UAM-02)
☐ Create Diagnostic Session (FR-CORE-01)
☐ Upload tongue image (FR-CORE-02)
☐ Automatic Tongue Image Analysis (FR-CORE-03)
☐ Generate and display the diagnostic report (FR-CORE-04)
☐ Continuous conversation within session (FR-CORE-05)
☐ Historical Conversation List Display (FR-HIST-01)
☐ View historical conversation details (FR-HIST-02)

2.1.1 User Account Management (FR-UAM)

2.1.1.1 FR-UAM-01: User Registration

☐ **Participants**: User, Backend Service, Database, Reverse Proxy
☐ **Precondition**: The user has opened the Client and can access the registration page; the network is available.
☐ **Trigger condition**: User clicks "Register" and submits email and password

☐ **Main success scenario**:

①The front end validates the email format and password complexity (minimum length, character requirements)

②The front end calls POST /api/user/register to submit registration information.

③The backend checks that the email has not been registered, performs password salted hashing, and persists the user.

④The backend returns registration success.

⑤The front end navigates to the login page and prompts "Registration successful, please log in".

☐ **Exception/Alternative Scenarios**:

① A1 Email format is invalid: The front end blocks submission and prompts "Email format error"

②A2 Password does not meet complexity requirements: Prompt "Password is too simple/insufficient length"

③ A3 email has been registered: the backend returns 101, and the frontend prompts "Email has been registered"

④A4 Network/Service Exception: Displays "Service is temporarily unavailable, please try again later"

☐ **Postcondition**: A new user record is added to the database; automatic login is disabled; the user is pending login.

2.1.1.2 FR-UAM-02: User Login

☐ **Participants**: User, Backend Service, Database, Authentication Service, Reverse Proxy

☐ **Precondition**: The user has completed registration; the network is available.

☐ **Trigger condition**: The user submits an email address and password on the login page.

☐ **Main success scenario**:

①The front end calls `PUT /api/user/login` to submit credentials.

②The backend verifies that the email exists and the password matches.

③The backend issues short-lived Access Tokens and long-lived Refresh Tokens.

④The front end saves the Token (secure storage) and redirects to the home page.

☐ **Exception/Alternative Scenarios**:

① B1 Email or password error: The backend returns 102, and the frontend prompts "Email or password is incorrect"

②B2 account does not exist: Prompt "Account not registered"

③B3 Network or Service Exception: Prompt "Login failed, please try again later"

☐ **Postcondition**: Session established, subsequent requests carry Bearer Token; login state persisted

2.1.2 Core process of intelligent tongue diagnosis (FR-CORE)

2.1.2.1 FR-CORE-01: Initiate a diagnostic session

☐ **Participants**: User, Backend Service, Database, Reverse Proxy
☐ **Precondition**: The user has logged in; enter the diagnostic page.
☐ **Trigger condition**: The user enters the record name and clicks "Add".

☐ **Main success scenario:**

①Front-end verification ensures the name is not empty

②The front end calls `POST /api/model/session`

③The backend creates a new session for the user, generates a sessionId, and persists the session metadata.

④The front end inserts a new record into the left-side conversation list and sets it as the active conversation.

☐ **Exception/Alternative Scenarios:**

①C1 name is empty: prompt "Please enter the record name"

②C2 Token Expired: Trigger Test Case 10 (Token Refresh), and if the refresh fails, return to the login page

③C3 service exception/database error: Prompted with "Failed to create session"

☐ **Postcondition**: A new session is added to the database; the front end activates the session and awaits image upload.

2.1.2.2 FR-CORE-02: Tongue image upload

☐ **Participants**: User, Frontend, Backend Service, Reverse Proxy
☐ **Precondition**: The user has logged in and activated a session; there is a picture file to be uploaded.
☐ **Trigger condition**: The user selects a file or drags an image in the upload area

☐ **Main success scenario:**

①Front-end verifies file type and size (jpg/png, size limit)

② The front end submits file_data, name, and user_input to `POST /api/model/session` via `multipart/form-data`.

③The reverse proxy forwards requests, the backend receives and stores the image path, and marks the session as entering the "analyzing" state.

④The front-end displays "Uploading..." and then switches to the "Analyzing..." state.

☐ **Exception/Alternative Scenarios:**

①D1 Illegal file type/too large: The front end blocks submission and prompts "Only jpg/png are supported, with a size not exceeding 50MB"

②D2 upload interrupted or network anomaly: Prompt "Upload failed, click to retry"

③D3 Token Expired: Use Refresh Token to refresh the token; if it fails, return to the login page.

☐ **Postcondition**: The image is successfully stored and triggers the analysis process; the session content records the image metadata.

2.1.2.3 FR-CORE-03: Automated tongue image analysis

☐ **Participants**: Backend Service, Model Pipeline (YOLO → SAM → SE-ResNet), Database

☐ **Precondition**: Image upload is successful; session is in the "Analyzing" state.

☐ **Trigger condition**: The backend receives the upload completion event or queue task.

☐ **Main success scenario:**

①Backend starts an asynchronous task: YOLO locates the tongue body bounding box

②Based on the positioning area, call SAM for precise segmentation to obtain the tongue mask.

③ Use SE-ResNet50 to classify the segmented tongue images and generate four-dimensional feature vectors.

④Write ModelResults (feature vector, version, Confidence Level), and update the session status to "Analysis Completed - Pending Report"

⑤Notify the front end or obtain the progress through front-end polling.

☐ **Exception/Alternative Scenarios:**

①E1 Tongue not detected/Confidence Level too low: Return "Poor image quality or tongue not detected", recommend retaking the photo and allow retry

②E2 Model Service Unavailable/Insufficient GPU: Log the error and prompt "Analysis service is busy, please try again later"

③E3 data processing exception: Log the record and error code, and the front end prompts failure

☐ **Postcondition**: The database saves the analysis results and status; if successful, it proceeds to the report generation process.

2.1.2.4 FR-CORE-04: Generate and display diagnostic reports

☐ **Participants**: Backend Service, LLM Service, Database, Frontend

☐ **Precondition**: Analyze the generated four-dimensional feature vector; LLM

service is available

☐ **Trigger condition**: The backend calls the LLM after successful analysis; or the frontend requests report generation.

☐ **Main success scenario:**

① The backend passes the feature vector and necessary context into the LLM (Ollama deepseek-r1-14b).

② LLM generates popular tongue image diagnosis explanations and relevant traditional Chinese medicine knowledge.

③The backend returns to the front-end dialog box in streaming mode (character by character/paragraph by paragraph).

④ The backend and frontend simultaneously persist the report as a session message.

☐ **Exception/Alternative Scenario:**

①F1 LLM unavailable/timeout: Return the fallback message "Unable to generate report at the moment, please try again later" and retain the analysis results

②F2 report generation failed: record the error and prompt; allow regeneration

③F3 Token Expired: Use Refresh Token to Refresh Token

☐ **Postcondition**: Report messages are stored in Messages; session status is updated to "Report Generated".

2.1.2.5 FR-CORE-05: Continuous conversation and context understanding

☐ **Participants**: User, Frontend, Backend Service, LLM Service, Database

☐ **Precondition**: The user is logged in; the session exists and has at least a report or a feature vector; the LLM is available.

☐ **Trigger condition**: The user enters a question in the dialog box and clicks Send.

☐ **Main success scenario:**

①The front end calls `POST /api/model/session/{sessionId}` to submit user input.

②Backend aggregation context: session historical messages + feature vectors

③Call the LLM to generate answers and return them in a streaming manner

④The front end displays "AI is typing..." and shows it character by character.

⑤Backend and frontend persistent Q&A messages

☐ **Exception/Alternative Scenario:**

①G1 LLM Timeout/Busy: Prompt "AI is busy, please try again later"

②G2 context is too long: Backend executes message digest/truncation strategy and continues generation

③G3 network anomaly: Prompt and allow resending

④G4 Token Expired: Use Refresh Token to Refresh Token

☐ **Postcondition**: New messages are written to the database; session context is updated

2.1.3 Diagnostic History Management (FR-HIST)

2.1.3.1 FR-HIST-01: Historical record display

☐ **Participants**: User, Frontend, Backend Service, Database
☐ **Precondition**: The user has logged in; session records exist.
☐ **Trigger condition**: User enters the diagnostic page or clicks "History"

☐ **Main success scenario:**

①The front end calls `GET /api/model/session`

② The backend returns the session list (session_id, name, created_at) in descending order of time.

③The front end renders in the left list, with the latest at the top.

☐ **Exception/Alternative Scenario:**

①H1 No Conversation Record: Displays Empty State and Guidance Text

②H2 network/service exception: Prompt "Failed to obtain session"

③H3 Token Expired: Use Refresh Token to Refresh Token

☐ **Postcondition**: No data change; list status has been updated

2.1.3.2 FR-HIST-02: View history

☐ **Participants**: User, Frontend, Backend Service, Database
☐ **Precondition**: The user has logged in; a conversation has been selected.
☐ **Trigger condition**: The user clicks on a conversation record.

☐ **Main success scenario:**

①The front end calls `GET /api/model/record/{sessionId}`

② The backend returns the complete message history of the session (including images, reports, and Q&A).

③ The front end renders the conversation content and allows scrolling for viewing.

☐ **Exception/Alternative Scenario:**

①I1 session does not exist or has been deleted: prompt "Session does not exist"

②I2 data acquisition failed: prompt and allow retry

③I3 Token Expired: Use Refresh Token to Refresh Token

☐ **Postcondition**: No new data; the front end highlights the current session and is ready to continue the conversation.

2.2 Non-functional Requirements

2.2.1 Performance requirements (NFR-PERF)

2.2.1.1 NFR-PERF-01: Response time

①The response time for regular UI operations (page navigation, button clicks, etc.) should be less than 500 milliseconds.

②The response time of the user login/registration verification process should be less than 2 seconds.

③The end-to-end processing time from when a user uploads an image to when the AI starts returning the first analysis report should be controlled within approximately 40 seconds.

④For subsequent AI conversation responses, the time from when the user sends a question to when the AI starts returning an answer should be less than 10 seconds.

2.2.2 Availability requirements (NFR-USAB)

2.2.2.1 NFR-USAB-01: Usability

①The system interface should be simple and intuitive, conforming to the user habits of mainstream desktop applications.

②The operation paths of all core functions should be clear and straightforward, allowing new users to complete a full tongue diagnosis process without training.

③ The system should provide clear status indicators, such as "Uploading...", "Analyzing...", "AI is typing...", etc.

2.2.3 Reliability and Availability (NFR-REL/AV)

2.2.3.1 NFR-REL-01: System stability

①Under normal operation, the system should be able to run stably for a long time without crashing or exiting unexpectedly.

②The availability of core analysis services should reach 99% to ensure that user requests can be successfully processed.

2.2.3.2 NFR-REL-02: High Availability Entry

①The public network access entry of the system should adopt a primary-backup (Nginx/Caddy) reverse proxy mechanism to ensure that services remain available in the event of a primary node failure.

2.2.4 Compatibility requirements (NFR-COMP)

2.2.4.1 NFR-COMP-01: Client Compatibility

① The system shall provide a desktop client packaged based on Electron and ensure compatible operation on the following platforms:

- Windows (x86-64)
- macOS (arm64)

② Web applications should behave consistently across mainstream modern browsers, including but not limited to:

- Google Chrome (and other Chromium-based browsers)
- Safari (WebKit Kernel)

2.2.5 Security requirements (NFR-SEC)

2.2.5.1 NFR-SEC-01: Authentication and Authorization

①All API interfaces that require user identity authentication must be protected by a valid JWT.

②The system should be able to correctly handle the issuance, verification, and expiration logic of Tokens.

2.2.5.2 NFR-SEC-02: Data transmission security

①All communication between the Client and the server must be encrypted via TLS to ensure the confidentiality and integrity of data during transmission. The reverse proxy layer (Nginx/Caddy) is responsible for TLS termination.

2.2.5.3 NFR-SEC-03: Data persistence

①Sensitive data such as user account information, diagnostic records, and chat history must be persistently stored in the backend SQLite database.

2.2.6 Maintainability and Scalability (NFR-MAINT/EXT)

2.2.6.1 NFR-MAINT-01: Modularization architecture

① The system must adhere to a clear MVC layered architecture to ensure low coupling between the view layer, the control layer, and the model layer.

② Code should adhere to good programming practices, and key modules must have necessary comments and documentation.

2.2.6.2 NFR-EXT-01: The algorithm module is pluggable

① The system architecture should support the replacement or upgrade of core algorithm components (YOLO, SAM, ResNet+SE, LLM) without affecting other business logic. For example, in the future, it will be convenient to replace them with a segmentation model with better performance or a more powerful LLM.

2.2.6.3 NFR-EXT-02: Database is migratable

① The data access layer should be abstracted through ORM (SQLAlchemy) to ensure that the underlying database can be smoothly migrated from SQLite to other relational databases (such as PostgreSQL, MySQL) in the future.

Part03: Software Architecture

3.1 System High-Level Architecture Diagram

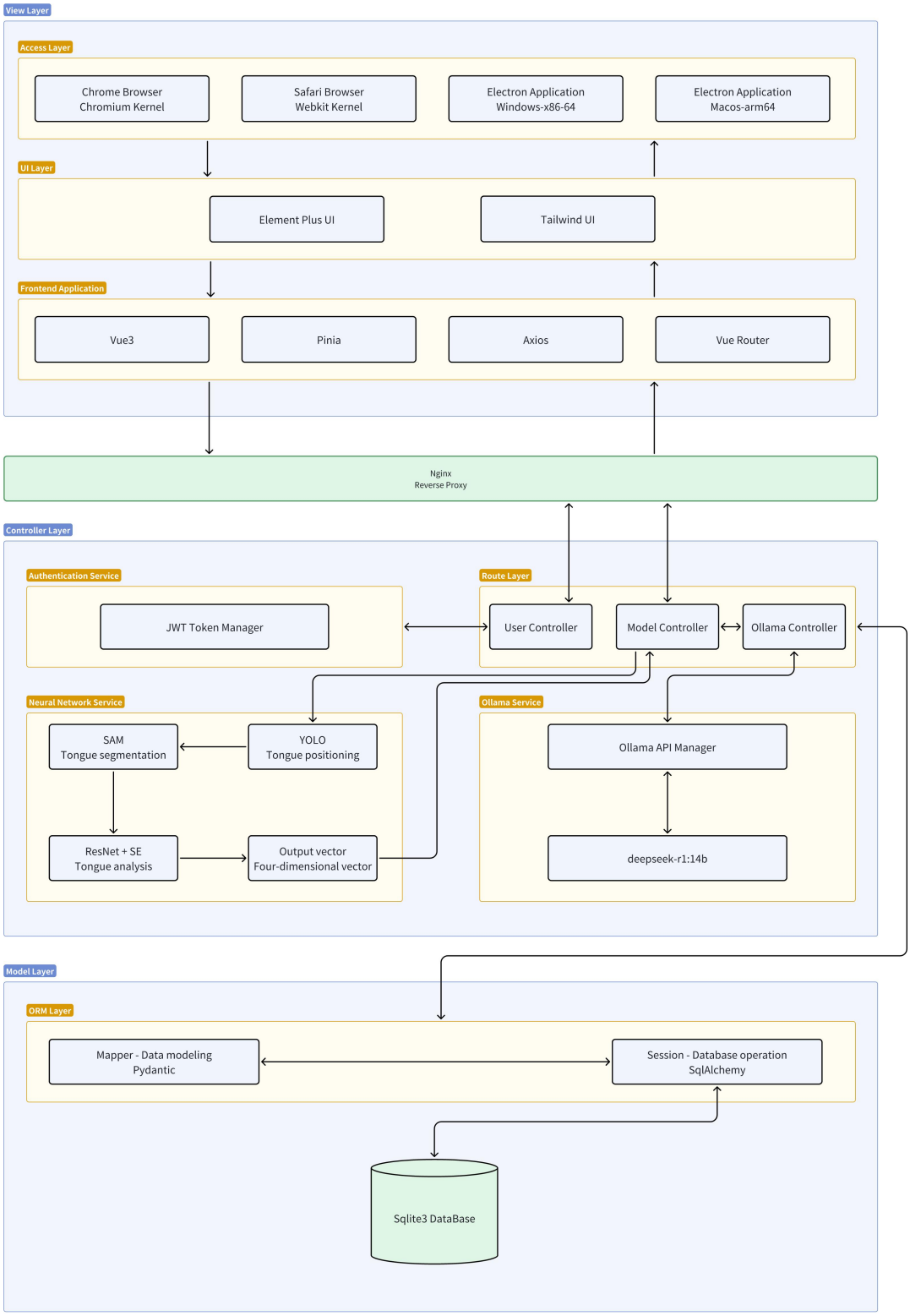
This application adopts the MVC architecture, with the overall structure divided into the view layer, control layer, and model layer, and the front-end and back-end connection is achieved through Nginx reverse proxy.

Next, we will provide a detailed introduction to our High-Level Architecture based on *Picture 3-1*.

The access end of the View layer supports Chrome based on the Chromium kernel, Safari based on the WebKit kernel, and desktop clients based on Electron (Windows x86-64 and macOS arm64). The UI layer uses Element Plus and Tailwind UI to build the interface. The front-end application is based on Vue3, using Pinia for global state management, Vue Router for routing management, and Axios for HTTP communication with the back-end; all front-end requests are routed and forwarded by Nginx.

The Controller layer consists of routing controllers (User Controller, Model Controller, Ollama Controller) and two types of business services: Authentication Service completes login authorization and token verification through the JWT Token Manager; Algorithm Service includes a deep learning pipeline for tongue image analysis, first using YOLO for tongue position localization, then using SAM for tongue body segmentation, followed by using ResNet+SE to extract features to form a four-dimensional output vector, which is uniformly scheduled by the controller and can be returned to the front end or used as input for subsequent inference. The Large Model Service encapsulates the call to the local large language model (deepseek-r1-14b) through the Ollama API Manager, providing inference capabilities for analysis, interpretation, and dialogue.

The Model layer implements data persistence and domain modeling, uses Pydantic for data model and mapping, uses SQLAlchemy to manage database sessions, and uses SQLite3 as the underlying storage; the controller interacts with the ORM to complete the reading and writing of user, model, and task data.



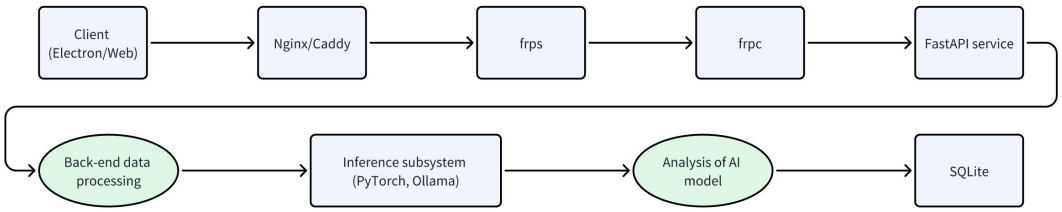
Picture 3-1: The High-Level Architecture Diagram

The main data flow of the application is shown in the *Picture 3-2*. The entire process starts with the Client (which can be an Electron-based desktop application or a web application), through which users initiate requests. These requests first pass through Nginx/Caddy, a high-performance web server or reverse proxy responsible for

handling client connections, load balancing, SSL termination, and serving static content, thereby enhancing the reliability and security of the system.

Next, the system introduced the frps (FRP Server) and frpc (FRP Client) components. This is commonly used to implement intranet penetration or tunnel proxy, allowing external Nginx/Caddy to securely communicate with the FastAPI service that may be deployed in a Virtual Private Cloud or behind a firewall. frps, as the server side, receives requests from Nginx/Caddy and forwards them to the FastAPI service via the frpc Client.

The FastAPI service is the core backend service of the system, handling business logic and API requests. From here, data enters a cyclic processing phase: the FastAPI service triggers Back-end data processing, which may involve data cleaning, transformation, or preprocessing. The processed data is then sent to the Inference subsystem, the intelligent core of the system, which uses frameworks such as PyTorch or Ollama to perform inference tasks of artificial intelligence models, such as image recognition, natural language processing, etc. The inference results will further go through the Analysis of AI model phase for post-processing, interpretation, or aggregation. Finally, these analysis results are persistently stored in the SQLite database, a lightweight embedded relational database commonly used for local data storage or small applications. This cycle ensures a closed loop of data processing, AI inference, and result analysis, with the final results stored for potential subsequent display or further business logic.



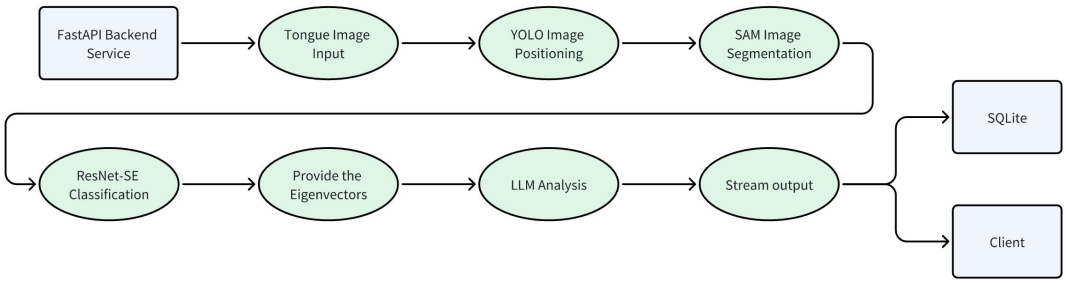
Picture 3-2: Data Flow Diagram of the Application

The data flow of the application's core algorithm is shown in the figure below. The entire process begins with the FastAPI Backend Service, which serves as the core business logic and API interface, responsible for receiving and coordinating subsequent image processing and AI analysis tasks. First, the system performs Tongue Image Input, i.e., receives the user's tongue image. Next, the image enters the first AI processing stage: YOLO Image Positioning, which uses the YOLO (You Only Look Once) Object Detection model to identify and precisely locate the tongue region in the image, preparing for subsequent analysis. Immediately following is SAM Image Segmentation, which uses the SAM (Segment Anything Model) to perform fine-grained image segmentation on the located tongue, thereby accurately separating the tongue from the background and ensuring that subsequent analysis focuses only on the tongue itself.

The segmented tongue image is then fed into the ResNet-SE Classification model for classification. ResNet-SE, a powerful deep learning network, is used at this stage to perform high-level recognition and classification of the tongue's visual features (such as color, texture, shape, etc.). The classification results or their internal feature representations are then used to Provide the Eigenvectors, which usually means extracting the most representative feature vectors of the image. These vectors can serve as high-dimensional data to more comprehensively describe the subtle features of the tongue.

These extracted feature vectors then enter the LLM Analysis (Large Language Model Analysis) stage. Here, the LLM may combine all previous visual analysis results to conduct deeper comprehensive understanding, inference, or generate diagnostic reports. The LLM can transform the complex outputs of AI models into human-readable and meaningful text descriptions or recommendations. Finally, the results of LLM analysis will be output in real-time or in batches via Stream output.

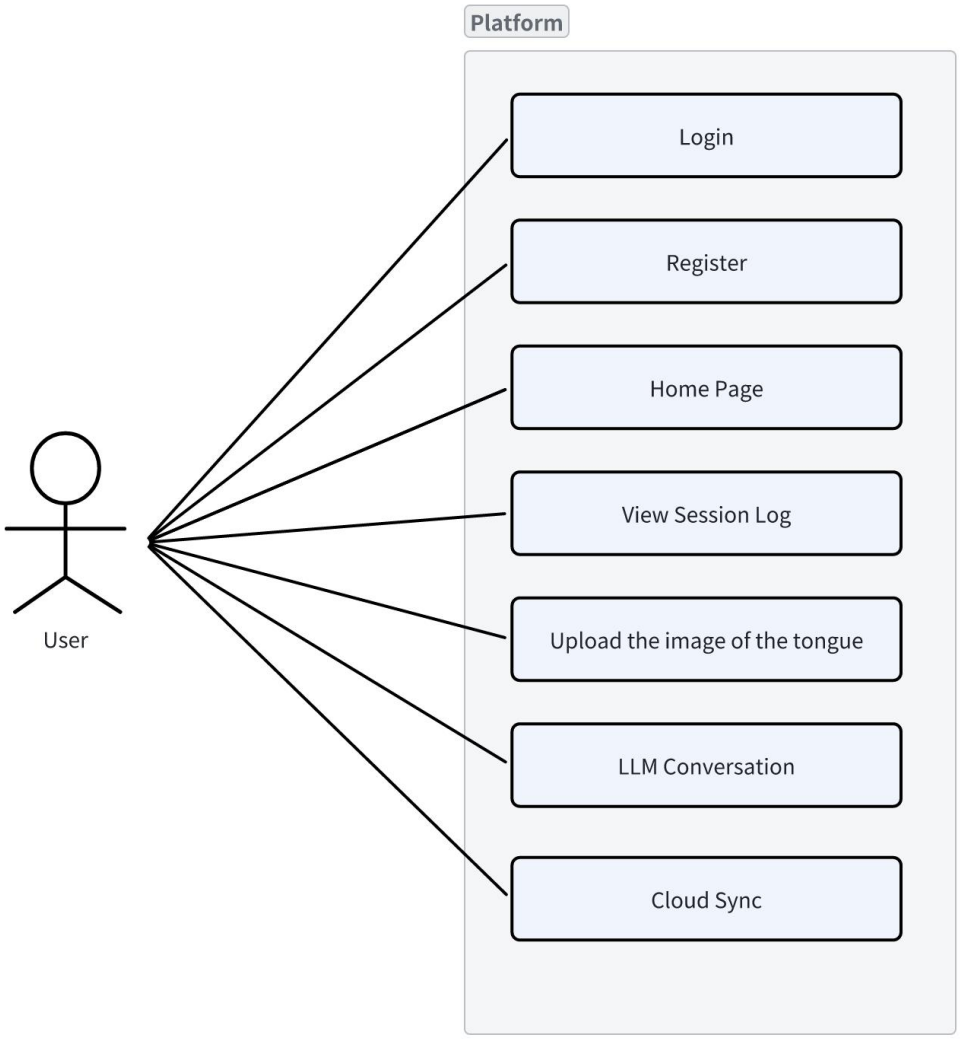
These output data will be sent to two destinations simultaneously: a portion of the data is persistently stored in the SQLite database for recording, subsequent querying, or offline analysis; the other portion of the data is transmitted back to the Client in real-time for users to view or further interact with, thus completing the closed loop from image analysis to result presentation.



Picture 3-3: Data Flow Diagram of the Core Algorithm

3.2 UML Diagrams (4+1 View)

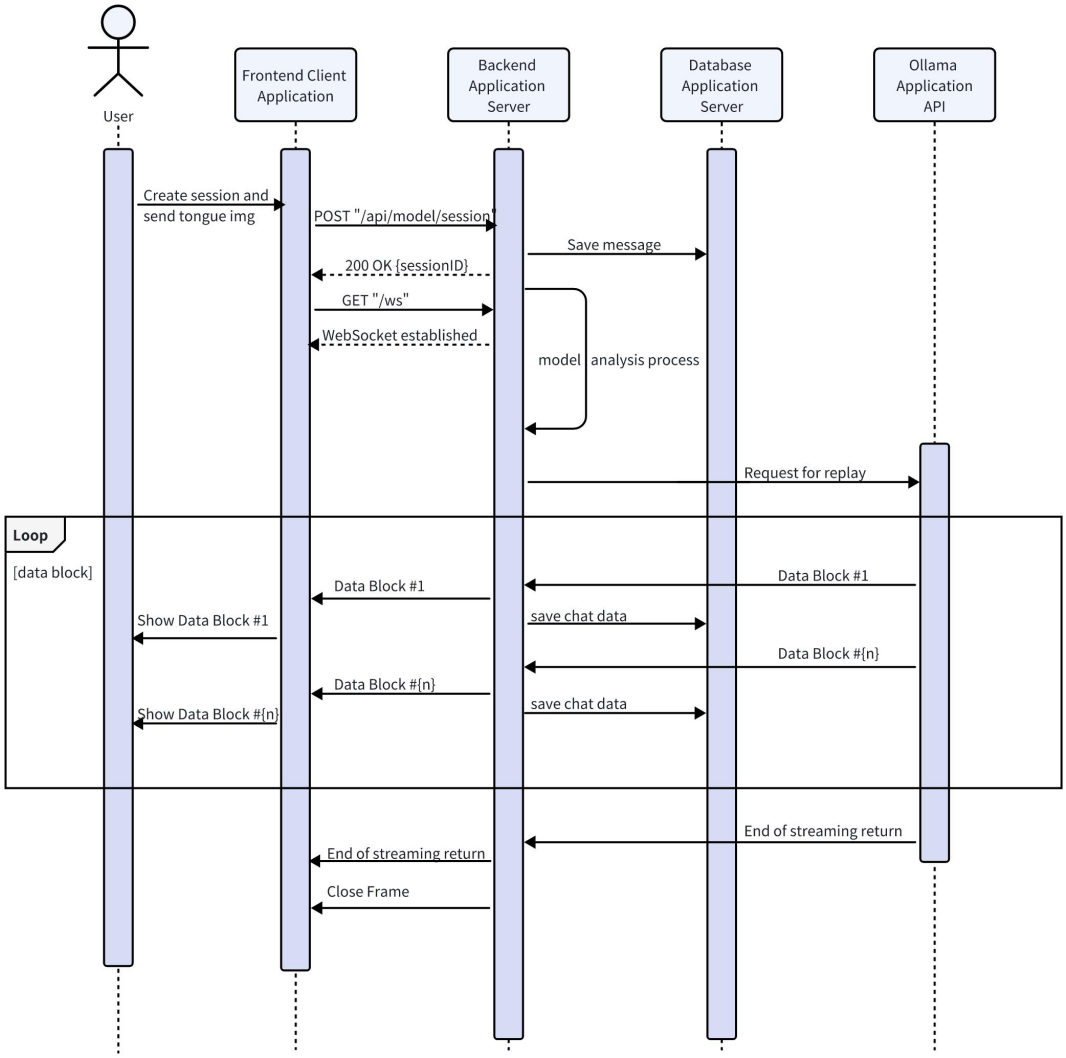
3.2.1 The Scenario View (User Interface View)



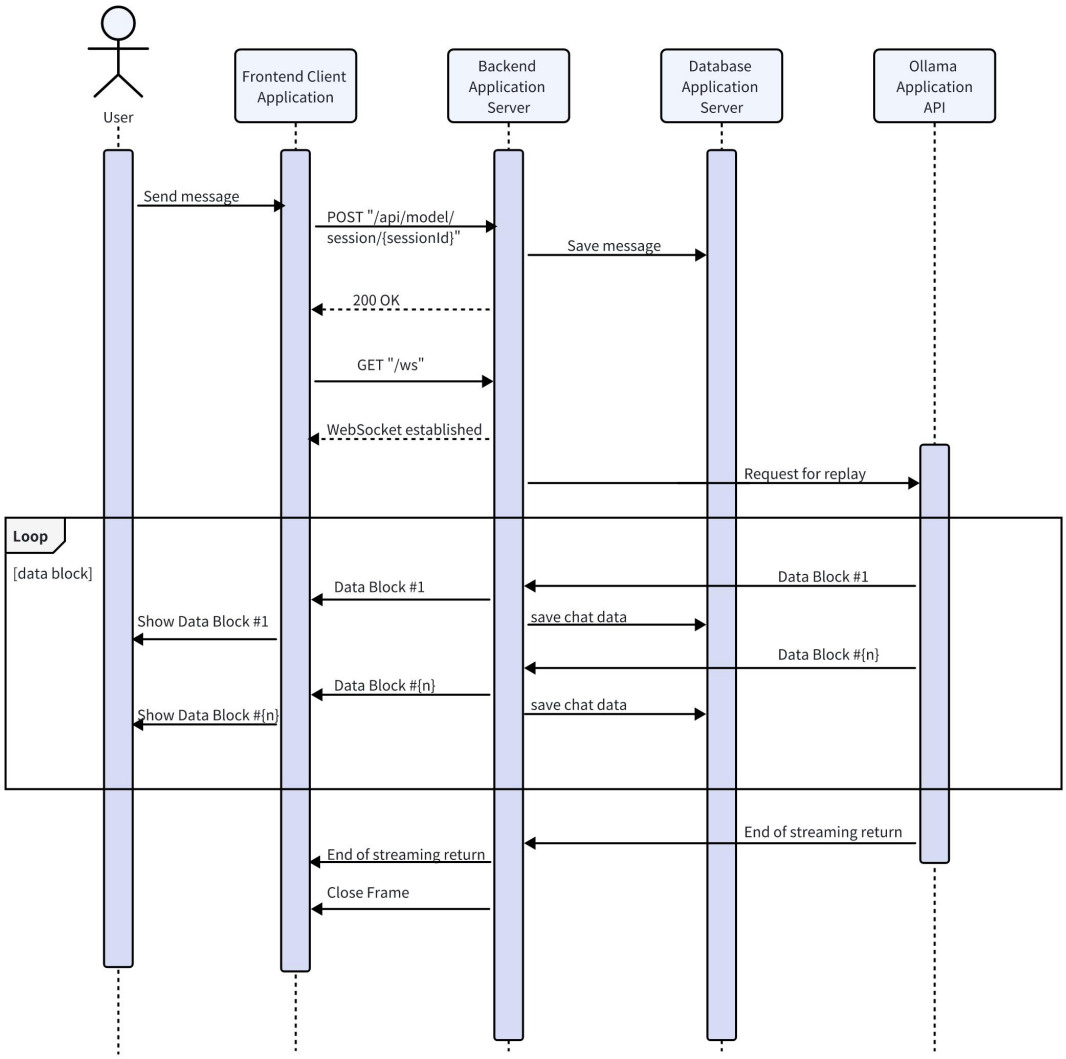
Picture 3-3: The Use Case Diagram

3.2.2 The Logical View

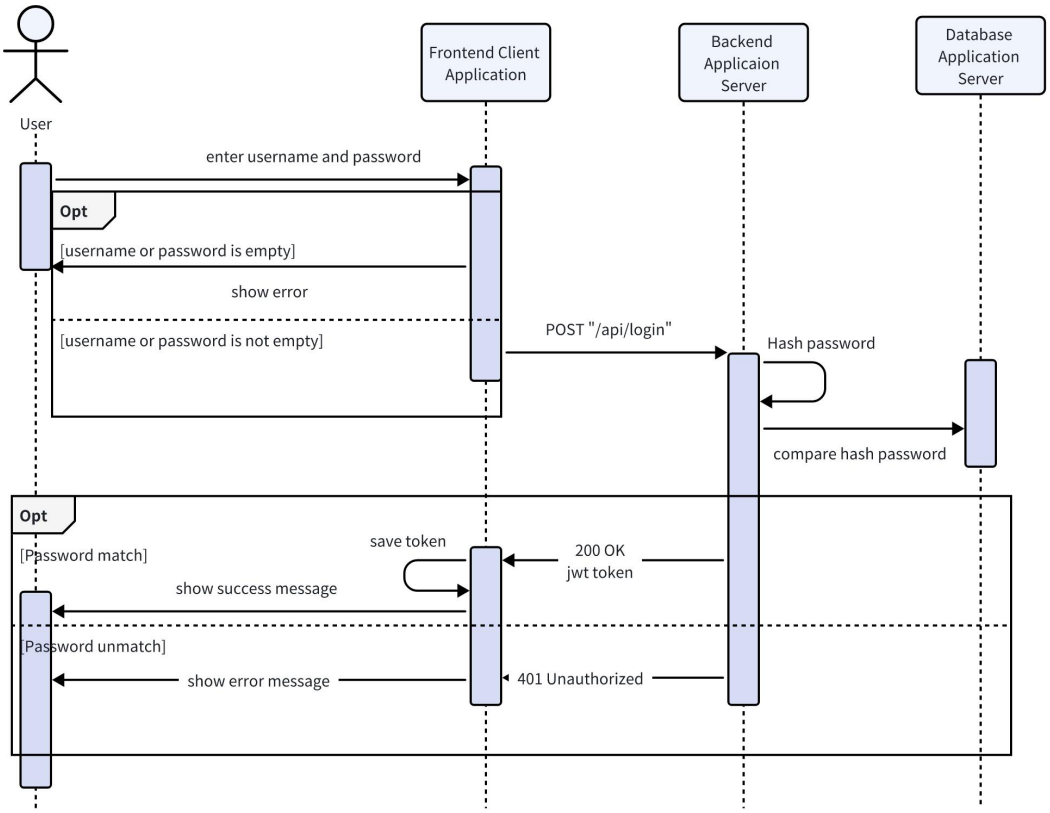
3.2.2.1 Sequence Diagram



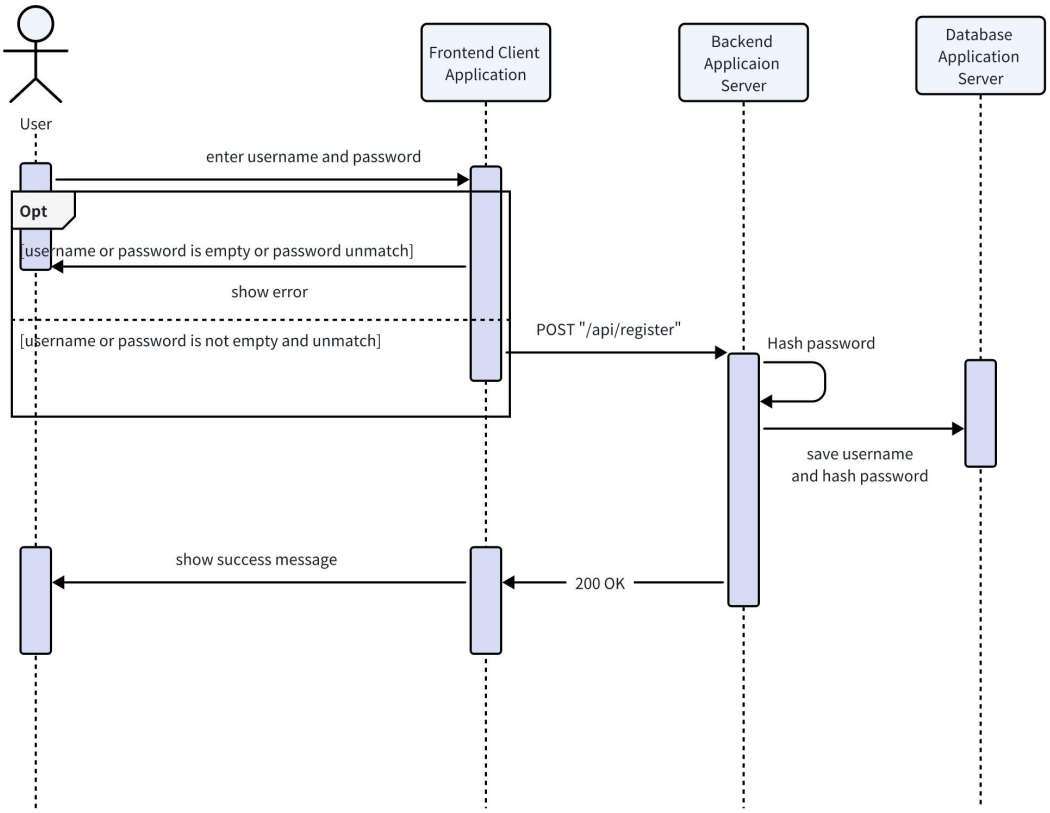
Picture 3-4: Sequence diagram for Automated tongue image analysis



Picture 3-5: Sequence diagram for chating

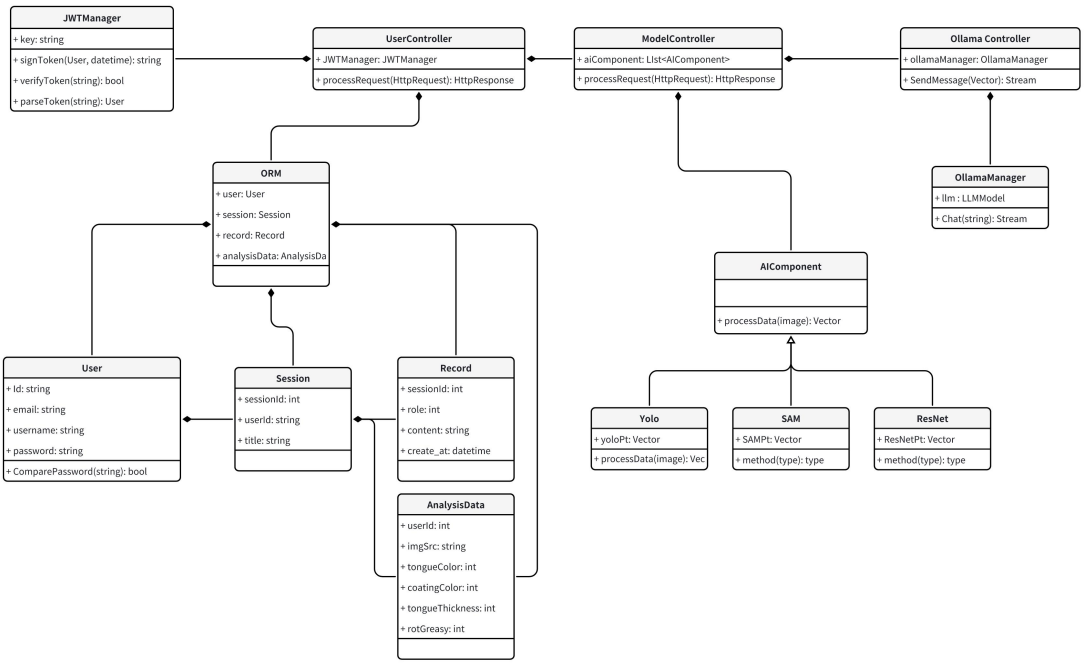


Picture 3-6: Sequence diagram for Login



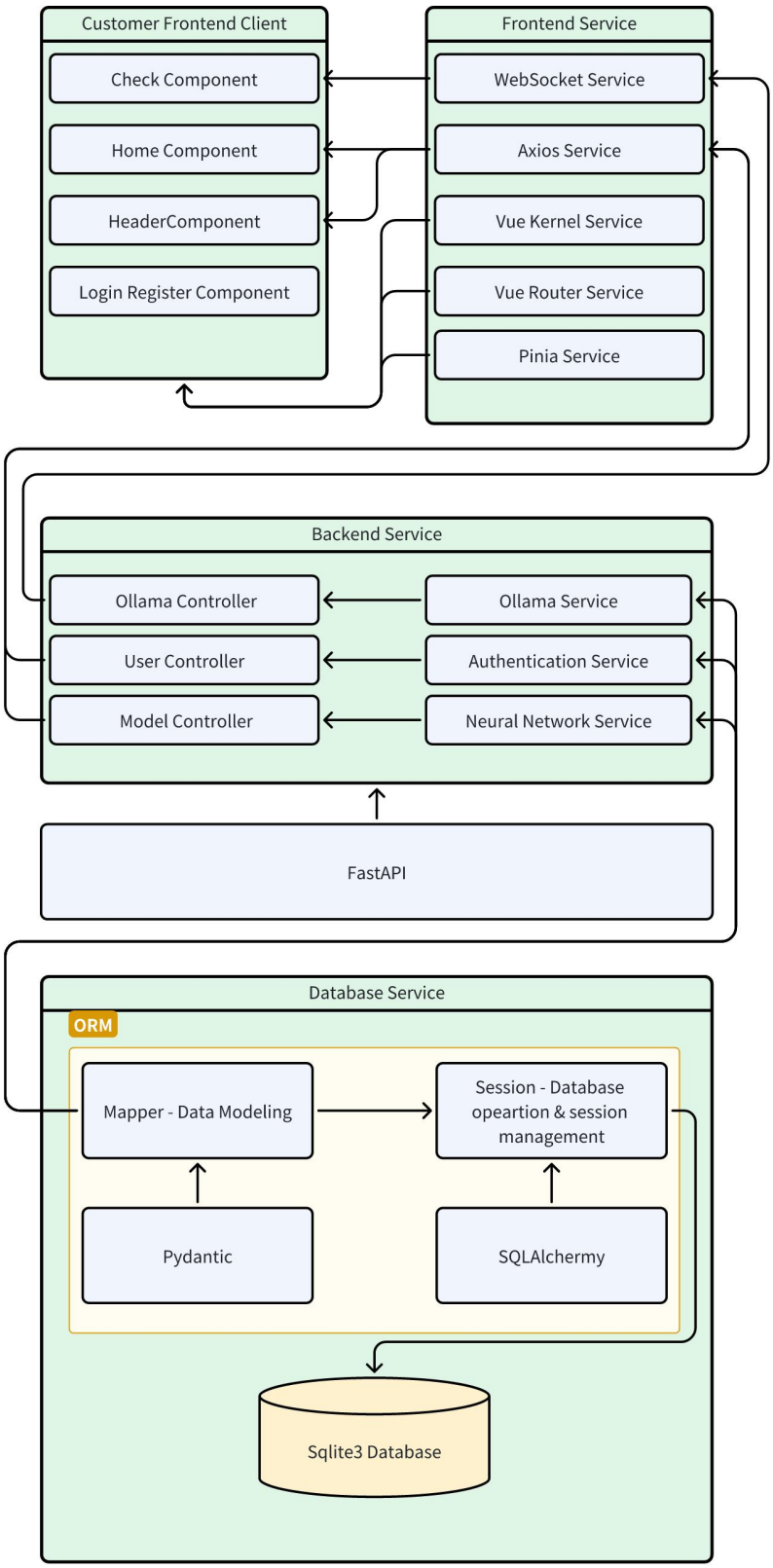
Picture 3-7: Sequence diagram for Register

3.2.2.2 Class Diagram



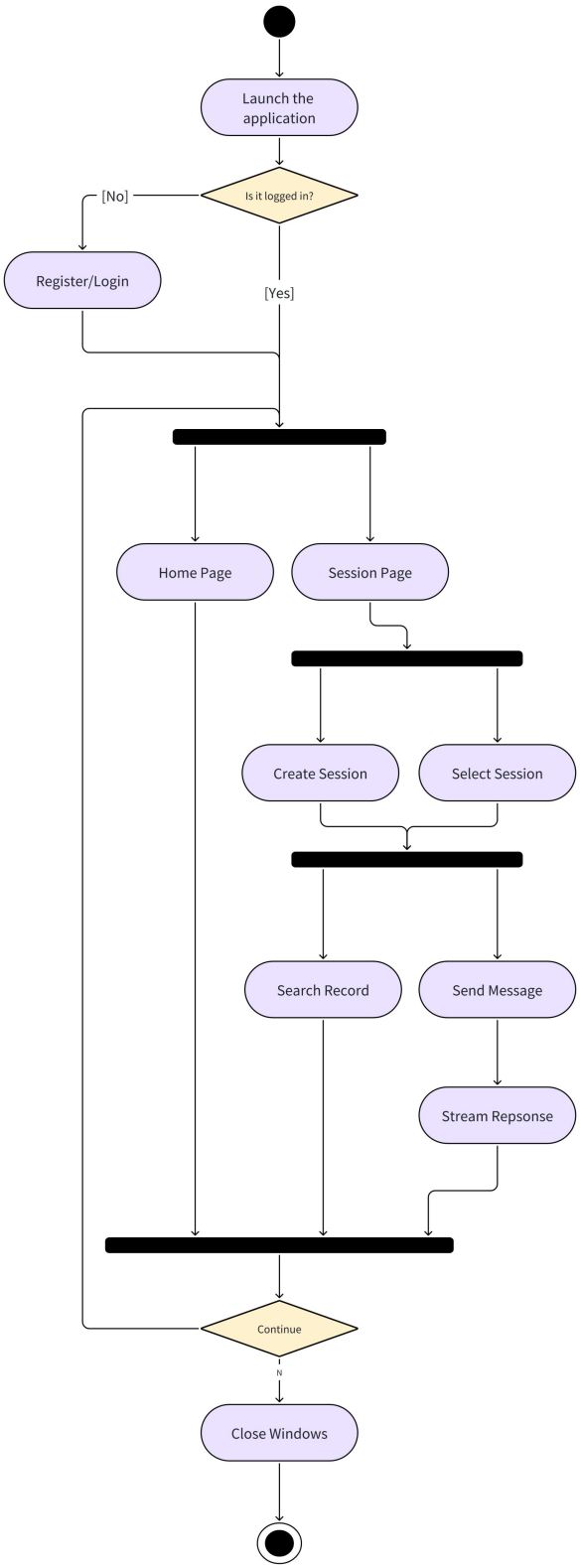
Picture 3-8: The Class Diagram

3.2.3 The Development View



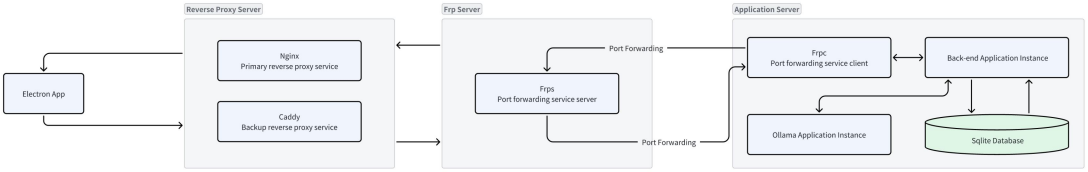
Picture 3-9: The Package Diagram

3.2.4 The Process View



Picture 3-9: The Activity Diagram

3.2.5 The Physical View



Picture 3-10: The Deployment Diagram

3.3 Database Design / ER Diagram

To effectively support the above system functions, especially user management, session communication, and tongue image analysis data storage, the database adopts the following relational structure:

☐ **`User` Table:**

①`id (integer, primary key)`: Unique Device Identifier of the user.

②`email (varchar(255))`: The user's email address, typically used for login and identity verification.

③ `password (varchar(255))`: The user's password, usually stored as an encrypted hash value to ensure security.

④This table serves as the basis for all user-related data.

☐ **`chatSession` Table:**

①`id (integer, primary key)`: Unique Device Identifier of the chat session.

②`user_id (int, foreign key)`: Associated with the id in the User table, indicating which user this conversation belongs to. A user can have multiple chat conversations.

③`tittle (text)`: The title or subject of a conversation.

☐ **`chatRecord` Table:**

①`id (integer, primary key)`: Unique Device Identifier of the chat record.

②`session_id (int, foreign key)`: Associated with the id in the chatSession table, indicating which chat session this record belongs to. A session can contain multiple chat records.

③`content (text)`: The specific content of the chat message.

④`create_at (int)`: Timestamp of message creation.

⑤`role (int)`: Identifies the role of the message sender (e.g., user or AI).

☐ **`TongueAnalysis` Table:**

① This table stores various structured diagnostic indicators and feature data obtained after the AI model analyzes the user's tongue image, `id (integer, primary`

`key)`: Unique Device Identifier for tongue image analysis records.

② `user_id (int, foreign key)`: Associated with the id in the User table, indicating which user this analysis record belongs to. A user can have multiple tongue image analysis records.

③`img_src (varchar(255))`: Storage path or URL of the original tongue image.

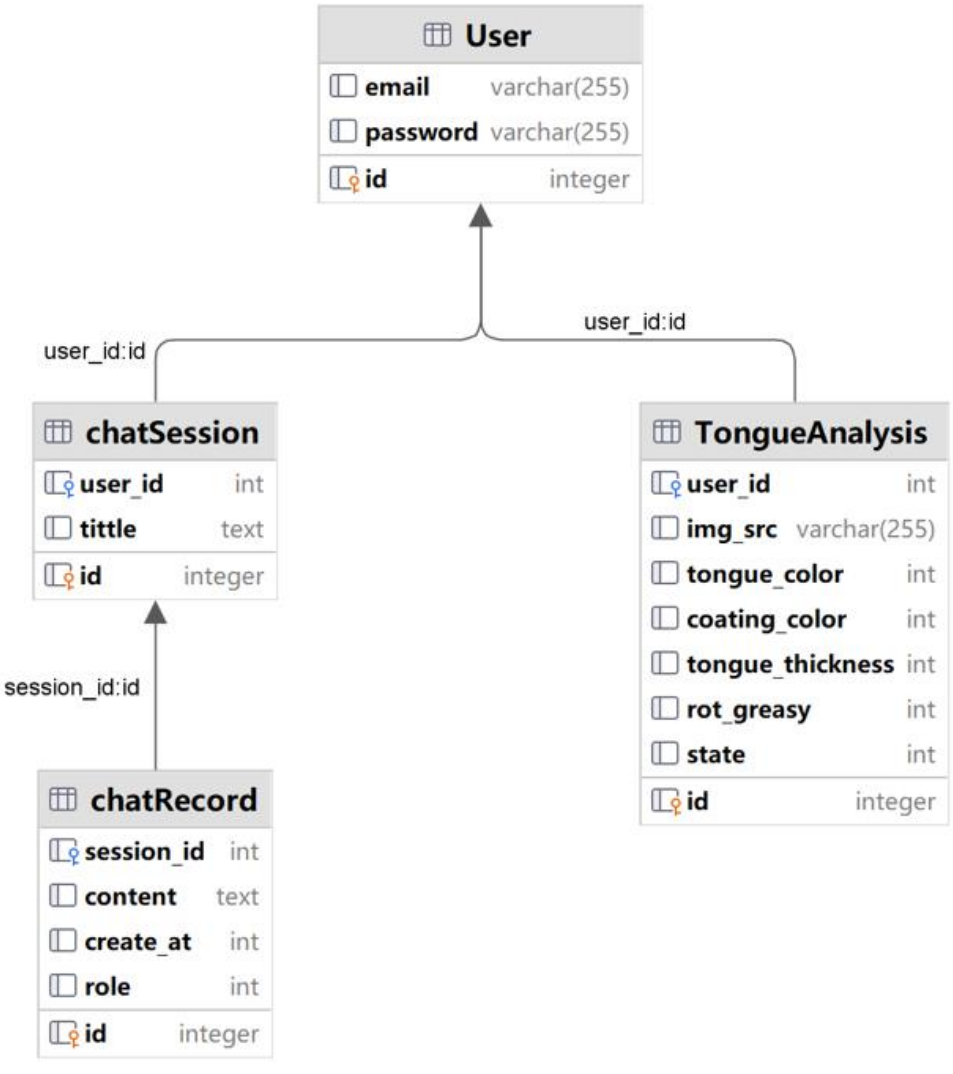
④`tongue_color (int)`: Analysis result of tongue color (possibly corresponding to a coded value).

⑤`coating_color (int)`: The analysis result of the tongue coating color.

⑥`tongue_thickness (int)`: Analysis result of tongue thickness.

⑦`rot_greasy (int)`: The analysis result of the degree of greasy tongue coating.

⑧`state (int)`: Overall health status or other classification results.



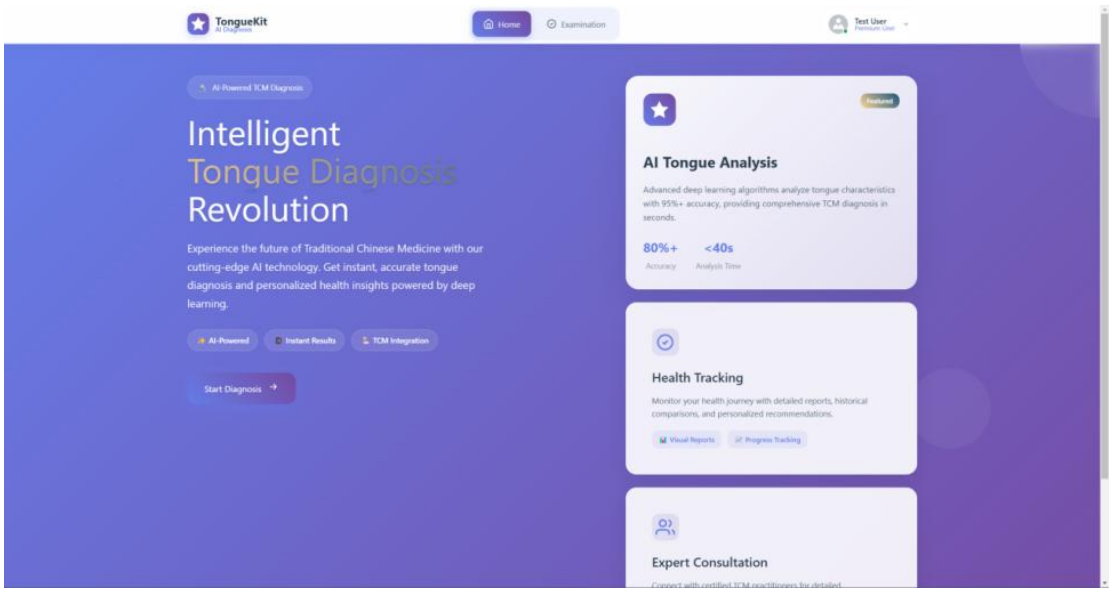
Picture 3-11: ER Diagram of the Database

3.4 UI/UX Design

In UI/UX (User Interface and User Experience) design, we have adopted a strategy led by developers, component-based, and fully leveraging mature front-end frameworks to achieve rapid development, maintain visual consistency, and ensure a good user experience. The front-end application interface is built on Element Plus and Tailwind UI:

We have chosen Element Plus as the core Vue.js User Interface (UI) component library, which provides a large number of pre-built, highly encapsulated UI components (such as buttons, forms, tables, navigation, dialogs, etc.). Element Plus adheres to a rigorous and mature design system and visual specification, which inherently brings visual consistency, interaction consistency, and component accessibility to our application.

We have integrated Tailwind UI to build and beautify specific layouts, page sections, or custom components that Element Plus cannot directly cover in the application. Tailwind UI is a collection of component examples built on Utility-first Tailwind CSS, which gives front-end developers great flexibility to quickly build beautiful, highly responsive, and modern interface elements by combining atomic CSS classes. This allows us to maintain the consistency of Element Plus core components while having more precise control over layout, spacing, typography, and color scheme.



Picture 3-12: UI/UX Design of the Home Page

Part04: API Documents

4.1 Basic Format

☐ Universal Response Format

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": {}
}
```

☐ **code**: Response Status Code
☐ **message**: Response description information
☐ **data**: Specific response data4.2 User Management

4.2 User Management

4.2.1 User Registration

☐ Path: `POST "/api/user/register"`
☐ Description: New user registration
☐ Request body:

```
JSON
{
    "email": "string",
    "password": "string"
}
```

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": null
```

```
}
```

☐ Status Code:

- ■ 0: Registration successful
- ■ 101: Registered
- ■ Other: Operation failed

4.2.2 User Login

☐ Path: `PUT "/api/user/login"`
☐ Description: User Login
☐ Form parameters:

Parameter name	Parameter meaning	Data type
email	User email	string
password	User password	string

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": {
        "token": "string"
    }
}
```

☐ Status Code:

- ■ 0: Login successful
- ■ 102: Incorrect password
- ■ Other: Operation failed

4.2.3 Get User Info

☐ Path: `GET "/api/user/info"`
☐ Description: Get current user information
☐ Authentication: `Bearer Token`

☐ Request Header:
- ■ `Authorization: Bearer <token>`

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": {
        "ID": "int",
        "email": "string"
    }
}
```

4.2.4 Get User Record

☐ Path: `GET "/api/user/record"`
☐ Description: Retrieve user's tongue diagnosis history records
☐ Authentication: `Bearer Token`
☐ Request Header:
- ■ `Authorization: Bearer <token>`

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": [
        {
            "ID": "int",
            "user_ID": "int",
            "img_src": "string",
            "state": "int",
            "result": {
                "tongue_color": "int",
                "coating_color": "int",
                "tongue_thickness": "int",
                "rot_greasy": "int"
            }
```

```
        }
    ]
}
```

4.3 Tongue Diagnosis Analysis

4.3.1 Upload image and analysis

☐ Path: `POST "/api/model/session"`
☐ Description: Upload tongue diagnosis images for AI analysis and start a new session
☐ Authentication: `Bearer Token`
☐ Request Header:
- ■ `Authorization: Bearer <token>`

☐ Form parameters:

Parameter name	Parameter meaning	Data type
file_data	picture file	multipart/form-data
user_input	User input text	string
name	session name	string

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": {
        "sessionId": "string"
    }
}
```

4.3.2 Send message to a session

☐ Path: `POST "/api/model/session/{sessionId}"`

☐ Description: Send a message in the specified session and obtain the AI's response
☐ Authentication: `Bearer Token`
☐ Request Header:
- ■ `Authorization: Bearer <token>`

☐ Path parameters:

Parameter name	Parameter meaning	Data type
sessionId	Session ID	string

☐ Request body:

```
JSON
{
    "input": "string"
}
```

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": null
}
```

4.3.3 Obtain conversation chat records

☐ Path: `GET "/api/model/record/{sessionid}"`
☐ Description: Retrieve the complete chat history of a specified conversation
☐ Authentication: `Bearer Token`
☐ Request Header:
- ■ `Authorization: Bearer <token>`

☐ Path parameters:

Parameter name	Parameter meaning	Data type
sessionId	Session ID	string

☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": {
        "records": [
            {
                "content": "string",
                "create_at": 1678886400000,
                "role": 1
            }
        ]
    }
}
```

role: 1 represents user, 2 represents AI assistant

4.3.4 Obtain all conversation lists

☐ Path: `"GET /api/model/session"`
☐ Description: Get all session lists of the current user
☐ Authentication: `Bearer Token`
☐ Request Header:
    ■ `Authorization: Bearer <token>`
☐ Response body:

```
JSON
{
    "code": 0,
    "message": "operation success",
    "data": [
        {
            "session_id": 1,
            "name": "string"
        }
    ]
}
```

☐ Response Code Explanation:

- ■ 0: Operation successful
- ■ 101: User not found or operation failed
- ■ 102: Failed to obtain chat history
- ■ 201: Operation Failed

Part05: Software Implementation

5.1 Development Approach

5.1.1 Front-end Application

The core of the front-end application is built on the Electron framework, which enables the use of web technologies (HTML, CSS, and JavaScript) to develop a fully functional desktop application that can be compiled and packaged for major operating systems such as Windows, macOS, and Linux. This design choice not only unifies the user experience across different platforms, but also grants the application capabilities beyond those of traditional browsers, such as access to the local file system for storing high-resolution tongue images and sending system-level notifications.

Within the Electron container, Vue 3.js is selected as the primary front-end framework. Vue 3, with its powerful Composition API and highly efficient reactivity system, significantly improves code readability and maintainability. Its component-based development paradigm allows complex user interfaces to be decomposed into independent and reusable units.

To achieve both aesthetic quality and development efficiency in UI design, a hybrid approach combining Element Plus and Tailwind UI is adopted. Element Plus provides a comprehensive set of well-designed foundational UI components, accelerating the development of standard interface elements. Tailwind UI, as an atomic CSS framework, offers great flexibility and efficiency for highly customized layouts and fine-grained visual effects, while avoiding the complexity and maintenance issues associated with extensive CSS overrides.

For application state management, Pinia, the officially recommended solution within the Vue ecosystem, is introduced. It provides a type-safe and lightweight centralized store for managing global states such as user authentication status and application configuration, ensuring a clear, predictable, and debuggable data flow. For front-end and back-end communication, Axios is used as the HTTP client. Its powerful request and response interceptor mechanisms are leveraged to uniformly handle authentication token injection and API error responses, resulting in a more robust and elegant networking layer.

5.1.2 Back-end Application

The back-end service is built on FastAPI, a high-performance Python web framework. The primary reason for choosing FastAPI lies in its native asynchronous support,

which allows it to efficiently handle high-concurrency requests. This capability is particularly critical in scenarios where a large number of users may upload tongue images for analysis simultaneously. In addition, FastAPI is deeply integrated with Pydantic, enabling automatic data validation and the generation of interactive API documentation, which significantly improves development efficiency and interface reliability.

As a traditional Chinese medicine tongue diagnosis application, the core functionality lies in image analysis algorithms. The back-end service is seamlessly integrated with neural network models developed using PyTorch. Acting as a lightweight and efficient interface layer, FastAPI is able to invoke PyTorch models with minimal latency and quickly respond to analysis requests from the front end, thereby providing users with a smooth and responsive experience.

For data persistence, SQLite is adopted as the database engine. For desktop applications, SQLite is an ideal choice due to its lightweight, serverless, and embedded nature. It exists directly as a file and does not require separate deployment or maintenance of a database service, greatly simplifying application distribution and deployment. To enable efficient and secure interaction with the database, the SQLAlchemy Object-Relational Mapping (ORM) library is used. It abstracts database tables into Python objects, allowing developers to manipulate data in an object-oriented manner, avoiding complex native SQL queries while ensuring robustness in data operations.

5.1.3 Security System

Security is treated as a core design principle of the system. A defense-in-depth security architecture is constructed to cover the entire pipeline of data transmission, user authentication, and data storage, ensuring the highest level of protection for user data and privacy.

First, at the data transmission layer, all communications between the client and the server are strictly enforced to use the HTTPS (HyperText Transfer Protocol Secure) protocol. By deploying SSL/TLS certificates, the communication channel is encrypted, effectively preventing data from being intercepted or tampered with during transmission. This serves as a fundamental safeguard against Man-in-the-Middle attacks. Whether it is user authentication, tongue image uploads, or the return of analysis results, all data flows securely through encrypted channels.

Second, for user authentication and authorization, a stateless mechanism based on Access Tokens and Refresh Tokens is implemented. Upon successful login, the server issues a short-lived Access Token and a long-lived Refresh Token. The Access Token acts as the credential for accessing protected APIs, and its short validity period significantly reduces security risks in the event of token leakage. The Refresh Token

is used to automatically and silently obtain a new Access Token after the original one expires. This mechanism effectively balances strong security with a smooth user experience, eliminating the need for frequent re-authentication and enabling seamless token refresh.

Finally, at the data storage layer, highly sensitive information such as user passwords is strictly protected through irreversible processing. Industry-standard SHA256 hashing combined with salting is employed, ensuring that user passwords are never stored in plaintext form. Even in extreme cases where the database is compromised, attackers cannot directly recover the original passwords from the stored hash values, providing a final and robust line of defense for user account security.

Through the combined use of HTTPS-encrypted communication, token-based authentication and authorization, and salted hash storage, a comprehensive security loop is established from the client to the server and from the network to the database, ensuring data confidentiality, integrity, and reliable authentication.

5.2 Key Algorithms and AI technologies

5.2.1 Data Pre-processing

5.2.1.1 Semantic Segmentation of Tongue Images

In a tongue image, pixels unrelated to the tongue body are inevitably present and act as noise. To accurately extract meaningful tongue features, a data preprocessing stage is introduced to remove these interfering elements.

First, the YOLOv5 object detection model is used to locate the position of the tongue within the image. As an open-source object detection framework, YOLOv5 allows us to isolate the primary tongue region, reduce background interference, and filter out abnormal or low-quality images. A custom annotated object detection dataset is used to train the YOLOv5 model.

Next, semantic segmentation is performed on the detected tongue region using the Segment Anything Model (SAM). SAM is a large-scale foundational semantic segmentation model that operates based on prompts and can generalize effectively to a wide range of downstream segmentation tasks. The output bounding box generated by YOLOv5 serves as the prompt input for SAM, enabling accurate tongue segmentation and extraction.

At this stage, the data preprocessing pipeline is essentially complete, with most interfering factors removed. By processing the original dataset in this manner, a new

dataset containing only isolated tongue regions is obtained, which serves as the input for subsequent classification tasks.

5.2.1.2 Dataset Normalization

Mean‐variance normalization is applied to the dataset, using the global mean and variance computed over the entire dataset. This method preserves relative distances between data points while transforming the data distribution to have a mean of 0 and a variance of 1. Such normalization accelerates model convergence, improves training efficiency, and enhances overall model performance.

5.2.1.3 Dataset Labeling

In this project, classification features are divided into four main dimensions: tongue color, tongue coating color, thickness, and greasiness. Tongue color is categorized into pale white, light red, red, crimson, and purplish. Tongue coating color includes white, yellow, and gray-black categories, while thickness and greasiness are treated as binary classification tasks. All dataset labels are recorded in JSON format for structured storage and processing.

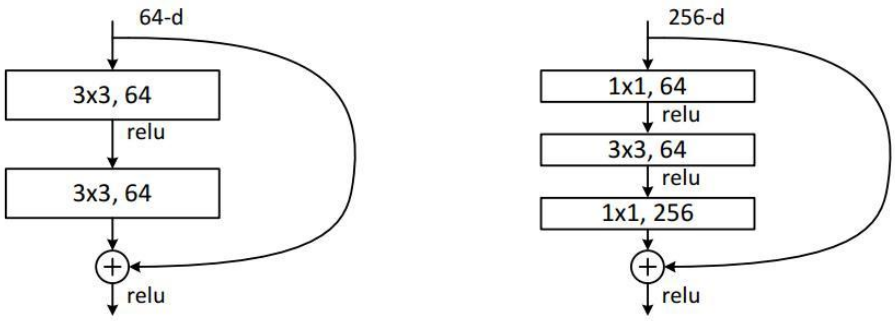
5.2.1.4 Data Augmentation

To address the issue of class imbalance, data augmentation is employed to increase the number of minority samples. Common data augmentation techniques include random rotation, cropping, and translation. Considering that cropping and translation may distort the fundamental structure of the tongue, only rotation-based augmentation is applied in this work.

5.2.2 Classification Network Construction

5.2.2.1 Residual Network

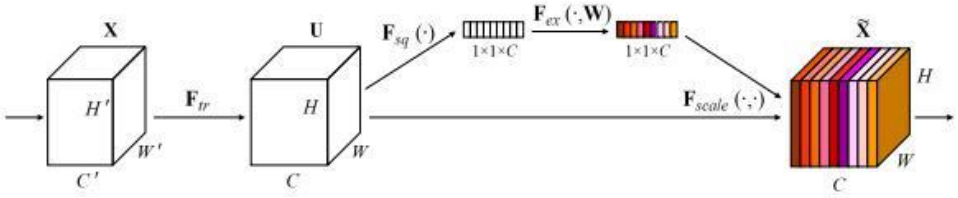
The residual learning framework addresses the degradation problem that occurs in very deep neural networks by introducing skip connections. By stacking residual blocks with basic convolutional structures, a deep convolutional neural network can be constructed. Within each residual block, a shortcut connection is introduced to preserve identity mapping when the residual becomes zero, thereby ensuring stable network performance. When the residual is non-zero, the network is able to learn deeper and more expressive features. In this work, ResNet50 is adopted as the backbone network for tongue image classification.



Picture 5-1: Core Structure of Residual Network

5.2.2.2 Squeeze-and-Excitation Network

The squeeze-and-excitation network is an attention-based model built upon the Squeeze-and-Excitation (SE) module, which enhances feature representation by explicitly modeling interdependencies among convolutional feature channels. Through the Squeeze operation (global information embedding), global average pooling is applied to compress each feature map into a 1×1×c1 \times 1 \times c1× 1 × c feature vector, effectively capturing global context and alleviating channel dependency issues. Subsequently, the Excitation operation (adaptive recalibration) employs two fully connected layers: the channel dimension is first reduced and activated using ReLU, then restored and activated using Sigmoid to generate channel-wise attention weights. Finally, the Scale operation reweights the original feature maps by applying these attention weights to each channel, producing the final output features.



Picture 5-2: The Structure of Squeeze-and-excitation Network

5.2.2.3 The design of the Loss Function

We adopt the Symmetric Modified Cross-Entropy (SMCE) loss function, which can been seen in (Formula 5.1). For tongue color classification tasks involving noisy samples, this loss function effectively mitigates the impact of noise. SMCE is derived by modifying the noise-robust Generalized Cross-Entropy (GCE) loss. The GCE loss combines the symmetry property of MAE with the fast convergence characteristics of CE. However, due to the influence of the dynamically adjusted parameter ***d***, GCE exhibits noise robustness only under certain conditions. By introducing ADD, SMCE

guarantees that, regardless of the value of ***d***, there is always a symmetric loss component that contributes to noise suppression.

$$L_{SMCE} = \alpha \cdot L_{GCE} + \beta \cdot L_{ADD}$$
$$= \alpha \cdot \frac{1 - p(y|x)^d}{d} + \beta \cdot \frac{\sum_{k=1}^{K} |p(k|x) - q(k|x)|}{\sum_{j=1}^{K} \sum_{k=1}^{K} |p(k|x) - q(y = j|x)|} \quad (5.1)$$

5.2.2.4 Gradient Descent based on Momentum

In the gradient descent optimization process, we adopt momentum-based stochastic gradient descent (SGD). During each update step, the optimizer retains the trend of previous updates, which helps prevent the loss from being trapped in local optima. Even in cases where the gradient becomes zero, the momentum term maintains a movement tendency, enabling continued optimization progress.

5.2.2.5 Learning Rate Scheduling Algorithm

For the learning rate scheduling strategy, we adopt the Poly policy, in which the learning rate decays as the number of training iterations increases. By adjusting the exponential parameter ***power***, the shape of the learning rate decay curve can be flexibly controlled. When ***power*** is set to a value less than 1, the learning rate decreases slowly at first and then more rapidly; when ***power*** is greater than 1, the learning rate drops quickly in the early stages and then slows down. As the model loss approaches convergence, using a smaller learning rate enables the model to more effectively locate the optimal solution.

5.3 Development collaboration

We use Git and GitHub for collaborative development, with main serving as the stable primary branch. Daily development work is carried out on a dev branch derived from main, while bug fixes are handled through dedicated fix branches. All changes are merged into main via Pull Requests, which facilitates changelog generation and change tracking. For each release, a version tag is created on the main branch in GitHub, forming clear version milestones and ensuring traceable version history.

A privately deployed Jenkins instance is responsible for CI/CD automation. Pushes to main or Pull Request merge events trigger the Jenkins pipeline, which executes a predefined workflow: checking out the relevant commit or merged result, installing project dependencies, performing basic static checks and necessary unit tests to ensure quality, and then building and packaging the application to generate distributable artifacts. Upon completion, Jenkins archives the build outputs (such as installers,

compressed packages, or images) in an artifact repository or within Jenkins build archives. Credentials and environment variables are securely injected through Jenkins' credential management and parameterized configuration, ensuring that sensitive information is not stored in the code repository.

When a build fails, Jenkins marks the build as failed and sends automatic notifications. Developers can fix the issues and trigger a new build by pushing updates, or manually retry the build within Jenkins when necessary. To facilitate rollback and comparison, Jenkins retains artifacts and logs from several recent builds. When a release is required, a tag is created on GitHub, which triggers a formal Jenkins build based on that tag, producing versioned artifacts that are archived and released.

Part06: Software Testing

6.1 Test Strategy

The software testing strategy for this project takes "verifiable backend logic, consistent interface contracts, end-to-end availability of the front end, and robust AI pipelines" as its core objectives, and is carried out around four levels: unit, interface, end-to-end, and model pipeline.

The backend uses pytest to write unit tests, covering key logic such as FastAPI routes and services, JWT issuance and verification, SQLAlchemy data access, exceptions, and boundary conditions. Mocking is widely used to isolate dependencies on databases, model inference, and Ollama, ensuring the verification of input-output contracts and error code consistency in a controlled environment.

The interface layer maintains API collections through Apifox, validates REST request and response formats, authentication and token refresh, and multi-part form image upload.

Frontend e2e testing uses Cypress to simulate real user paths in both Web and Electron environments, covering core processes such as login, session creation and status prompts, image upload and progress, report streaming display, history retrieval, and continued conversation.

On the algorithm side, lightweight pipeline integration testing is conducted using pytest to evaluate the cascade correctness, mask quality, and feature vector consistency of YOLO localization, SAM segmentation, and SE-ResNet classification, record the Confidence Level and version tags, and verify the failure fallback and retry strategies.

The test environment covers backend Debian 12 (Python 3.9, Uvicorn/FastAPI), client Windows x86-64 (Electron/Web), SQLite 3.35+, Ollama (deepseek-r1-14b), and provides public network access through Nginx/Caddy and FRP;
The entire testing system is integrated into the Jenkins pipeline, which executes dependency installation, Pytest, Apifox collection, Cypress execution, and report archiving.

6.2 Test Cases & Results

Due to space limitations, we only provide one case for each type of test.

6.2.1 Front-end e2e testing

☐ **Test code**

```
JavaScript
it("Upload File Test", () => {
    // Login
    cy.visit('/');
    cy.get('input[id="l_email"]').type("testuser@testuser.com");
    cy.get('input[id="l_password"]').type("123456");
    cy.get('button').contains("登录").click();

    // Examination
    cy.get('a').contains("进行检测").click();
    cy.get('.el-upload__input').attachFile("657.jpg", {
        subjectType: 'drag-n-drop'
    })
})
```

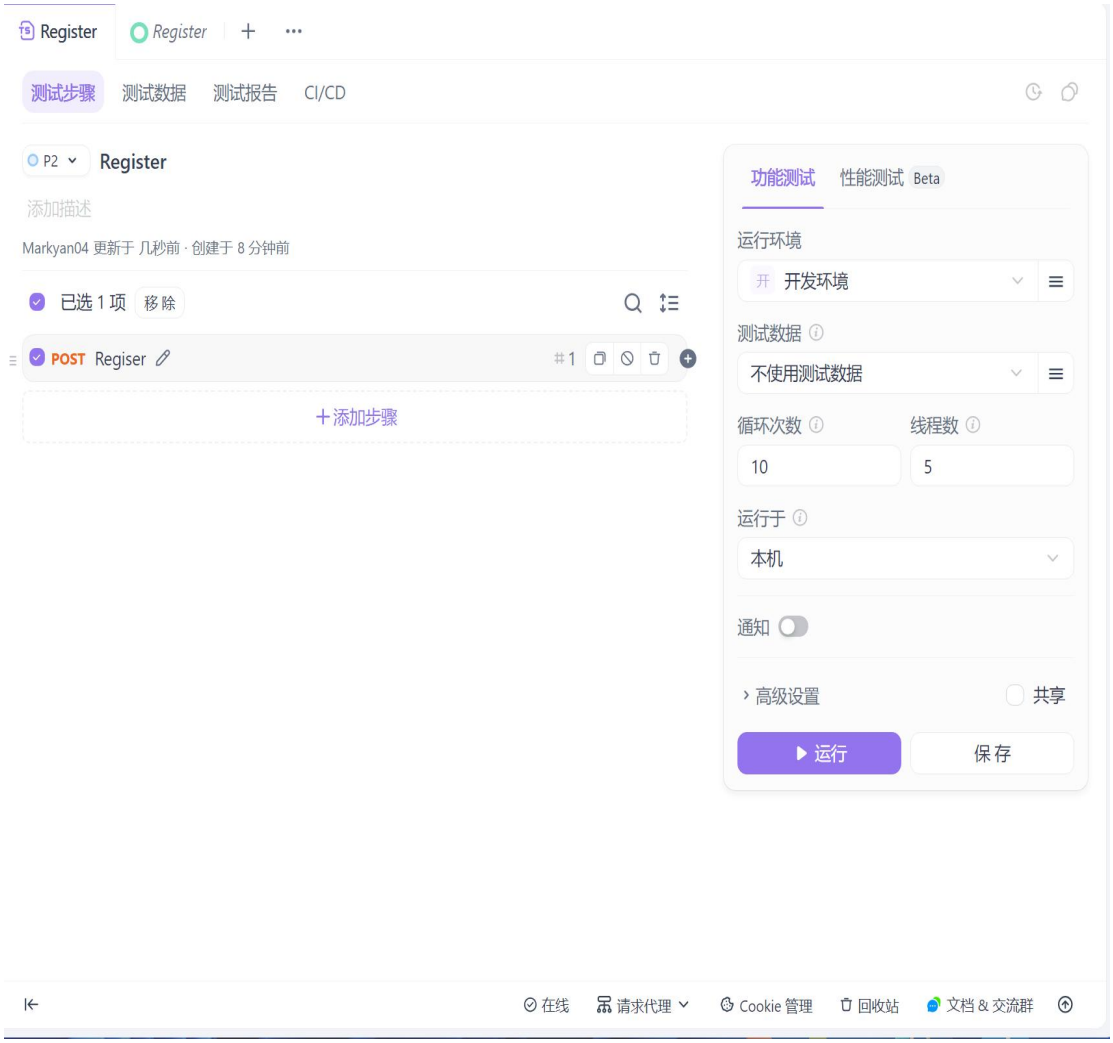
☐ **Test Results**:

■ The reason for 401 is that when the page initializes, it will retrieve user information, but at this time the user has not logged in yet, so the response to this request is 401.

Picture 6-1: Test Result of e2e Testing

6.2.2 Interface Layer - Apifox Testing

☐ **Test Case:** A POST request to register a new user. It is configured to run in the Development environment. The execution plan runs locally with 10 iterations and 5 threads, which means about 50 total requests will be sent concurrently.



Picture 6-2: Test Case of Interface Testing

☐ **Test Results**:

Picture 6-3: Test Result of Interface Testing

6.2.3 Backend unit testing

☐ **Test case**: Write unit tests for Authentication using pytest to verify whether the authentication function is working properly.

☐ **Test Results**:

```
Bash
(TongueDiagnosisEN) PS F:\ParadoxAIStudio\TongueDiagnosisEN> python
-m pytest tests/test_authentication.py -v
============================================================ test
session starts
============================================================
platform win32 -- Python 3.9.21, pytest-8.4.2, pluggy-1.5.0 --
E:\Anaconda\envs\TongueDiagnosisEN\python.exe
cachedir: .pytest_cache
rootdir: F:\ParadoxAIStudio\TongueDiagnosisEN
plugins: anyio-4.3.0
```

```
collected 9 items

tests/test_authentication.py::TestCreateAccessToken::test_create_a
ccess_token_with_expiration PASSED
[ 11%]
tests/test_authentication.py::TestCreateAccessToken::test_create_a
ccess_token_default_expiration PASSED
[ 22%]
tests/test_authentication.py::TestCreateAccessToken::test_create_a
ccess_token_additional_data PASSED
[ 33%]
tests/test_authentication.py::TestGetCurrentUser::test_get_current
_user_valid_token PASSED
[ 44%]
tests/test_authentication.py::TestGetCurrentUser::test_get_current
_user_invalid_token PASSED
[ 55%]
tests/test_authentication.py::TestGetCurrentUser::test_get_current
_user_expired_token PASSED
[ 66%]
tests/test_authentication.py::TestGetCurrentUser::test_get_current
_user_token_without_email PASSED
[ 77%]
tests/test_authentication.py::TestGetCurrentUser::test_get_current
_user_nonexistent_user PASSED
[ 88%]
tests/test_authentication.py::TestOAuth2Scheme::test_oauth2_scheme
_type PASSED
[100%]

============================================================ 9 passed in
6.66s ============================================================
```

6.3 AI Performance Test

We conducted a comparative analysis of four classical convolutional classification networks, including AlexNet[19], Inception3[20], GoogleNet[21], VGG11[22]. The

models were evaluated on multi-class classification tasks across different tongue-related features, such as tongue color, tongue coating color, thickness, and greasiness, using the same dataset and identical hyperparameter settings to ensure a fair comparison of model performance.

In terms of evaluation metrics, we introduce both F1 Score and Accuracy. Among them, the F1 Score is defined as the harmonic mean of precision and recall, and it provides a balanced measure of a model's overall performance. Its calculation formula is given as formula 6.1:

$$F1 = 2 \times \frac{TP}{2TP + FN + FP} \tag{6.1}$$

The method adopted in this project is SE_ResNet50.

6.3.1 Tongue color characteristics

Model	F1 Score	Acc
AlexNet	0.1558	**0.6379**
Inception3	0.2104	0.6195
GoogleNet	**0.2858**	0.6235
VGG11	0.1558	**0.6379**
SE_ResNet50	0.1578	**0.6379**

6.3.2 Tongue coating color characteristics

Model	F1 Score	Acc
AlexNet	0.3000	0.6779
Inception3	0.4143	0.6803
GoogleNet	**0.5132**	**0.7218**

VGG11	0.3127	0.6715
SE_ResNet50	0.4256	0.6779

6.3.3 Thickness of the tongue

Model	F1 Score	Acc
AlexNet	0.3965	0.5803
Inception3	0.5963	0.6467
GoogleNet	**0.7286**	**0.7130**
VGG11	0.3965	0.5803
SE_ResNet50	0.5969	0.6515

6.3.4 Greasy or slimy characteristics of the tongue

Model	F1 Score	Acc
AlexNet	0.5545	0.6531
Inception3	0.6528	0.6811
GoogleNet	**0.6561**	**0.6914**
VGG11	0.5929	0.6571
SE_ResNet50	0.6187	0.6803

6.3.5 Summary

Through comparative analysis, we found that the overall performance of our model is

relatively unsatisfactory. Among all evaluated features, GoogLeNet consistently achieves strong performance, indicating that the current approach still has room for optimization. In future improvements, selecting GoogLeNet as the classification backbone represents a reasonable and promising direction.

At the same time, we observed that most models perform poorly on tongue color classification, and exhibit relatively low F1 scores and accuracy on other features as well. Therefore, in addition to considering the choice of model architecture, it is necessary to examine potential issues within the dataset itself and to explore more effective data preprocessing strategies in order to improve overall model performance.

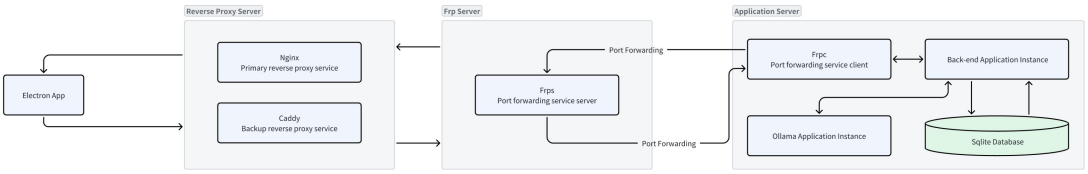
Part07: Software Deployment and Operations Management

7.1 Ops Technical Route

The core of the operations and maintenance (O&M) technical route of this project is to construct a secure, stable, and efficient access path from the client to application services within a private network by combining public reverse proxy and internal network tunneling technologies.

The entire data flow begins at the user's Electron client application. When a user initiates a request, it is first routed to a reverse proxy layer deployed on a public server. This layer serves as the unified entry point of the system and is configured in an active–standby mode to ensure high availability. Nginx acts as the primary service, handling daily traffic forwarding, load balancing, SSL termination, and certificate management thanks to its excellent performance, thereby forming the first robust line of defense for the application. Caddy, on the other hand, serves as the standby service. Its automated HTTPS management capabilities make it an ideal failover option, ensuring uninterrupted external service even when the primary service encounters failures.

Unlike traditional architectures, this reverse proxy layer does not directly forward requests to application servers exposed on the public network. Instead, traffic is passed to a co-deployed FRP (Fast Reverse Proxy) server component (frps). In this setup, FRP plays the critical role of a "tunnel hub". Meanwhile, on the application servers located within a private network (such as a corporate intranet or home network), an FRP client (frpc) actively connects to the public frps upon startup and establishes a stable, encrypted long-lived tunnel. As a result, once Nginx hands off external requests to frps, frps can securely "push" these requests through the pre-established tunnel to the internal frpc.
Ultimately, the requests reach the application service cluster securely isolated within the private network. Upon receiving a request, frpc forwards it to the appropriate local service instances. These include the back-end application instances responsible for core business logic, as well as a separately deployed Ollama service instance dedicated to handling computation-intensive large language model tasks. This service decoupling design facilitates independent scaling and maintenance. Data persistence required by the back-end application is provided by a tightly coupled SQLite database, whose lightweight and embedded nature further reduces deployment and operational complexity.



Picture 7-1: Application Deployment Structure

7.2 Software Deployment Process

7.2.1 Pre-requisite Requirements

☐ Conda ≥23.10.0
☐ Python 3.9.21
☐ SQLite 3.35+
☐ Node.js 20.0+

7.2.2 Configuration

7.2.2.1 Backend Configuration

☐ Modify `application/config/config.py` to set backend parameters:
- ■ **`ACCESS_TOKEN_EXPIRE_MINUTES`**: Token expiration time in minutes
- ■ **`SECRET_KEY`**: Secret key for JWT token generation
- ■ **`ALGORITHMS`**: Algorithm used for JWT token encoding
- ■ **`IMG_PATH`**: Path to save uploaded tongue images
- ■ **`IMG_DB_PATH`**: Path stored in the database for tongue images
- ■ **`OLLAMA_PATH`**: Ollama API endpoint
- ■ **`SYSTEM_PROMPT`**: System prompt for LLM
- ■ **`LLM_NAME`**: Name of the LLM model to use in Ollama
- ■ **`APP_PORT`**: Port for backend server

☐ Default configuration:

```
Bash
    ACCESS_TOKEN_EXPIRE_MINUTES: int = 60 * 24
    SECRET_KEY: str = "f2e1f1b1c1a1"
    ALGORITHMS: str = "HS256"
    IMG_PATH: str = "frontend/public/tongue"
    IMG_DB_PATH: str = "tongue"
    OLLAMA_PATH: str = "http://localhost:11434/api/chat"
    SYSTEM_PROMPT: str = "You are now an AI traditional Chinese medicine
```

```
doctor specializing in tongue diagnosis. At the very beginning, I will show you four image features of the user's tongue. Please use your knowledge of traditional Chinese medicine to give the user some suggestions. Answer in English"
    LLM_NAME: str = "deepseek-r1:14b"
    APP_PORT: int = 5000
```

7.2.2.2 Frontend Configuration

☐ Modify `frontend/src/config/config.js` to set frontend parameters:
- ■ **`ServerUrl`**: Backend server URL

☐ Default configuration:

```
JavaScript
export const settings = {
    ServerUrl: 'https://paradoxai.markyan04.cn'
};

export default settings;
```

7.2.3 Application Setup

7.2.3.1 Backend Setup

☐ Default running port: 5000
☐ Before the following step, you need to install Ollama[https://ollama.com/download] on the device which will run the backend application. Your Ollama service will defaultly run on port 11434.

```
Bash
# Clone repository
git clone
https://github.com/TonguePicture-SKaRD/TongueDiagnosis.git
cd TongueDiagnosis/application

# Create environment
conda create -n tongueai python=3.9.21
conda activate tongueai
```

```
cd ..
pip install -r requirements.txt

# Initialize database
sqlite3 AppDatabase.db < models/create_ChatRecord.sql  # Creates 4
tables
sqlite3 AppDatabase.db < models/create_Session.sql  # Creates 4
tables
sqlite3 AppDatabase.db < models/create_TongueAnalysis.sql  # Creates
4 tables
sqlite3 AppDatabase.db < models/create_User.sql  # Creates 4 tables

# Make a directory for saving model weights
cd application
mkdir -p ./net/weights

# Make a directory for tongue images saving
cd ../frontend/public
mkdir -p ./tongue

# Download model weights (If the terminal cannot run, please manually
download the weight file, a total of 7)
wget -P ./net/weights/ \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
ownload/V1.0_Beta/rot_and_greasy.pth" \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
ownload/V1.0_Beta/thickness.pth" \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
ownload/V1.0_Beta/tongue_coat_color.pth" \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
ownload/V1.0_Beta/tongue_color.pth" \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
```

```
ownload/V1.0_Beta/unet.pth" \

"https://github.com/TonguePicture-SKaRD/TongueDiagnosis/releases/d
ownload/V1.0_Beta/yolov5.pt" \

"https://dl.fbaipublicfiles.com/segment_anything/sam_vit_b_01ec64.
pth"

# Launch backend
cd..
python run.py
```

7.2.3.2 Frontend Setup

☐ Default running port (dev mode): 5173
☐ Default running port (preview mode): 4173
☐ Before starting, ensure a "tongue" folder exists in the "./public" directory (create if missing)

```
Bash
# Web Application
cd frontend
npm install
npm run build
npm run preview

# Electron Desktop
npm run electron:build
npm run electron:start

# Web browser (Chrome recommended)
npm run dev
```

☐ You can execute the command npm run electron:build to build the Windows x86-64 installation application.

7.3 Frp Service Configuration

Our backend application uses the Frp service to achieve intranet penetration, exposing the application from a personal server to public network services. First, we need to configure the Frp server Frps on the public network Frp server. The frps.ini configuration is as follows (for security purposes, I will hide some sensitive information):

```TOML
[common]
bind_port = 7****
token = ****************************
log_file = "/var/log/frp/frps.log"
log_level = "info"
log_max_days = 30
vhost_http_port  = ****8
vhost_https_port = ****0
dashboard_port   = ****1
dashboard_user   = mark*****
dashboard_pwd    = *********

[markyan04_jenkins_http_server_debian_1]
listen_port = 5****

[paradoxai_app_server_debian_1]
listen_port = 1****
```

After completing the server configuration, we need to configure the Frp Client Frpc on the personal server. The frpc.ini configuration is as follows (for security purposes, I will hide some sensitive information):

```TOML
[common]
server_addr = ***.***.***.***
server_port = 7****
token = ****************************

[markyan04_jenkins_http_server_debian_1]
```

```
type = tcp
local_ip = 127.0.0.1
local_port = *****
remote_port = 5****

[paradoxai_app_server_debian_1]
type = tcp
local_ip = 127.0.0.1
local_port = *****
remote_port = 1****
```

In addition, I have configured frp as a systemd system service, enabling comprehensive management via systemctl commands, for example:

```
Bash
systemctl start frp
systemctl stop frp
systemctl restart frp
systemctl enable frp
```

This configuration enables the frp service to start automatically with the system and provides a standardized service management interface.

7.4 Nginx/Caddy Service Configuration

After completing intranet penetration, we need to configure Nginx reverse proxy for the application and implement HTTPS secure access through an SSL certificate. Below, we take Nginx, which serves as the main reverse proxy service, as an example:

7.4.1 Preceding dependency download

Ensure that the system has installed the necessary dependencies:

```
Bash
sudo apt update
sudo apt install certbot python3-certbot-nginx
sudo apt install nginx
```

☐ `certbot` is used to automatically obtain and renew SSL certificates.
☐ `python3-certbot-nginx` provides an integration plugin for Nginx.
☐ `nginx` is the core component of reverse proxy services.

7.4.2 Obtaining SSL Certificate - Let's Encrypt

To obtain an SSL certificate using Let's Encrypt, the command is as follows:

```
Bash
sudo certbot certonly --nginx \
    -d paradoxai.markyan04.cn \
    --email your-email@example.com \
    --agree-tos \
    --non-interactive
```

7.4.3 Create the Nginx configuration file

☐ Main configuration file
    ■ Ensure that `/etc/nginx/nginx.conf` contains the correct global configuration (default settings can be retained).
☐ Domain name configuration file
    ■ Create the domain name configuration
    ■ `/etc/nginx/sites-available/paradoxai.markyan04.cn`

```
Bash
server {
    listen 80;
    server_name paradoxai.markyan04.cn;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl http2;
    server_name paradoxai.markyan04.cn;

    ssl_certificate
/etc/letsencrypt/live/paradoxai.markyan04.cn/fullchain.pem;
    ssl_certificate_key
```

```
/etc/letsencrypt/live/paradoxai.markyan04.cn/privkey.pem;

    location / {
        proxy_pass http://***.***.***.***:1****;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

☐ `proxy_pass` needs to be replaced with the actual intranet penetration service address (e.g., `http://frp-server-ip:frp-port`).
☐ `ssl_certificate` and `ssl_certificate_key` point to the certificate path generated by Let's Encrypt.
☐ The above configuration is not a good practice because this is just an application demo, and I have only implemented the most basic configuration functions.

7.4.4 Start the site and test the configuration

Create a symbolic link to activate the configuration file:

```
Bash
sudo ln -s /etc/nginx/sites-available/paradoxai.markyan04.cn
/etc/nginx/sites-enabled/
```

Check if the configuration file has any syntax errors:

```
Bash
sudo nginx -t
```

If there are no errors, restart the Nginx service to apply the changes:

```
Bash
sudo systemctl reload nginx
```

After using reload instead of restart, the service will restart smoothly, avoiding interrupting current connections.

7.4.5 Set up automatic renewal

Let's Encrypt certificates are valid for 90 days. Set up automatic renewal:

```
Bash
sudo crontab -e
```

Add the following content (at the end of the file)

```
Plain Text
0 12 * * * /usr/bin/certbot renew --quiet --nginx && systemctl reload
nginx
```

☐ The `--quiet` parameter avoids logging output during the renewal process.
☐ `--nginx` will automatically update the Nginx configuration file to adapt to the new certificate.
☐ After renewal, reload the configuration via `systemctl reload nginx` to ensure the service is available.

7.5 Client Application Preparation

Users can directly obtain the installation package of our application via *https://download.markyan04.cn*, or they can deploy the front-end application themselves and package the desktop installation package. The specific installation process is detailed in 9.1 Installation.

Part08: Project Management

8.1 Roles & Responsibilities

The division of responsibilities and interface collaboration is as follows:

☐ **Yan Hongyu (Software Architecture Design and Project Framework Construction)**

■ **Responsibilities**: Overall system architecture and module boundaries; definition of pluggable algorithm interfaces; back-end project skeleton and configuration system; CI/CD and release strategy

■ **Key Deliverables**: System architecture documentation; FastAPI project skeleton; Vue project skeleton; Pydantic/SQLAlchemy base classes; Jenkins pipelines; deployment scripts; software deployment and server operations

■ **Collaboration Interfaces**: Interface contracts between back-end controllers and algorithm services; front-end Axios base configuration; coordination with operations for the reverse proxy chain

■ **Risk Mitigation**: Architecture evolution and dependency upgrade strategy; leading refactoring efforts when coupling issues arise

☐ **Wang Siyun (Login and Registration Module)**

■ **Responsibilities**: UAM module (registration/login/user information); JWT issuance and refresh; front-end authentication state and interceptors

■ **Key Deliverables**: /api/user/register, /api/user/login, /api/user/info; Pinia user store; token management and secure storage

■ **Collaboration Interfaces**: Defining authentication middleware with Yan Hongyu; integration with front-end routing and interceptors; merging authorization with historical record access

■ **Risk Mitigation**: Protection against weak passwords and brute-force attempts; consistency of error codes and user-friendly messages

☐ **Liu Siyuan (AI Detection and Conversational Module)**

■ **Responsibilities**: Session creation; image upload triggering analysis; conversational APIs; context aggregation; front-end streaming rendering of conversations and status indicators

■ **Key Deliverables**: /api/model/session; /api/model/session/{sessionId}; front-end conversation components and progress indicators

■ **Collaboration Interfaces**: Agreement on feature vector formats with algorithm services; prompt and context structure alignment with the Ollama controller; linkage with historical record persistence

■ **Risk Mitigation**: Network interruption handling and retry mechanisms; context truncation and summarization strategies

☐ **Chen Sifan (AI Inference Algorithm Module)**

■ **Responsibilities**: YOLO localization, SAM segmentation, SE-ResNet classification; training and inference optimization; error fallback and version management

■ **Key Deliverables**: Model weights and loaders; asynchronous inference services; feature vector generation and confidence logging; pipeline integration tests

■ **Collaboration Interfaces**: Defining inference APIs with back-end controllers; interfacing feature vectors and report context with LLM services; recording ModelResults with the storage layer

■ **Risk Mitigation**: Inference latency and resource utilization; quality thresholds and failure notification strategies

☐ **Gai Leilei (Historical Conversation Storage Module)**

■ **Responsibilities**: Session and message persistence; historical list and detail APIs; retrieval and pagination; ER design and migration scripts

■ **Key Deliverables**: /api/model/session (list), /api/model/record/{sessionId} (details); SQLite table schemas and ORM models; data consistency strategies

■ **Collaboration Interfaces**: Message write integration with conversation and report modules; front-end historical view rendering; unified authorization checks with the authentication layer

■ **Risk Mitigation**: Concurrent writes and transaction management; data migration and version evolution

8.2 Tools & Methodology

The project adheres to agile principles in both methodology and tooling. A Scrum-based process with weekly iterations is adopted, complemented by 15-minute online stand-up meetings every two days to identify and remove impediments. Each sprint is driven by clearly defined goals, and once requirements are frozen, only minor changes are accepted.

Requirements and task management are carried out using Feishu cloud documents, where use-case-driven work items are decomposed with explicitly defined priorities, owners, and deadlines. All tasks are described through well-defined requirement documents to ensure clear scope boundaries and executable acceptance criteria. Version and branch management follow a fixed standard: main serves as the stable branch, dev is used for daily development, and fix/* branches are dedicated to bug fixes. All Pull Requests must pass required checks before being merged.

The CI/CD process is centered around Jenkins pipelines, which uniformly execute dependency installation, static code analysis (Lint), back-end unit and service tests (pytest), and front-end end-to-end tests (Cypress), followed by build, packaging, and artifact archiving. A work item is considered complete only when the code passes CI, necessary test cases and documentation are updated in sync, and API contracts remain

stable without breaking existing integrations. By adopting a testing pyramid strategy, unit and API tests form the foundation, while end-to-end tests cover critical user paths, achieving a balance between quality assurance and feedback speed.

In terms of security and compliance, the back end uniformly enforces JWT-based authentication and TLS-encrypted communication, with sensitive information injected via credential management rather than stored in the codebase. License compliance checks are performed for all third-party and open-source assets (such as YOLOv5, SAM, and Ollama), strictly adhering to usage and distribution boundaries. Configuration and environment management rely on requirements.txt for the back end and package-lock.json for the front end to lock dependency versions. Parameterized configuration via config.py supports multiple environments (dev/prod), while Jenkins parameters and environment variables enable pipeline-level environment differentiation.

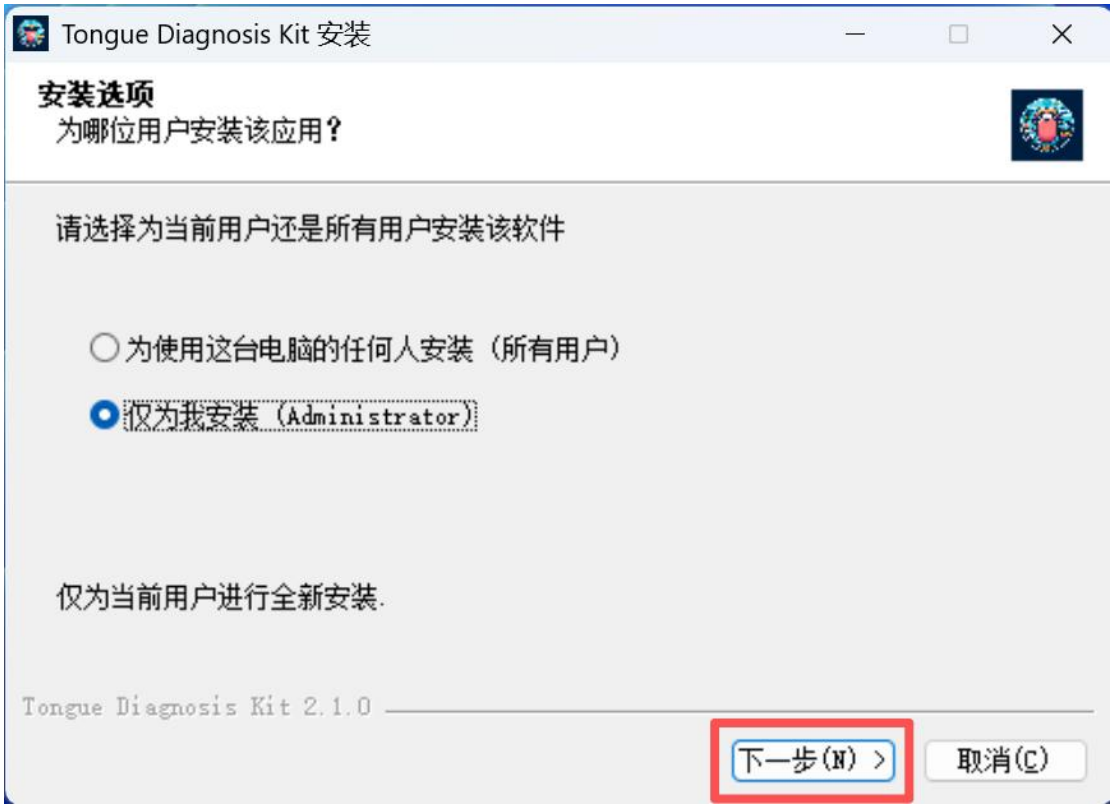
For deployment and operations, an active – standby reverse proxy setup using Nginx/Caddy, combined with FRP for internal network tunneling, is employed to establish a secure access chain, with automatic certificate renewal configured to ensure long-term HTTPS availability. On the client side, Electron packaging and release processes are provided, along with maintained download sources and version notes. Documentation and communication adhere to a single source of truth principle: technical documentation, Apifox API documentation, and README files are kept consistent. The team uses Feishu as the primary communication channel, ensuring transparency and traceability of information and discussions.

Part09: User Guide

9.1 Installation

Since the system language of the demonstration machine is Chinese, the current installation guide is in Chinese. If your computer's system language is English, the installation guide will automatically switch to the corresponding language.

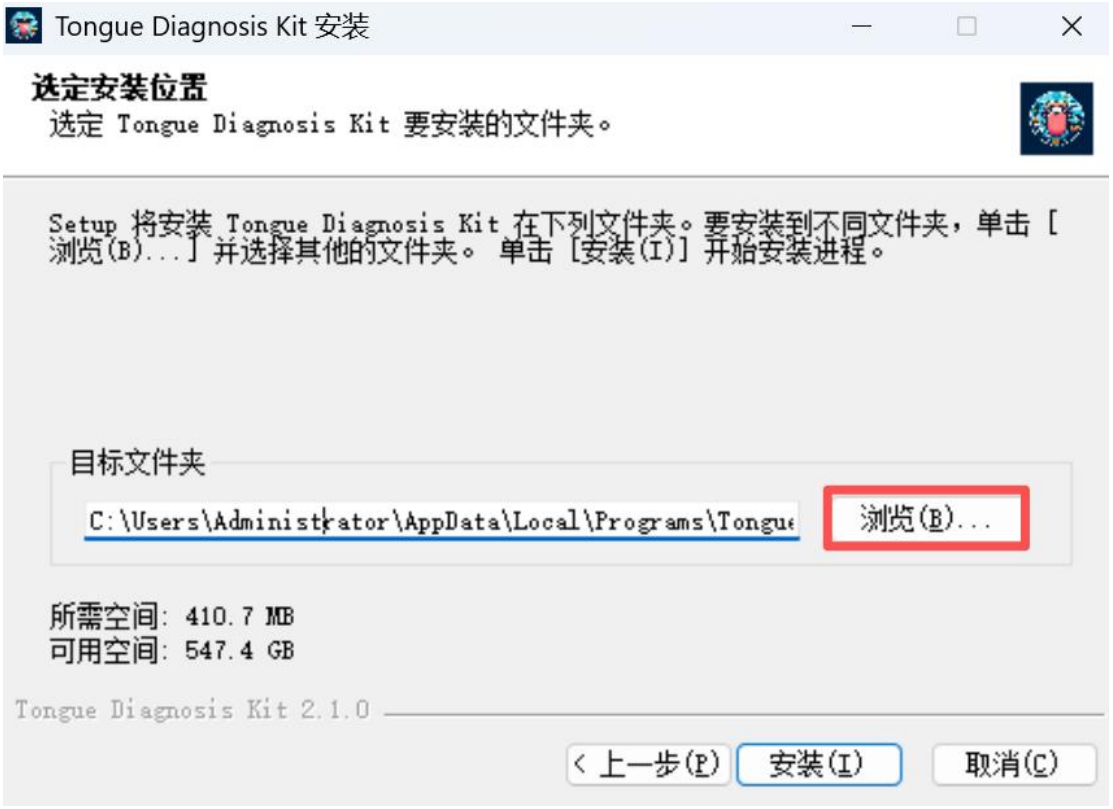
Taking the Windows x86-64 installation package as an example, after double-clicking the installation package (a.exe executable file), the following interface will pop up:



Picture 9-1: Select the installation user

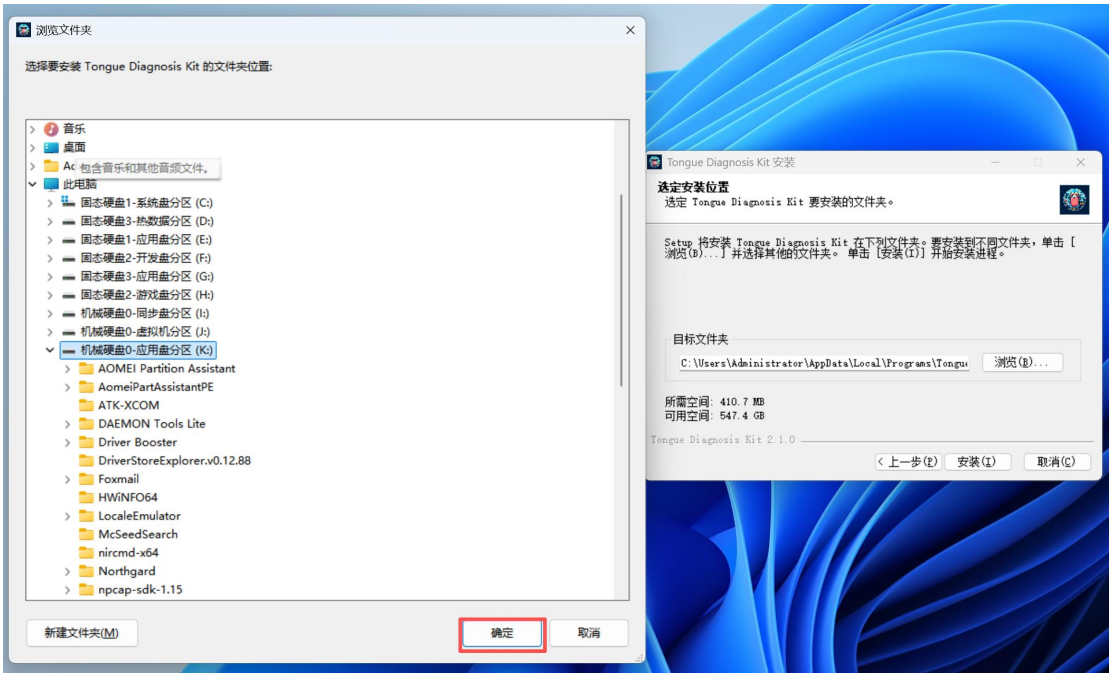
You can choose "Install just for me" or "Install for all users of this computer", then click "Next".

Then you can choose the installation path of the application. You can use the default installation path or click "Browse" to select your desired installation path.



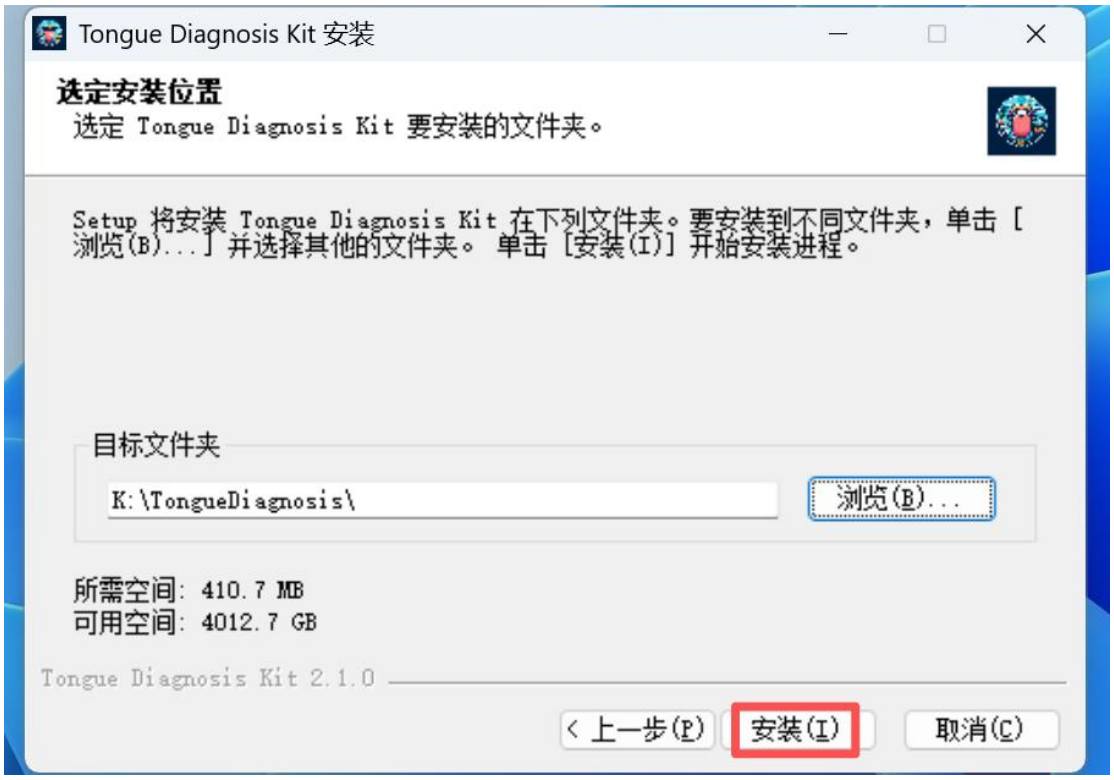
Picture 9-2: View the installation path

Select the folder where you want to install the app, then click "OK".



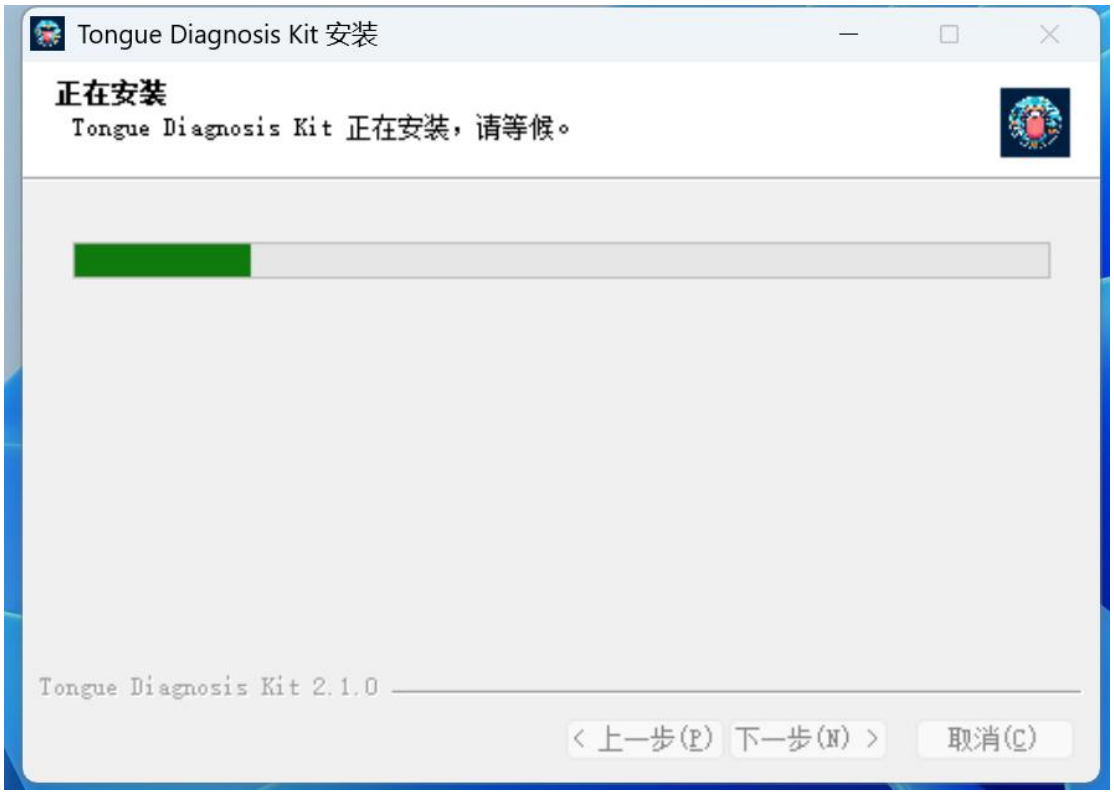
Picture 9-3: Select the installation path

After completing the selection, simply click "Install".



Picture 9-4: Selection Completed

Then wait for the installation process.



Picture 9-5: Installing

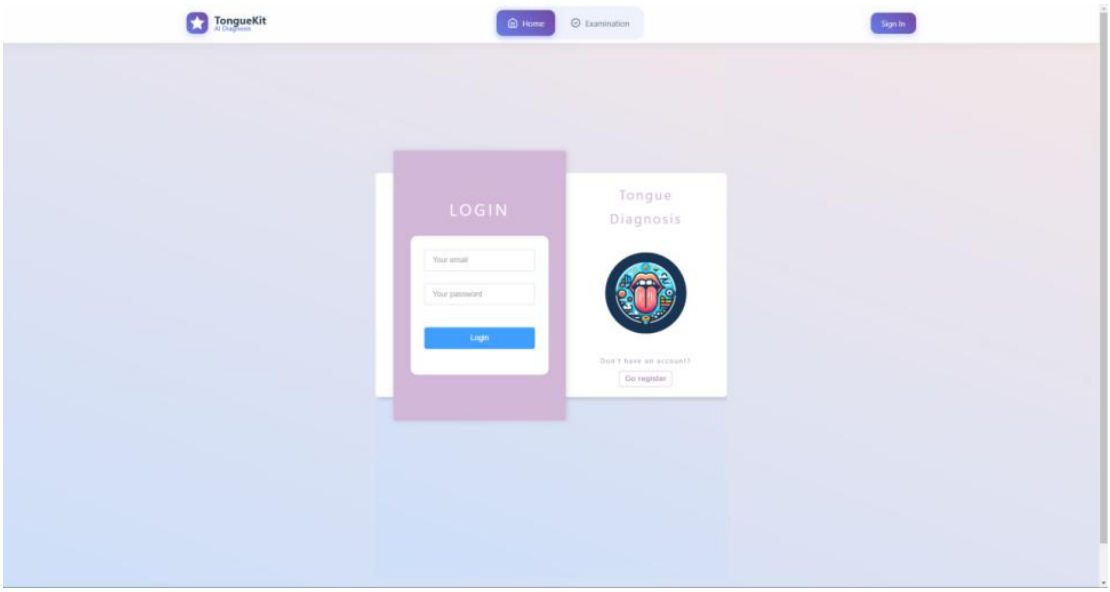
After completing the installation, you can click "Finish" to exit the installation guide. You can also check whether you want to launch the application immediately.

Picture 9-6: Installation Completed

9.2 Register & Login Page

After the user successfully starts the page, they will directly enter the login interface. If the user does not have an account to log in, they need to click the "Go register" button to reach the registration interface. The user can complete the registration by filling in the email and password as prompted.

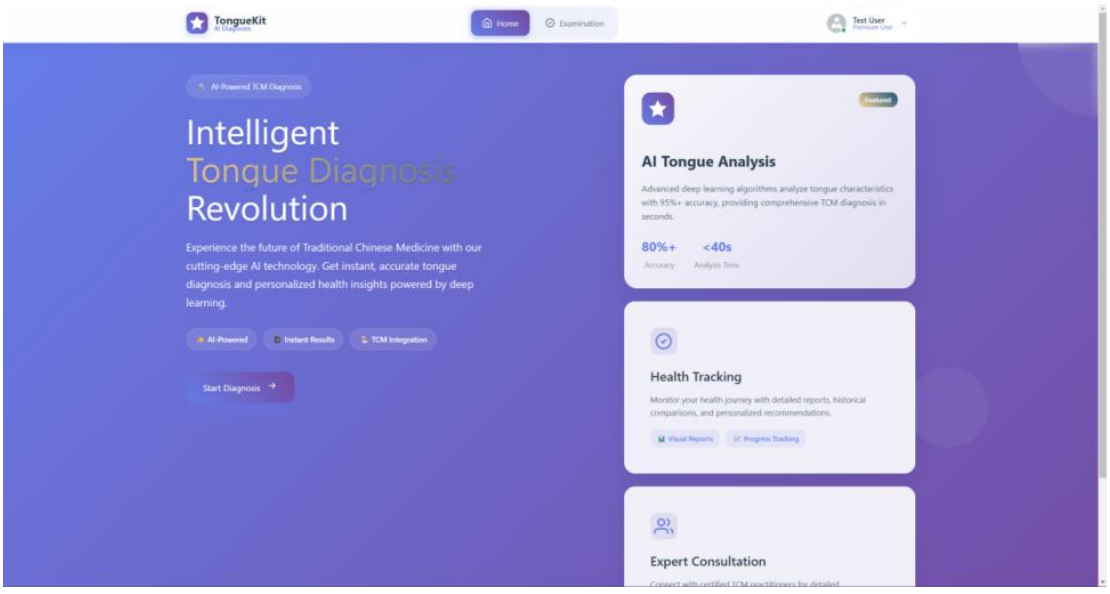
After the user registers an account, they click the "Go Login" text on the page. They will enter the login interface and use the email and password they just registered to log in. After successful login, they will enter the homepage of Paradox AI Tongue Diagnosis Kit.



Picture 9-7: Register & Login Page

9.3 Home Page

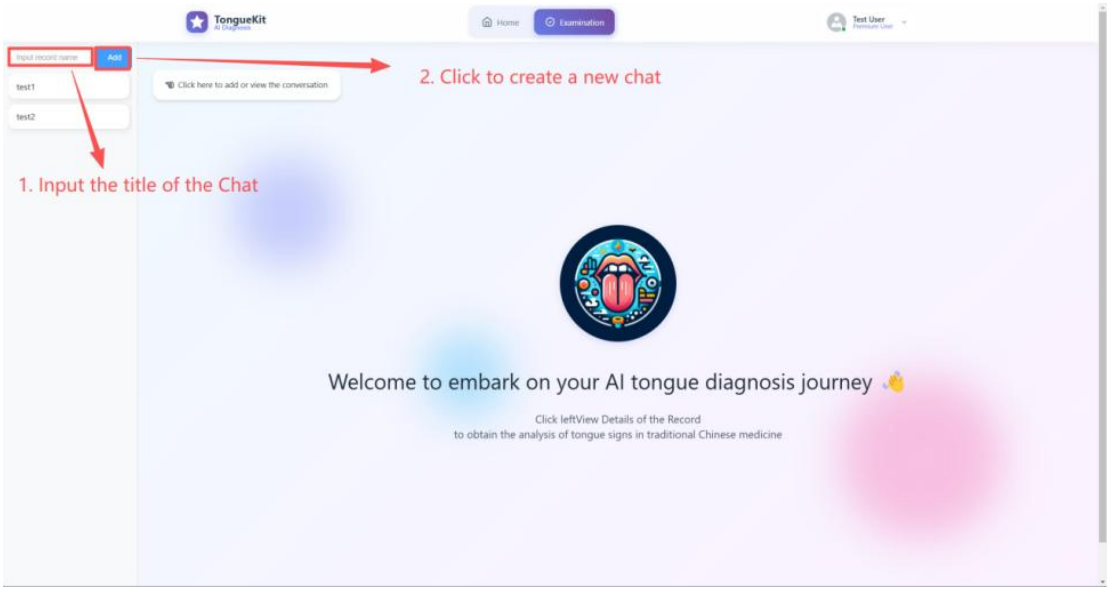
The homepage presents some basic information about Paradox AI Tongue Diagnosis. You can click "Start Diagnosis" or the "Examination" at the top to navigate to the tongue image analysis page.



Picture 9-8: Home Page

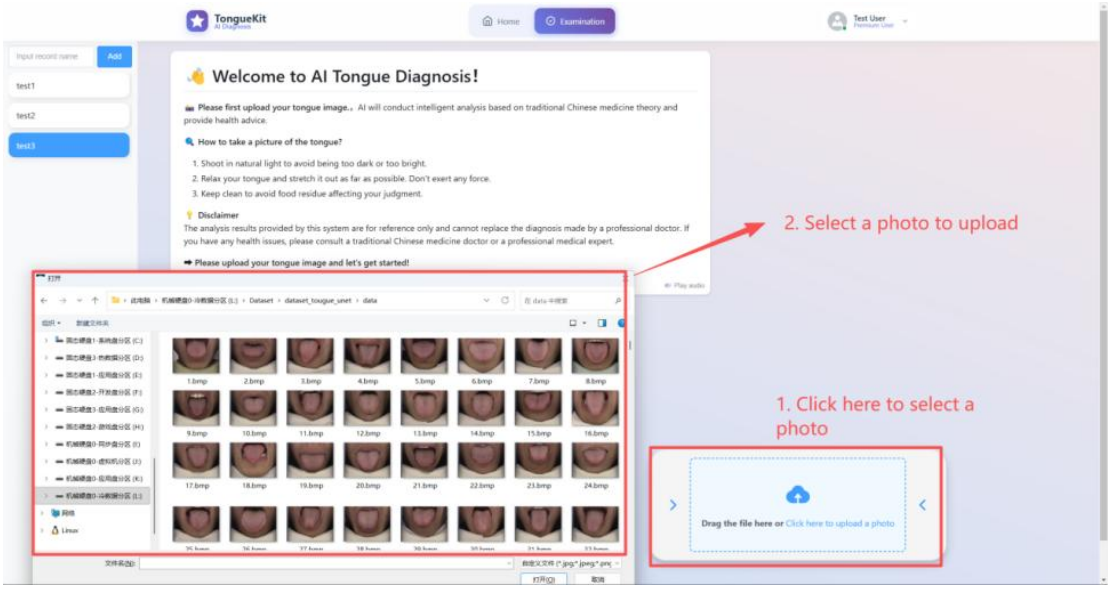
9.4 Intelligent Tongue Diagnosis

Enter the title in "Input Record Name" and click the "Add" button to load a new dialog.



Picture 9-9: Create a new chat

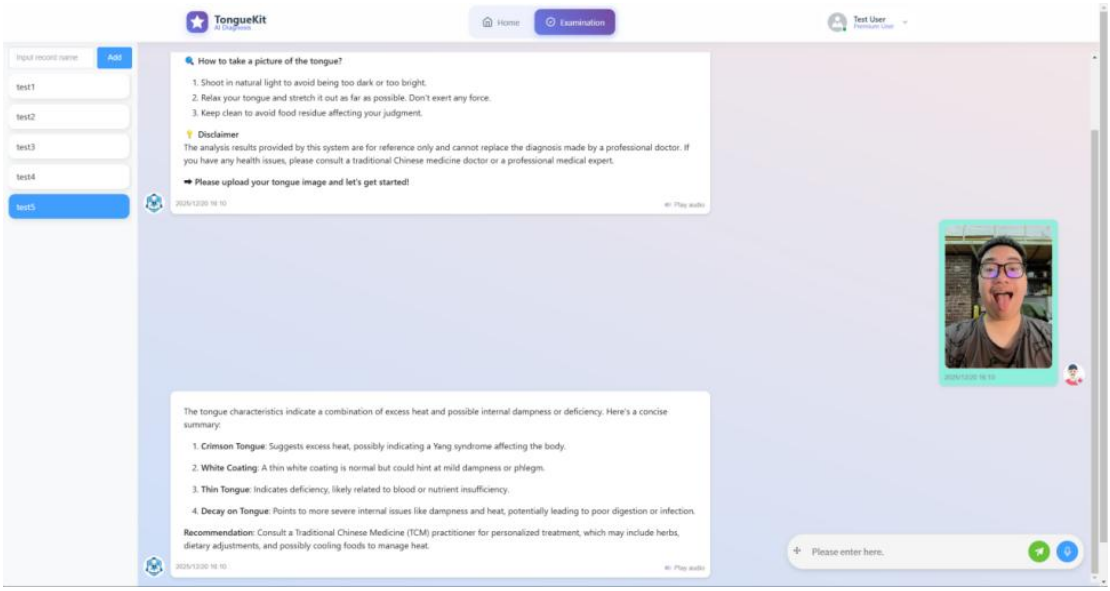
Upload the photo and click "Click here to upload a photo" to upload the tongue image to the application.



Picture 9-10: Select an image

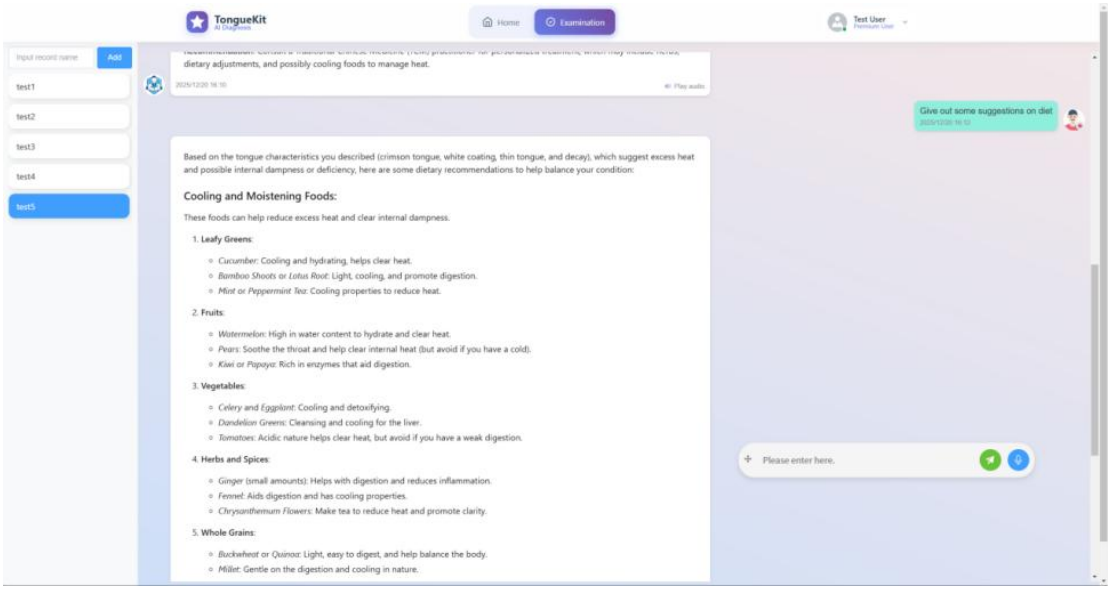
After the image upload is successful, the user needs to wait for about 25 seconds for the model to run.

After waiting for about 30 seconds, the left AI dialog box will start to return the user's tongue image status and the corresponding traditional Chinese medicine content of the tongue image reaction word by word.



Picture 9-11: Return the analysis results

After the AI provides a specific report, the user can still have a conversation with this AI model. At the same time, the model's backend stores all the chat records of this conversation, and the model can call the features of the previous tongue image to provide precise answers to the user's needs.

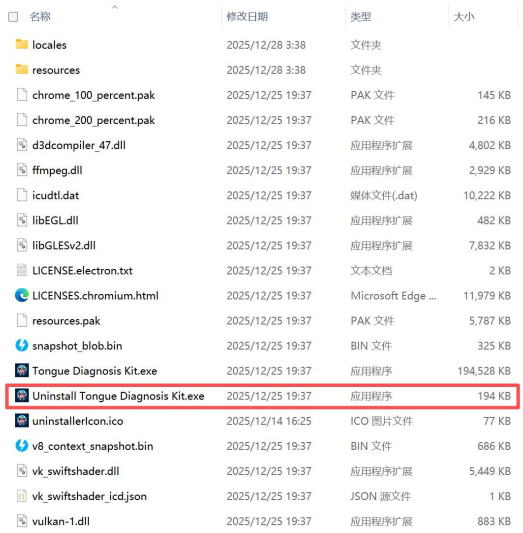


Picture 9-12: Continue the conversation

At the same time, the user can view the past chat records in the left column. Each chat records the time when the conversation occurred.

9.5 Uninstallation

We still take the desktop application corresponding to the Windows x86-64 installation package as an example. You can open the installation directory corresponding to the application, find the "Uninstall Tongue Diagnosis Kit.exe" application, and double-click to execute it.



Picture 9-13: Find the Uninstall executable file

After double-clicking, click "Next" to continue.

Picture 9-14: Uninstallation Program

Wait for the uninstallation to complete

Picture 9-14: Uninstalling

After the uninstallation is complete, simply click "Finish".

Picture 9-15: Uninstalled

Part10 Conclusion

10.1 Challenges & Limitations

10.1.1 Challenges

10.1.1.1 Data quality and diversity

The scarcity of traditional Chinese medicine (TCM) tongue image data represents a major challenge in current research. Due to the complexity of acquiring tongue images, the availability of high-quality data for training and validation is relatively limited. Moreover, the lack of unified annotation standards and significant variations in data sources (such as lighting conditions, imaging devices, and shooting angles) result in discrepancies between the training data distribution and real-world usage scenarios, thereby constraining the model’s generalization ability. In addition, the fine-grained discrimination of features such as tongue color, coating color, thickness, and greasiness relies heavily on professional expertise. Annotations provided by non-experts are prone to introducing noisy labels, which further limits the upper performance bound of the model.

10.1.1.2 The Accuracy Issue of Multi-model Integration

The current cascaded architecture of “YOLO-based localization → SAM-based segmentation → SE-ResNet-based classification” introduces error propagation and amplification across stages: inaccurate localization adversely affects segmentation, while segmentation boundary errors interfere with classification features. Moreover, differences in model versions, inference latency, and competition for computational resources among multiple models add engineering complexity, leading to increased performance variability and stability challenges.

10.1.2 Limitations

At the current stage, the main limitations of this project lie in areas such as medical knowledge and validation, annotation standardization, scenario generalization, and evaluation methodology. First, the team lacks a systematic background in traditional Chinese medicine, and interpretations of tongue features primarily rely on large language model generation, without yet forming a reusable “feature–syndrome/constitution” mapping or a closed-loop expert validation mechanism. Second, the four-dimensional feature labels are inherently subjective and inconsistent across data sources, lacking multi-rater agreement and a recognized gold standard,

which undermines the credibility of both training and evaluation. In addition, model performance remains unstable under complex imaging conditions such as low lighting, color bias, occlusion, and exposure of lips or teeth, while differences in device color gamuts and white balance have not been sufficiently calibrated, resulting in limited generalization capability. Finally, the evaluation framework is still largely based on a single data source and a limited set of metrics, lacking systematic benchmarks and error analysis across multiple datasets, devices, and population groups.

10.2 Future Enhancements

10.2.1 Model Performance Optimization

With respect to the current project outcomes, the model accuracy still has considerable room for improvement. We plan to explore multiple approaches to optimize model performance. For example, knowledge distillation (KD) can be employed to guide a lightweight student model with fewer parameters to mimic a high-capacity teacher model, thereby mitigating overfitting and enhancing the model’s generalization ability.

10.2.2 Optimize the User Experience

The current application interface logic is relatively simple. In response to the diverse needs of different user groups (such as general users and assisting physicians), we plan to further optimize the interface and functional design based on user requirements. For example, the application will be enhanced to allow direct access to the user’ s camera for tongue image acquisition, and the interface will be refined to better align with common user interaction habits.

10.2.3 Detailed test results

Due to the lack of a formal medical background among project members, the results currently displayed on the front-end interface are limited to symptom analyses directly related to tongue image features. In future work, we plan to combine multiple features to establish a four-dimensional “feature–symptom” mapping library, enabling more precise and informative recommendations for users. In addition, we will explore the visualization of historical records to provide users with more intuitive and user-friendly services

References

[1] Zhang, J., Xie, Y., Xia, Y., & Shen, C. (2021). Dodnet: Learning to segment multi-organ and tumors from multiple partially labeled datasets. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 1195-1204).

[2] Ji, W., Yu, S., Wu, J., Ma, K., Bian, C., Bi, Q., … Zheng, Y. (2021). Learning Calibrated Medical Image Segmentation via Multi-rater Agreement Modeling. 12336–12346. doi:10.1109/CVPR46437.2021.01216

[3] Tang, Q., Yang, T., Yoshimura, Y., Namiki, T., & Nakaguchi, T. (2020). Learning-based tongue detection for automatic tongue color diagnosis system. Artif. Life Robot., 25(3), 363–369. doi:10.1007/s10015-020-00623-5

[4] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (06 2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. 4510–4520. doi:10.1109/CVPR.2018.00474

[5] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

[6] Girshick, R., Donahue, J., Darrell, T., & Malik, J. (11 2013). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. doi:10.1109/CVPR.2014.81

[7] Brown, M., Gunn, S. R., & Lewis, H. G. (1999). Support vector machines for optimal classification and spectral unmixing. Ecological Modelling, 120(2-3), 167-179.

[8] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[9] Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Cham: Springer international publishing.

[10] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).

[11] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... & Girshick, R. (2023). Segment anything. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 4015-4026).

[12] 黄恩铭, 谭敏敏, & 袁晓琳. (2023). 基于舌象特征的中医体质自动辨别系统研究. 现代计算机, 29(7), 116-120. http://dx.chinadoi.cn/10.3969/j.issn.1007-1423.2023.07.021.

[13] 江涛.(2020).基于深度学习的舌象智能诊断研究与应用(博士学位论文,上海中医药大学).博士 https://doi.org/10.27320/d.cnki.gszyu.2020.000042.

[14] Ma, C., Zhang, P., Du, S., Li, Y., & Li, S. (2023). Construction of Tongue Image-Based Machine Learning Model for Screening Patients with Gastric

Precancerous Lesions. Journal of personalized medicine, 13(2), 271. https://doi.org/10.3390/jpm13020271

[15] Wu, L., Luo, X., & Xu, Y. (07 2020). Using convolutional neural network for diabetes mellitus diagnosis based on tongue images. The Journal of Engineering, 2020. doi:10.1049/joe.2019.1151

[16] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv preprint arXiv:1602.07360.

[17] 牛富泉.(2021).基于深度学习的中医舌像分类模型研究(硕士学位论文,电子科技大学).硕士 https://doi.org/10.27005/d.cnki.gdzku.2021.005334.

[18] Li, Z., Ren, X., Xiao, L., Qi, J., Fu, T., & Li, W. (2022). Research on Data Analysis Network of TCM Tongue Diagnosis Based on Deep Learning Technology. Journal of healthcare engineering, 2022, 9372807. https://doi.org/10.1155/2022/9372807

[19] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Commun. ACM, 60(6), 84–90. doi:10.1145/3065386

[20] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826).

[21] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[22] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.