# AI Agent on Sepsis Final Report

Intelligent Agent System for Sepsis Analysis and Intervention

Pang Liu

GitHub: `https://github.com/jeffliulab/Brandeis_CS149_Project2_Sepsis`
Video: `https://youtu.be/_B7MiawMT4U?si=QrLJYrmFOZn2nm1K`

May 2, 2025

## 1 Introduction

This report presents a prototype designed to showcase an easy-to-understand, intuitive small-scale AI Agent system. The system uses large language models (LLMs) to understand user inputs and employs Prompt Engineering to standardize execution instructions, enabling the AI Agent to execute strictly limited Python scripts through the Sepsis module. Additionally, the AI Agent integrates with the Open-Manus module, which operates in parallel with the Sepsis analysis module. After task completion, console outputs are sent back to the LLM, allowing it to summarize the completed work and creating an end-to-end user-task execution pipeline.

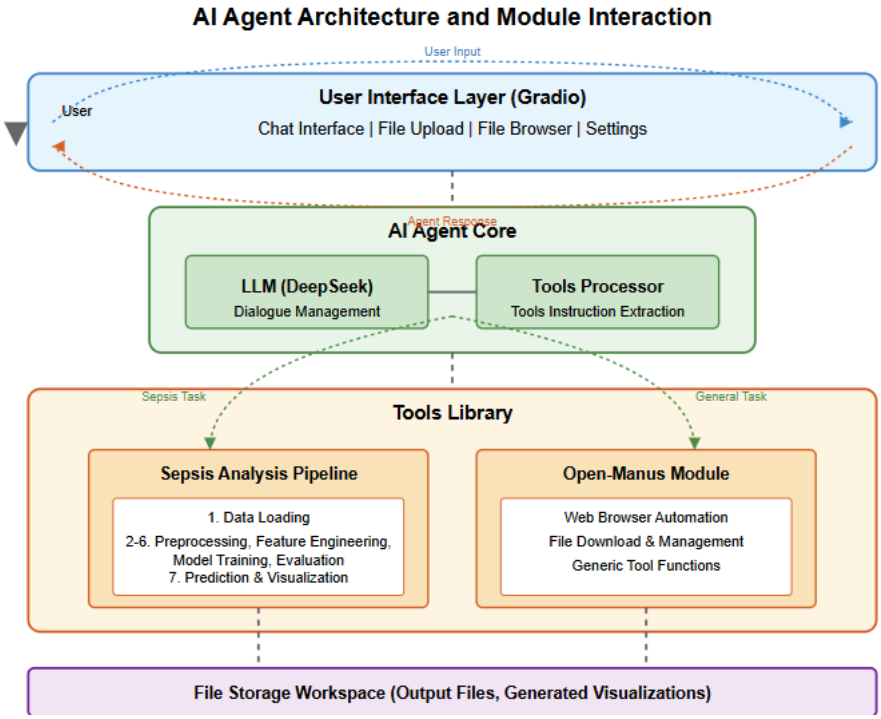


Figure 1: AI Agent Architecture and Module Interaction diagram showing the layered structure of the system: User Interface Layer (Gradio) at the top, AI Agent Core in the middle with LLM and Tools Processor components, Tools Library with Sepsis Pipeline and Open-Manus modules, and File Storage Workspace at the bottom.

# 2 Sepsis Pipeline

## 2.1 Pipeline Architecture

The Sepsis tool module consists of a sequential seven-step training pipeline for analyzing Electronic Health Records (EHR) to predict patient mortality:

1. **Data Loading** (`1_load.py`): Imports data from CSV files
2. **Missing Value Imputation** (`2_impute.py`): Fills missing data points
3. **Feature Engineering** (`3_feature.py`): Creates predictive features
4. **Model Training** (`4_train.py`): Trains a Random Forest classifier
5. **Model Evaluation** (`5_evaluate.py`): Assesses model performance
6. **Model Explainability** (`6_explain.py`): Interprets model decisions
7. **Prediction** (`7_predict.py`): Generates forecasts on test data

## 2.2 Pipeline Components

Each component of the pipeline serves a specific function in the sepsis analysis process:

### 2.2.1 Data Loading and Preprocessing

The pipeline begins by loading clinical data from ICU patients:

- Loading tabular EHR data from CSV files containing patient vitals and lab values
- Separating into training and test datasets
- Initial data validation and storage as intermediate files

In the imputation phase, missing values are handled:

- Conversion of zero values to NaN for relevant clinical measurements
- Application of median imputation to handle missing values
- Special handling for categorical variables

### 2.2.2 Feature Engineering

The feature engineering module transforms time-series clinical data into meaningful predictive features:

- Aggregation of temporal measurements for each patient using statistical functions:
  - Last value: capturing most recent measurements
  - Mean: representing central tendency
  - Maximum and minimum: capturing extreme values
- Feature naming that maintains traceability (e.g., `feature_last`, `feature_mean`)

### 2.2.3 Model Training

The training module implements a Random Forest classifier to predict 90-day mortality:

- Implementation of stratified train/validation split (80%/20%)
- Hyperparameter optimization via grid search with 5-fold cross-validation
- Hyperparameter space exploration:
  - Number of estimators: $\{100, 200\}$
  - Maximum depth: $\{\text{None}, 10, 20\}$
- Selection of optimal parameters based on F1 score

#### 2.2.4 Performance Evaluation

Model evaluation using standard classification metrics:

- Classification report including precision, recall, and F1-score
- ROC curve analysis and AUC calculation
- Detailed performance metrics by class

#### 2.2.5 Model Explainability

SHAP (SHapley Additive exPlanations) analysis for model interpretation:

- Random sampling of training instances for computational efficiency
- Calculation of feature contribution to predictions
- Visualization of feature importance via mean absolute SHAP values
- Generation of SHAP summary plots

#### 2.2.6 Prediction on Test Data

Application of the trained model to unseen test data:

- Generation of mortality predictions on the test dataset
- Production of probability estimates for each patient
- Statistical analysis of prediction distributions
- Visualization of results through distribution plots

### 2.3 Training and Evaluation Results

The pipeline demonstrates robust performance in predicting sepsis-related mortality, with the following evaluation metrics:

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| 0 (Survive) | 0.96 | 1.00 | 0.98 | 12216 |
| 1 (Death) | 0.98 | 0.83 | 0.90 | 2908 |
| Accuracy | | | 0.96 | 15124 |
| Macro avg | 0.97 | 0.91 | 0.94 | 15124 |
| Weighted avg | 0.96 | 0.96 | 0.96 | 15124 |

Figure 2: Classification Report for the Random Forest Model

The model achieved an AUC of 0.9845, indicating excellent discriminative power between survival and mortality cases.

### 2.4 Model Explainability Results

SHAP analysis revealed the most important features influencing mortality predictions:

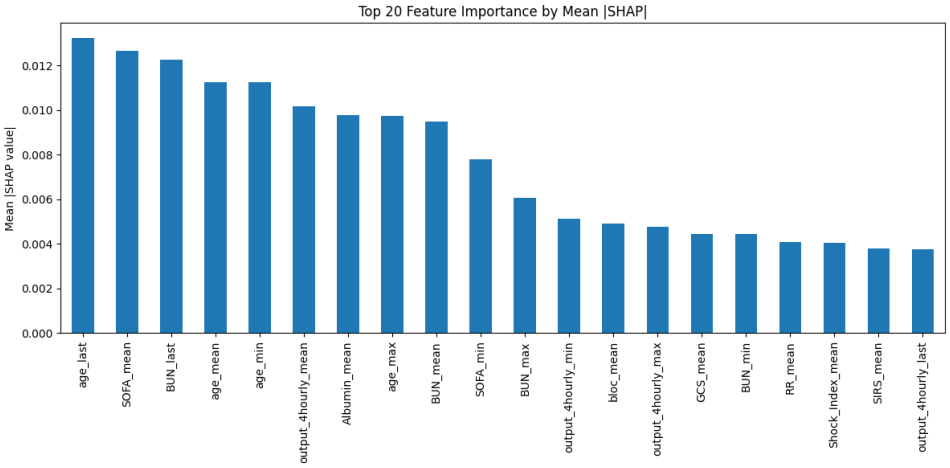


Figure 3: Top 20 Feature Importance by Mean —SHAP— Value

The top three influential features were:

- `age_last`: The patient's last recorded age
- `SOFA_mean`: Mean Sequential Organ Failure Assessment score
- `BUN_last`: Last recorded Blood Urea Nitrogen level

The SHAP summary plot further illustrates how feature values affect predictions:

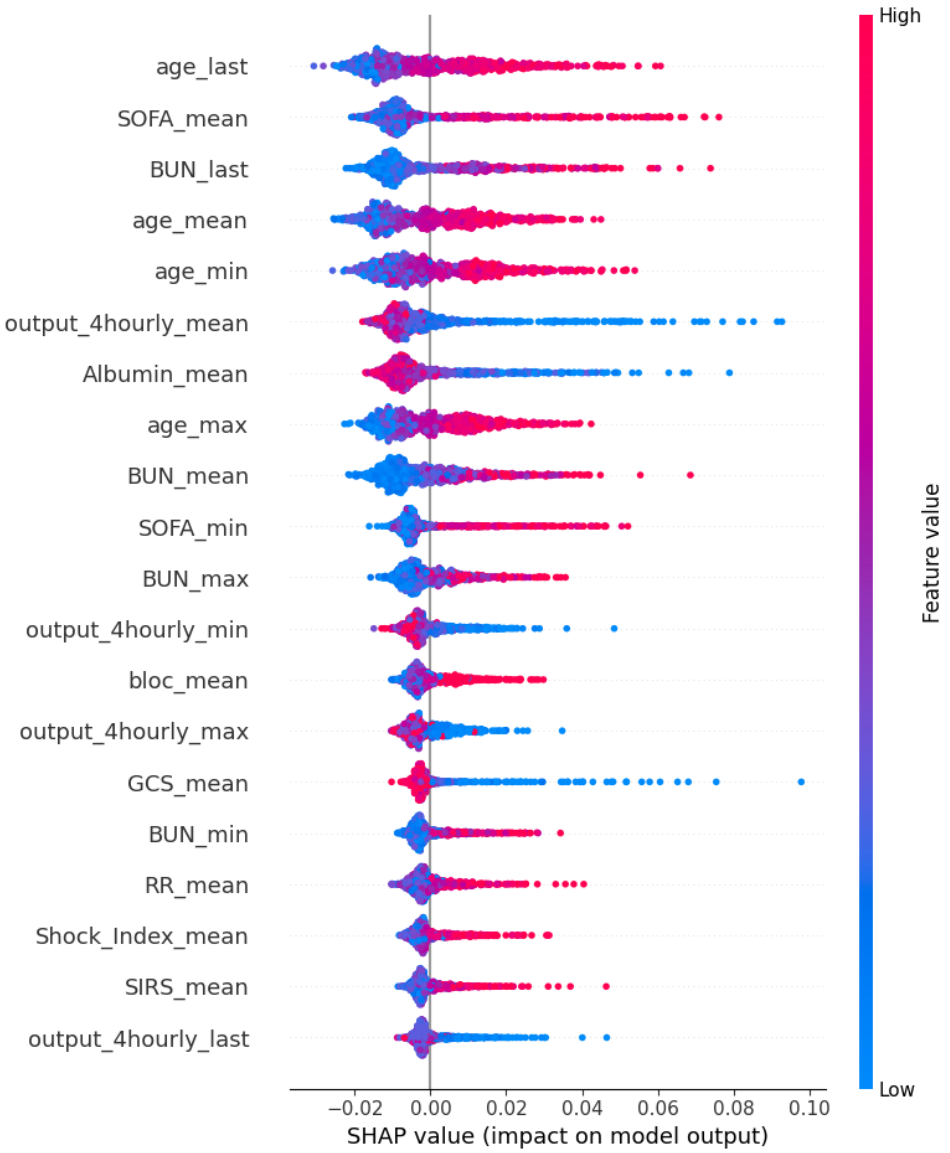


Figure 4: SHAP Summary Plot Showing Feature Impact Distribution

In this visualization, each point represents a sample, with color indicating feature value (red = high, blue = low). Position on the x-axis shows the SHAP value impact on the model output, with positive values increasing mortality prediction probability.

## 2.5 Prediction Results

When applied to the test dataset, the model generated the following prediction distribution:

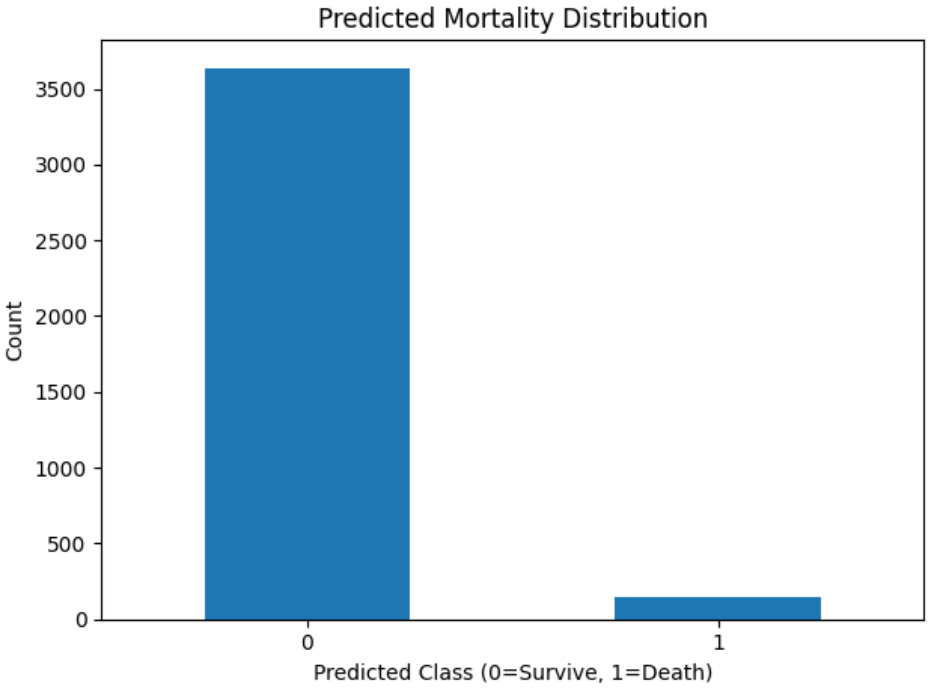


Figure 5: Distribution of Mortality Predictions on Test Data

The probability distribution of mortality predictions shows:

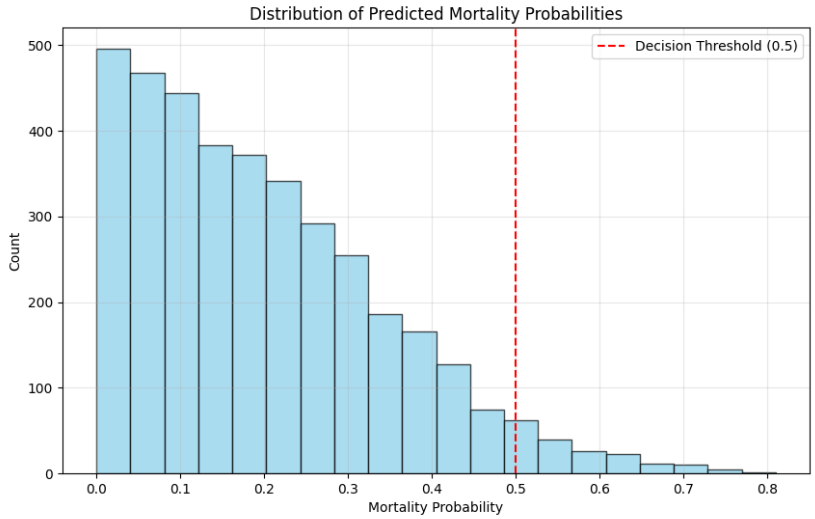


Figure 6: Distribution of Predicted Mortality Probabilities

Key observations from the prediction results:

- The majority of patients were predicted to survive (class 0)
- The mortality probability distribution is skewed toward lower values
- A decision threshold of 0.5 was used to classify patients (red dashed line)
- The model provides confidence levels for each prediction, enabling risk stratification

# 3 Sepsis Module in AI Agent Usage

## 3.1 User Interaction Interface

After opening the AI Agent interface, users can chat with the Agent. When the Agent (LLM) determines that the user's input relates to a Sepsis analysis task, the Sepsis tool module is activated:

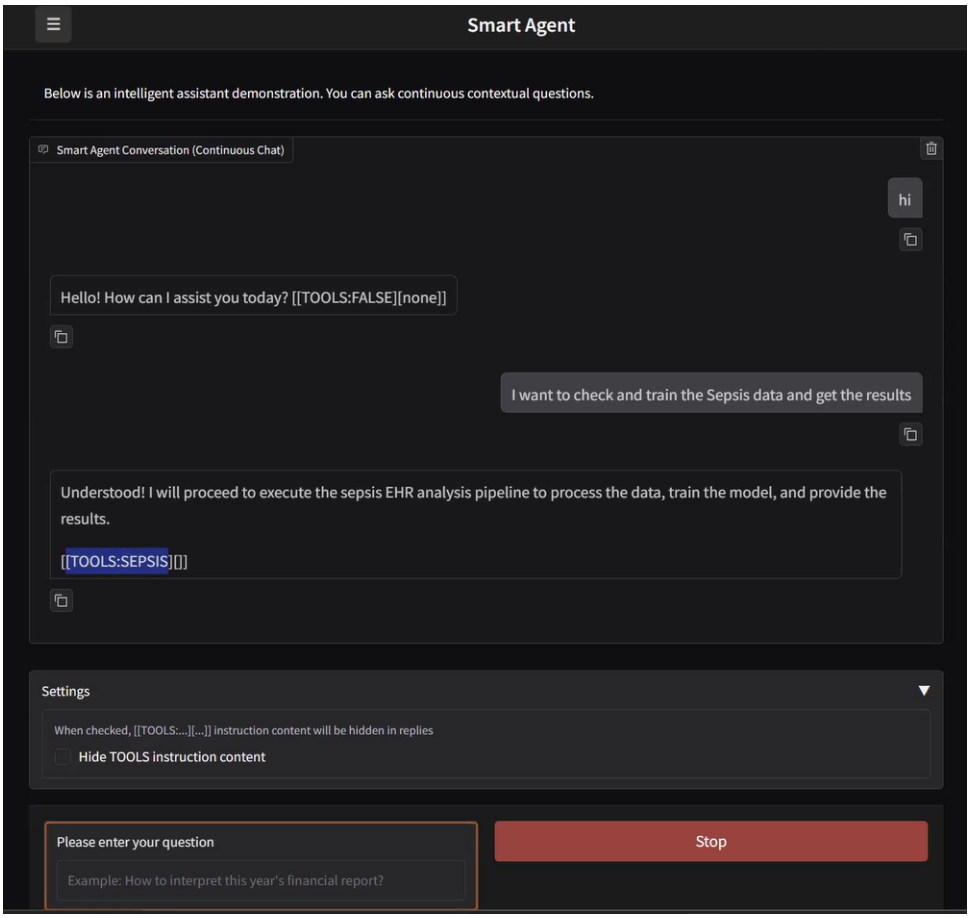


Figure 7: Interface: Chat with AI Agent.

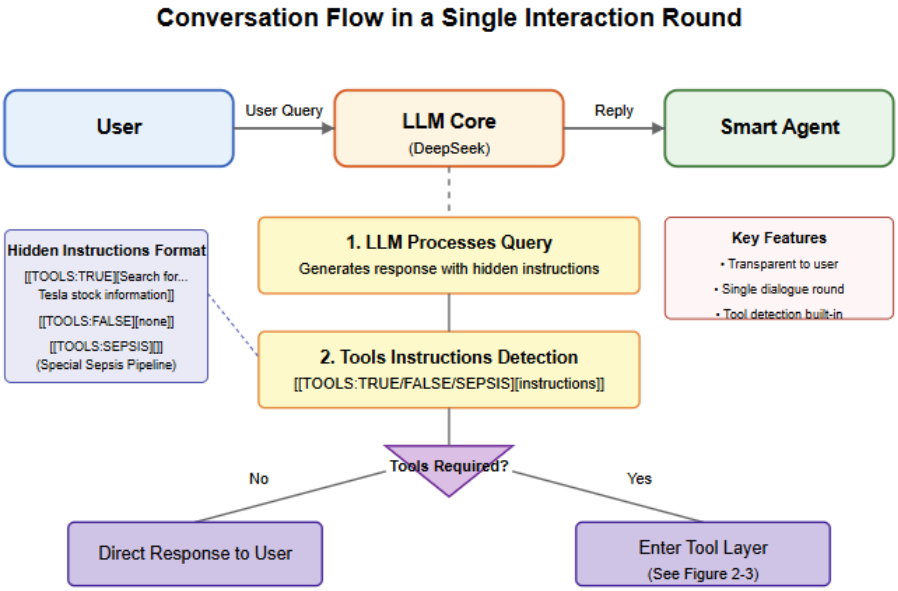


Figure 8: Conversation Flow

## 3.2 Pipeline Execution and Results

As the pipeline executes, each of the seven steps processes the data sequentially, storing intermediate outputs and generating visualizations. Upon completion, the console outputs are collected and sent back to the LLM, which summarizes the task results:

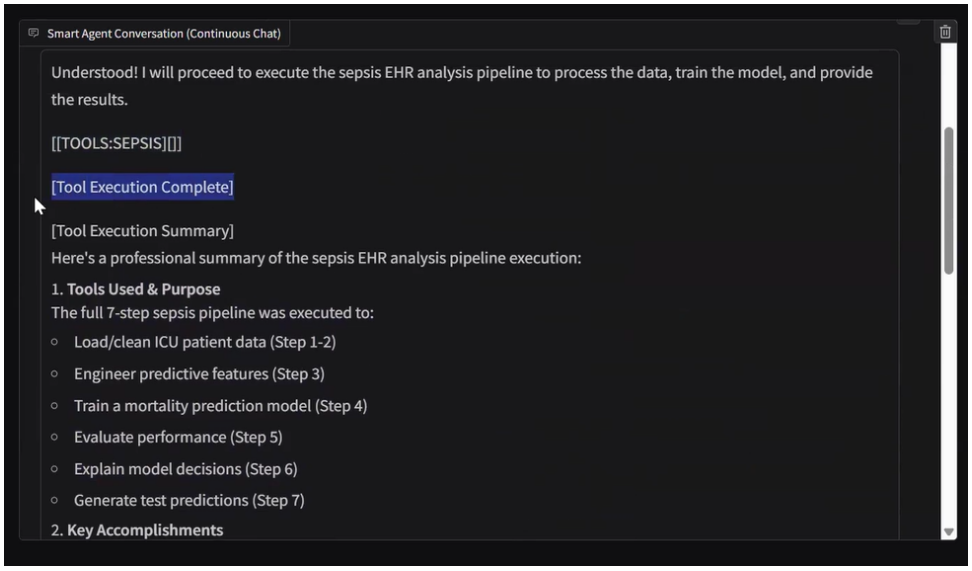


Figure 9: Summary by AI Agent (Interface)

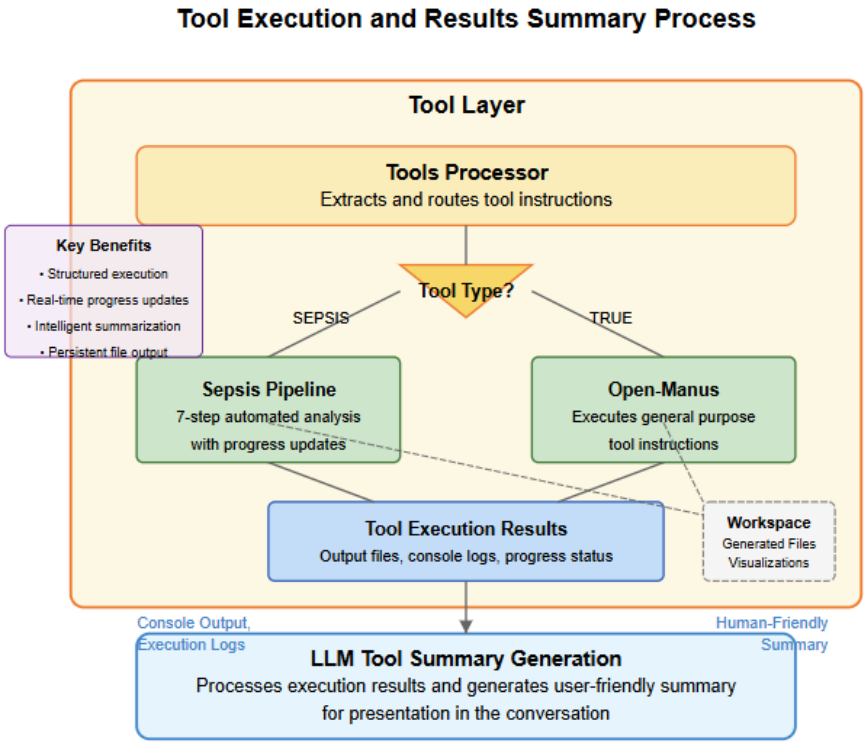


Figure 10: Summary by AI Agent (Real Flow)

# 4 Reflection: Tool-Free AI Agent Design

In this project, the first layer AI Agent corresponds to the LLM itself, possessing general knowledge capabilities. This forms the conversation layer.

The second layer is the tool layer, where various modules such as OpenManus, LangChain, or specific execution scripts like the Sepsis module are invoked.

After tools in the second layer complete their tasks, they report results back to the first layer, which determines whether the tasks were executed correctly.

The decision to not use LangChain or OpenManus directly as the first interaction layer stems from the belief that leveraging the LLM layer's comprehensive understanding capabilities is more important. Current capabilities in LangChain and OpenManus already include allowing LLMs to write and execute Python scripts.

If the following assumptions hold true: LLMs possess sufficient context understanding and memory capabilities, demonstrate precise programming skills, and have access to an efficient memory storage area, then the tool layer may no longer require specific tools, only interfaces. The first-layer LLM could dynamically design code and utilize these interfaces to complete tasks. For complex tasks, specific code tools or even low-code containers would be unnecessary in the second layer; with efficient memory storage and precise programming skills, LLMs could accomplish various tasks on demand.

This design places higher demands on the LLM itself, but I believe that overly complex AI Agent tool chain designs will not be the ultimate solution. As LLMs continue to enhance their programming capabilities and context length, what AI Agents will need is not so much "how to do things" but "what not to do"—more constraints.

Some projects approximating tool-free design already exist, such as Google DeepMind's AutoRT. Tool-free designs face challenges like the lack of robust process mechanisms and difficulties in debugging when errors occur. Given my limited mastery of LLMs, this section represents more of a structural reflection based on the "essence of intelligence": should we build a highly integrated mechanical arm for specific tasks, or create a humanoid intelligent entity that can adapt to human society? For industry, the former is undoubtedly more efficient and practical; but for the development of artificial intelligence, the ethically controversial latter may be the ultimate goal.

# 5 Conclusion

This project demonstrates a functional prototype of an AI Agent system for sepsis analysis and intervention. By integrating LLMs with specialized tools through a two-layer architecture, the system achieves intuitive user interaction while maintaining the ability to execute complex analytical tasks. The Sepsis pipeline demonstrates promising results in predicting sepsis cases, with potential applications in clinical settings.

The reflection on tool-free design points to future possibilities as LLM capabilities continue to advance. This project serves as a starting point for exploring more sophisticated AI Agent architectures that balance specialized functionality with general intelligence capabilities.