Design Documents

*Abstract*—This document is to refine the design of the XAI service and software requirements specification tasks.

Keywords—Explainable AI, Responsible AI, Black-box, Machine Learning, Deep Learning, Microservices Architecture.

# Introduction

We design a software system that provides XAI as a service. We investigated the state-of-the-art XAI method and summarized the taxonomy. Based on the XAI taxonomy, we developed the XAI process. Then, we establish a microservice architecture for XAI services. Based on our previous work, we divided the microservices into four domains. They are Data Processing Domain, AI Model Domain, XAI Methods Domain, and Evaluation Domain. There can be multiple independent microservices inside each domain to focus on computation. We have introduced a coordination center layer for managing the instances, tasks and pipeline.

Our research found deployed AI models or XAI methods can only handle one data type. So the data type is the criterion to determine whether the microservice is a match. By refining this microservice architecture, it is possible to execute XAI methods on one of the three main data types, which are tabular, text, and images on selected AI models, to obtain explanations. The architecture diagram, data model class diagram, use case diagram and so on can be found in the diagram file. We introduced a design to record the provenance data. Please see the data model for details.

# user stories

According to our survey and case studies, the model and XAI method give the results base on the input data. The relevant dataset should be provided for both using the AI model and the XAI method. The process indicates that the model and the appropriate XAI method need to be selected. Subsequently, the model performs output prediction on the data, and the XAI method performs an explanation analysis of the model. Eventually, the explanation results can be evaluated. Overall, the provenance data are generated at each step of the operation. We collect the structured XAI provenance data for further studies.

The following four user stories are about task management.

## User Story 1: The user provides their own case study dataset

XAI services should target the individual needs of the user. The service expects users can upload their data sets to the server's database. Afterward, the AI model and XAI method can directly read the database to make predictions and explanations.

## User Story 2: Execute AI model and obtain predictions

XAI service should provide the user-selected AI model to predict the input data sets. The user should be able to review the results. AI models are necessary for XAI services, and AI models are constantly evolving. The service should allow users to select among the published model as microservices.

## User Story 3: Execute XAI methods and obtain explanations

Users should be able to obtain explanation results of the selected models and the dataset. For instance, some XAI methods can give the feature contribution value for tabular data. The results show which features in the tabular data has a decisive impact on the results. The user can obtain the feature contribution value as an explanation. For text data, the user can know which part of the text or works token is more important for the semantic result. For image data, XAI provides a saliency map indicating which area or object causes the model's prediction.

## User Story 4: Evaluation of the XAI methods}

After the user obtains the explanation results, the service can evaluate the selected XAI methods. The service can compute the consistency value of the XAI methods based on users' own data.

The following user story is pipeline management.

## User Story 5: Project Pipeline

Users can manage the XAI project pipeline, a workflow of multiple tasks. The pipeline can automatically execute the four tasks. It provides the result to the user and records provenance data.

The following user story is about managing service instances.

## User story 6: Manage service instance

In general, the microservices for tasks are provided by the XAI service. However, in some cases, users want to provide their own data instance for data privacy. Users can provide model service instances that they want to explain. User can test their own XAI method by providing an XAI method service instance in our service platform. User can have their own evaluation metric for the results. Users can register, update and delete service instances through service management. The registered instance can be deployed and used in tasks and pipeline.

# Requirements (for backend)

The previous section introduced the user stories and the functionality of the system. This section discusses the specification requirements to finalize technical details in software. For the content of the data, please refer to the class diagram.

## Software Requirements Specification for task operation

### Create Task

Description: Create a task.

Functional Requirements: user can create a task by giving the task name, task description, instance id, instance type, task owner and so on. The data model has the information list. Then, the service generates and records creation time and task\_id. Information stored in database and provenance data. The created task id, task name and identifying information are returned to the user.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Update Task

Description: Update the task set.

Functional Requirements: Similar to Create Task requirements, the user can update the setting according to the provided task id. Only the task owner can update their tasks. Creation time does not change. Record update time and history log.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Delete Task

Description: Delete the task.

Functional Requirements: The task owner can delete the task and relevant data.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Execute Task

Description: User can commend on executing task by given task id or task name.

Functional Requirements: With the task id or task name, the system can easily get information about this task. The first step is identifying the instance id and checking the instance type. Then, send an HTTP post request with setting information to the instance through RESTAPI. The instance is deployed as a microservice. The microservice should receive the post request and compute. Send a successful message to the front end.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend, coordination center

### Stop Task

Description: User can commend on stop task by given task id or task name.

Functional Requirements: The service computing takes uncertain time. Users can stop the task and update the setting. Therefore, the user should be able to stop a task. With the task id or task name, the system can send an HTTP post request with a stop command to the instance through RESTAPI. Send a successfully stopped message to the front end.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, front end.

External Interface: frontend, coordination center

### Query Task Status

Description: The user can check the current running state of the task.

Functional Requirements: With the task id or task name, the system checks the task status in the running, stopped and error. Then feedback to the user.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend, coordination center

### Present Task Output

Description: Get the computed result from the database.

Functional Requirements: After running the task, the results are saved in the database. Users can retrieve the result from the database by task id.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend

## Software Requirements Specification for Pipeline

### Create Pipeline

Description: The user can create a pipeline.

Functional Requirements: user can create a pipeline by giving the pipeline name, pipeline description, pipeline owner and so on. Collect the four instance id and other information to make the pipeline. The data model has a detailed information list. Then, the service generates and records creation time and pipeline\_id. The created identical information is return to the user.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Update Pipeline

Description: Update the pipeline set.

Functional Requirements: The user can update the setting according to the provided pipeline id. Only the pipeline owner can update their pipelines. The creation time does not change. Record update time and history log.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Delete Pipeline

Description: Delete the pipeline.

Functional Requirements: The pipeline owner can delete the pipeline and relevant data.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface: frontend, coordination center

### Execute Pipeline

Description: User can commend on executing pipeline by given pipeline id or pipeline name.

Functional Requirements: With the pipeline id or pipeline name, the system can get information about this pipeline. The first step is identifying the instance id and checking the instance type. Then, send an HTTP post request with setting information to the instance through RESTAPI. The instance is deployed as a microservice. The microservice should receive the post request and compute. Send a successful message to the front end.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend, coordination center

### Stop Pipeline

Description: User can commend on stop pipeline by given pipeline id or pipeline name.

Functional Requirements: Computing takes uncertain time. Users can stop the pipeline and update the setting. Therefore, the user should be able to stop a pipeline. With the pipeline id or pipeline name, the system can send an HTTP post request with a stop command to the instance through RESTAPI. All of the running tasks under the pipeline should be stopped. The intermediate results should be deleted to prevent the influence of another execution. Send a successfully stopped message to the front end.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend, coordination center

### Query Pipeline Status

Description: The user can check the current running state of the pipeline.

Functional Requirements: With the pipeline id or pipeline name, the system checks the task status in the running, stopped and error. Then feedback to the user.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend, coordination center

### Present Pipeline Output

Description: Get the computed result from the database.

Functional Requirements: After running the pipeline, the results are saved in the database. Users can retrieve the result from the database by pipeline id. Users can also access the provenance data according to the design.

Tech Stack: HTTP, REST API, NoSQL Database server, Microservice, frontend.

External Interface: frontend

## Software Requirements Specification for functionality

### Upload data

Description: Allow users to upload data. Prepare data for the model and XAI.

Functional Requirements: The user can upload data samples through the front end. The data format is described in the data model class diagram. Storage in database.

Tech Stack: HTTP, REST API, NoSQL Database server

### Get dataset

Description: The user can view the list of data through the Task\_id. Allow the user to check the dataset for the task.

Functional Requirements: After uploading data, the user can send the GET request through HTTP with the label name, and the database server can respond with the list of data which belongs to the user.

Tech Stack: HTTP, REST API, NoSQL Database server

External Interface Requirements: DB server, UI, Model server, XAI server, Evaluation server.

### Execute AI model

Description: Compute model prediction. Input data samples, and get model prediction results.

Functional Requirements: The user can send the task name to the AI model through an API request. The model server can get the dataset from the DB server by the task name. Then, the AI model returns the prediction results. The results go back to the user through API and also store in the database.

Tech Stack: HTTP, REST API, NoSQL Database server, AI model(e.g. Pytorch)

External Interface Requirements: DB server, UI, XAI server.

### GET AI model

Description: Transport, the model to XAI

Functional Requirements: The XAI service needs to obtain a model for the model-specific XAI method. Use PyTorch to save the model file. And allow the XAI server to request this model file.

Tech Stack: HTTP, REST API, AI model(e.g. Pytorch)

External Interface Requirements: XAI server

### Execute XAI method

Description: get explanations

Functional Requirements: Get the task through API with a task name. The server retrieves the dataset through the database server by task name. If the XAI method needs a model, get the model from the model server. After computing, save the explanations to the database. The save contains the task name and results.

Tech Stack: HTTP, REST API, XAI method

External Interface Requirements: AI model server, database server

### Check XAI status

Description: Check the status of XAI

Functional Requirements: Get the task status through XAI server API with a task name. Return to the current state of tasks.

Tech Stack: HTTP, REST API, XAI method

External Interface Requirements: UI

### Get explanation results

Description: retrieve explanation results from the database server.

Functional Requirements: Get the explanation results through database server API with a task name. Return the XAI results.

Tech Stack: HTTP, REST API

External Interface Requirements: UI

### Execute evaluation

Description: main activity for evaluation

Functional Requirements: Request a task through API. The task contains the selected XAI results name. Get the explanation results through the database server API with the name. Compute the consistency according to the data type. Then save the score value with the task id.

Tech Stack: HTTP, REST API, the Consistency evaluation metric

External Interface Requirements: UI, database server

### Check evaluations status

Description: Check the status of evaluations

Functional Requirements: Get the task status through the evaluation server API with a task name. Return to the current state of evaluation tasks.

Tech Stack: HTTP, REST API, the evaluation metric

External Interface Requirements: UI

### Get evaluation

Description: retrieve evaluation results from the database server.

Functional Requirements: Get the results through database server API with a task name. Return the evaluation results.

Tech Stack: HTTP, REST API

External Interface Requirements: UI

# Microservice Architecture

Diagram

Description automatically generated

# Communication

## REST API Documentation

Components field: Refer to API JSON files.

## API gateways

[TODO]

# Frontend functionality

Refer to UI design

[TODO]

1. *(references)*

Appendix(Previous Versions)

Paths field:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Endpoint** | **POST** | **GET** | **PUT** | **DELETE** |
| /rawdb/{datatype} | Insert data |  |  |  |
| /rawdb/{sampleId} |  | Retrieve data | Update data | Delete data |
| /task/{datatype} | Create a task |  |  |  |
| /task/{taskid} |  | Retrieve the task | Update the task | Delete the task |
| /model/{model\_id}&  {task\_id} | Execute model with a task | Retrieve the task prediction result |  | Delete the pred task result |
| /xai/{xai\_id}&{task\_id} | Execute XAI with a task | Retrieve the task XAI result |  | Delete the XAI task result |
| /evaluation/{eval\_id}&{xai\_id}&{task\_id} | Compute evaluation results | Retrieve the evaluation result |  | Delete the evaluation result |