A Framework for Healthcare Data Integration Based on Model-as-a-Service

Yujie Fang[1], Chao Gao[1], Xinyue Zhou[1], Jianmao Xiao[2], and Zhiyong Feng[1(✉)]

[1] Tianjin University, Tianjin, China
{fyj0328,gc_2019,zhouxinyue,zyfeng}@tju.edu.cn
[2] Jiangxi Normal University, Nanchang, Jiangxi, China
jm_xiao@jxnu.edu.cn

Abstract. More and more IoT detection devices are entering into healthcare domain. They collect remote data through the MQTT protocol. Along with the chaos of server subscription, data format, message structure, and content parsing, how to realize the end-to-end "model-as-a-service" in the healthcare scenario is an issue worthy of further study. This paper designs and implements a healthcare data integration framework that integrates the whole process from detected data subscription to model training deployment and data analysis automatically based on workflow, and provides users with a low-code workflow configuration method. First, this paper defined a custom description language for the workflow of the integration problem. Next, fully considering the situation of message parsing and storage of different devices, we build an end-to-end healthcare integration framework that realizes the dynamic management of access data and subscription clients. In addition, it provides customization and AutoML-based automation options to select machine learning models and parameters. Finally, the experiment shows that the framework completes the dynamic subscription, parsing, storage, model training and deployment, and data analysis of various device messages. This framework can further integrate techniques such as streaming data analysis and deep learning automation to perform complex tasks in different scenarios like real-time data analysis of elderly care and medical diagnosis.

Keywords: IoT · Healthcare · Model-as-a-Service · Workflow · Machine Learning · Data Analysis

1 Introduction

China has the largest number of elderly people in the world today, and the healthcare of the elderly is attracting more and more attention. Currently, elderly care in China has basically formed a combination of home, community, and institutional, where remote analysis of the elderly's physical data is critical.

The rapid development of IoT has brought new solutions to healthcare problems. A large number of IoT devices have entered into the healthcare domain, which is profoundly reshaping healthcare services. Nowadays, it is very convenient to collect physical data

Y. Cao and X. Shao (Eds.): MONAMI 2022, LNICST 474, pp. 154–168, 2023.
https://doi.org/10.1007/978-3-031-32443-7_11

such as blood pressure, heart rate, blood sugar as well as eye movement from various detection devices and sensors [1]. These devices use the MQTT protocol to distribute data with greatly low power consumption [2]. Users can subscribe to the data, making it possible to remotely monitor and analyze the health status of the elderly [3].

In addition, there is a wide range of applications that combine IoT with various technologies and tools like data storage and machine learning to improve the efficiency of diagnosis and treatment, and assist doctors in their work [4]. Guided by "model-as-a-service", training collected data, deploying machine learning models can further improve the added value of the business [5].

In the healthcare scenario, the ultimate realization of "model-as-a-service" needs to consider a series of issues. For IoT communication protocols, device data reception, data parsing, data storage, and integration are chaos. Different IoT devices generally have their own independent vertical IoT architecture. Device data will be published to their respective MQTT servers. Therefore, data consumers need to dynamically deploy and manage a large number of subscription clients to complete data subscription and collection tasks for multiple devices according to the changing situation of the collected devices. For data analysis, the subsequent complete data parsing, storage, and machine learning analysis process need to be done in collaboration with multiple technologies and systems. Machine learning-based models need to be deployed and trained separately on demand. Thus, the lack of seamless integration and management among IoT middleware, data systems, and machine learning systems is one of the main challenges in implementing "model-as-a-service" for remote health detection and data analysis services. Since data collection, data storage, and integration as well as data analysis of IoT devices are relatively fixed, the use of workflow to integrate the IoT data processing process and automate various end-to-end "model-as-a-service" integration tasks can effectively reduce user's operational difficulty and learning costs. In summary, the contributions of this paper are as follows.

- We proposed a set of unified custom description language from the perspective of workflow to abstract and model the "model-as-a-service" problem in the healthcare scenario, and use the custom language to unify the description of different types of task flows, tasks, and related dynamic configuration resources in the integrated workflow.
- We specified the workflow analysis and processing process corresponding to the custom description language, which provided the basis for the realization of the workflow integration framework of "model-as-a-service".
- The functions provided by the design and implementation of the "model-as-a-service" workflow integration framework in the healthcare scenario: data subscription, data analysis, data storage, machine learning model training and deployment, data analysis, etc.
- Verification of the usability and ease of use of the integration framework: In the case of a healthcare scenario. The user defines a personalized workflow, uses this framework to dynamically subscribe, parse and store data for a variety of health testing devices, and utilizes The data set is used for model training and deployment, and the entire process of model loading and data analysis is completed by fetching data from different devices.

The subsequent part of this paper is structured as follows. Section 2 introduces the related work. Section 3 outlines the methodology of the framework implementation. Section 4 reports the experimental results. Section 5 concludes this paper and points out the future work.

2 Related Work

In the past few years, IoT sensors and devices have been gradually applied in areas such as health and disease detection, and the reliance on IoT technologies for health and disease detection is increasing [6]. For example, the Mi Smart Band can detect the heart rate of the human body and diagnose conditions such as arrhythmia in time [7]. Yang et al. [8] proposed an ECG monitoring system, which collects and transmits ECG signals using Wi-Fi in the IoT infrastructure, and uses HTTP and MQTT to transmit and collect data in the IoT cloud server. In the work of Laport et al. [9], the signals from the EEG sensors are used to classify the eye status, where the ESP8266 Wi-Fi module is used to send the raw EEG signals and the MQTT protocol handles the communication between the different IoT agents. Moreover, fall detection is an important application in healthy aging. Yacchirema et al. [10] proposed a 3D-axis accelerometer embedded in a 6LoWPAN device and used the MQTT protocol to send emergency alert notifications to caregivers. Kadarina et al. [11] applied blood oxygen sensors to collect heart rate and blood oxygen from mothers and infants and then uploaded the data to their own IoT platform via the MQTT platform. The proliferation of detection sensors and devices such as these is the backdrop for this work, and they provide a variety of data sources needed for health and disease detection and analysis using techniques such as machine learning [12]. However, since the detection of health or disease often involves a large number of analytic metrics, the functionality of a single sensor or device cannot cover all the analytic metrics alone, and therefore remote data collection from multiple devices is often required for further analysis.

Table 1. Workflow Reference Model

Application	Workflow Development Language	Process Definition Forms	Workflow Framework
Business Process Management	Java	XML	jBPM Activiti
Data Task Management	Python	Python Python YAML	Ariflow Perfect DVC

The healthy aging integration framework proposed in this paper is based on workflows [13], whose main purpose is to effectively organize different tasks to collaboratively

accomplish the set goals, and is applicable to scenarios and domains where a large number of complex tasks exist. Based on different directions, a series of different workflow frameworks and process definition methods are derived [14], as shown in Table 1.

The framework uses perfect core as a workflow engine kernel to call data access, data integration, and data analysis sub-modules to achieve a complete end-to-end "model-as-a-service" process.

3 Methodology

Guided by the idea of "model-as-a-service", the IoT data integration solution for the healthcare domain opens up the entire end-to-end process from data subscription, analysis, storage to model training, deployment, analysis, and service.

The solution mainly faces the following technical challenges. This framework organically bonds and expands different technical links, and provides a necessary support for the implementation of automated applications of remote health data collection and analysis.

- The integration of health data involves multiple types of business such as IoT subscription, database storage, data analysis, etc. How to formalize the workflow issues to represent the basis for subsequent custom language design and framework design.
- The business process workflow is generally defined in XML form and the data analysis workflow is generally defined in python. How to design a custom description language for integration solutions to take into account the legibility of business process workflow and the good support of python workflow framework for machine learning tasks, and provide it for healthcare industry personnel.
- Designing the corresponding parsing process to execute the workflow tasks using the workflow engine.

3.1 Formalization of Integration Program Issues

The workflow is described by a directed acyclic graph $G = (T, E)$, where $T = \{t_1, t_2, \ldots t_N\}$ is the set of N workflow tasks, and E denotes the dependency between workflow tasks. The subscription client of the MQTT data should remain continuously open after the task is opened, and the data analysis task depends on the data subscription [15]. The execution of the task should not depend on the end of the data subscription task. Therefore, it is appropriate to split the health aging IoT data integration workflow into a data subscription task flow and multiple data analysis task flows.

Definition 1 (Healthcare IoT data integration workflow): The senior care data integration workflow can be written as a two-tuple *Flow*=<*flowName, taskFlowSet*>, where *flowName* denotes the unique representation of the workflow and *taskFlowSet* denotes the set of task flows contained in the workflow.

Definition 2 (Task Flow): Task flow can be divided into data subscription task flow and data analysis task flow in terms of task flow type. Task flow can be divided into timed workflow and one-time workflow in terms of execution conditions. A task flow can be represented as a six-tuple *taskFlow*=<*taskFlowName, taskFlowType, Resource,*

scheduleType, scheduleValue, taskSet>, where *taskFlowName* indicates the unique name of the task flow, and *taskFlowType* denotes the type of the task flow, *Resource* denotes the dynamic task resource on which the task flow depends, *scheduleType* denotes the type of task flow execution, *scheduleValue* denotes the timed execution interval, and *taskSet* denotes the set of tasks contained in the task flow.

Definition 3 (Task): A task flow consists of one or more tasks and can be defined as a four-tuple *Task*<*taskName, taskType, taskParmSet, taskState*>. Where *taskName* represents the unique representation of the task, *taskType* represents the specific type of the task, *taskParmSet* represents the set of parameters needed to execute the task, and *taskState* represents the completion status of the task.

Definition 4 (Task Parameter): A task parameter can be represented as a two-tuple, which can be written as *Parm*=<*ParmName, ParmValue*>, where *ParmName* represents the name of the parameter and *ParmValue* represents the value of the parameter.

3.2 Custom Description Language for Integration-Oriented Solutions

The Basque paradigm (BNF) is a specification for the definition of a set of computer language symbols expressed in a recursive way of thinking [16]. The BNF paradigm is used to give a strict syntactic definition of the custom description language of the scheme, which serves as a standard interface for the outside world to interact with this framework. The basic meta-symbols for BNF [17] and their descriptions are shown in Table 2.

Table 2. BNF main meta symbol description

Symbol	Explanation
X::=Y	X is defined as Y
<X> [X] {X} X|Y	X is required X is optional X is a repeatable option X or Y

The specific important tags and attributes in the custom description language are explained below.

IoTWF::=<[TaskFlow,…]|[TaskResource]>: The IoTWF tag is a root tag that can contain multiple TaskFlow tags or a TaskResource tag. The IoTWF tag is used to describe the workflow when it contains multiple TaskFlow tags, and to describe the dynamically configured resources for data subscription tasks when it contains a TaskResource tag.

A general form of a workflow defined by an IoT integration domain-driven language is given, as shown in Fig. 1.

Lines 1–8 define a data subscription task flow, and line 5 defines the address of the dynamic configuration resource on which the task flow depends. Lines 10–20 define a data analysis task flow. Lines 12–15 define a data query task, and lines 16–18 define a model prediction task.

```
<!--Data Subscription Task Flow Example-->
<IotWF>
    <TaskFlow taskFlowName = "FN_0001" taskFlowType = "SubscribeFlow" scheduleType = "Rolling" scheduleValue = "60">
        <task taskName = "t0" taskType = "DataCollection">
            <parm parmName = "resource">...</parm>
        </task>
    </TaskFlow>
</IotWF>
<!--Data Analysis Task Flow Example-->
<IotWF>
    <TaskFlow taskFlowName = "FN_0002" taskFlowType = "AnalysisFlow" scheduleType = "Once" scheduleValue = "0">
        <task taskName = "t1" taskType = "SQL">
            <parm parmName ="SQL">...</parm>
            <parm parmName = "outgoing">...</parm>
        </task>
        <task taskName = "t2" taskType = "M1Predict">
            <parm parmName = "joblib">...</parm>
        </task>
    </taskFlow>
</IotWF>
```

Fig. 1. General form of a Workflow

3.3 Workflow Analysis Execution Process

The parameters and task sets of different task flows are heterogeneous, and the execution process of the task flow engine is divided into three main steps: task flow parameter construction, task instantiation construction, and task flow instantiation execution, as shown in Fig. 2.

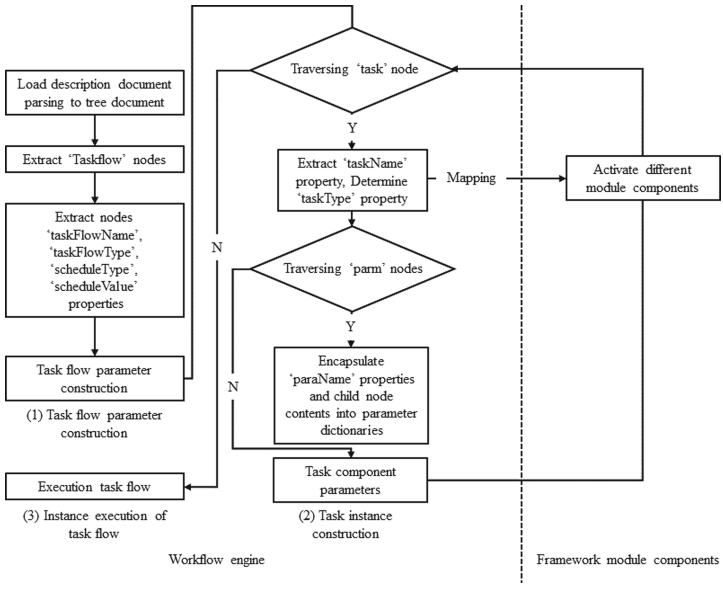

Fig. 2. Workflow Analysis Execution Flowchart

Task Flow Parameter Construction. The workflow engine first loads the description document and parses it, and then saves the parsed structure as a tree structure. After that, the TaskFlow node in the tree structure is extracted and the taskFlowName, taskFlowType, scheduleType, and scheduleValue properties under this node are obtained to

represent the name, type, timing type, and timing interval parameters of the task flow respectively for constructing the parameters of the task flow.

Task Instantiation Constructs. The next step is to construct the task collection under the task flow, first by traversing the task nodes under TaskFlow, extracting the taskName property as the task name parameter, extracting the taskType property, and calling different framework modules and components according to their property types. Then, by traversing the parm nodes under task, we obtain the parmName property and the content of the child nodes to encapsulate the parameters, which can be represented as a dictionary: TASK_PARM:{<key, value>,…} where key is the parameter name parmName attribute and value is the child node content parmValue. After completing the parameter encapsulation, different components are called and passed as parameters to complete the instantiation of a single task. Then the instantiation of the entire task set is completed by analogy.

Task Flow Instantiation Execution. When the task flow parameters and all tasks under the task flow are constructed, the instantiation of the current task flow is finally completed and executed.

3.4 Overall Framework

The healthcare IoT data integration framework uses the workflow engine core Prefect Core to orchestrate and execute, calling each module component within the framework to perform specific tasks, which reduces the user's difficulty in the form of low code on the one hand, and improves the scalability of the framework on the other hand, facilitating the user to configure according to different task requirements to achieve the integration of complex business processes.

The designed framework is functionally divided into a parsing engine, data access module, data parsing module, data analysis module, data storage module, and model-as-a-service module and workflow management and maintenance of each functional module through the workflow as shown in Fig. 3.

3.5 MQTT-Based Remote Messaging Access

The Data Access module handles the dynamic update and execution of data subscription tasks, receives taskResource configuration path parameters, and reads the dynamic configuration of data subscription tasks through task flow timed polling.

Data subscription tasks subscribe to remote data via MQTT clients [18]. An MQTT client can only subscribe to a subscription topic under one MQTT server, so the framework regularly reads the dynamic configuration in taskResource and assembles MQTT clients according to the corresponding parameters to correspond to different servers or subscription topics with different parsing storage mechanisms under the same server [19]. At the same time through the configuration of the state identifier for subscribing clients to mention the destruction mechanism to achieve dynamic message subscription.

Example of Device Data Subscription Topic Client Description Snippet is shown in Fig. 4.

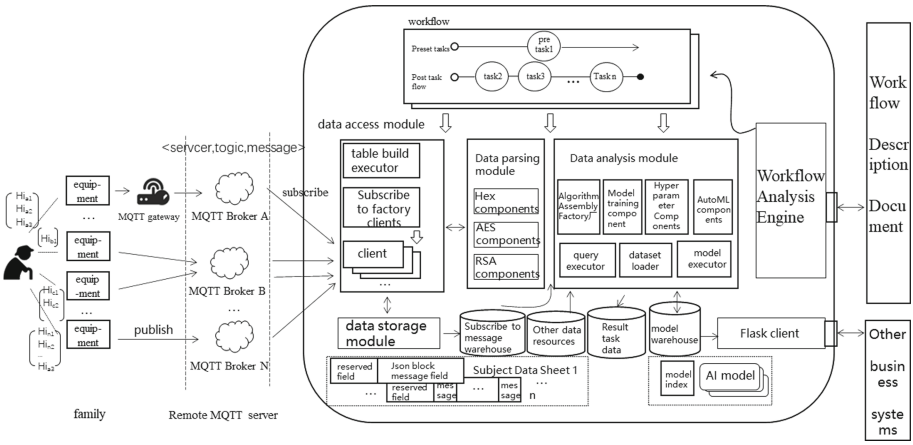

Fig. 3. IoT Data Integration Framework

The two attributes db and flag in the subject node are used to identify the table build status (0-not built, 1-built) and subscription status (0-not subscribed, 1-subscribed, 2-need to cancel, 3-canceled) of the subscribed topics, respectively.

```
<equipmentsub>
    <server>******</server>
    <port>****</port>
    <subject db = "1" flag = "1">oldTopic/device01</subject>
    <subject db = "1" flag = "1">oldTopic/device02</subject>
    <username>*****</username>
    <password>*****</password>
</equipmentsub>
<equipmentsub>
    <server>*****</server>
    <port>*****</port>
    <subject db = "0" flag = "0">newTopic/#</subject>
    <username></username>
    <password></password>
</equipmentsub>
```

Fig. 4. Example of automatic creation of description fragment for a data table of a device data subscription topic IoT Data Integration Framework

The data access module obtains the corresponding server IP address, port number, and the list of subscribed topics of all subject nodes under the node by traversing all equipmentsub nodes. Based on the db identification status, the table builder is started, and the flag identification status starts the subscription client assembly factory or closes the client instance. Finally, we update the corresponding db and flag identifiers and taskResource configuration to achieve the dynamic update of data subscription tasks.

Build Table Actuator. The subscribed message data comes from different devices, and the parsed message bodies vary widely. Message body fields and data meaning of the corresponding structure using sql, including complex JSON type of messages using nosql for storage. The framework uses postgresal, using the characteristics of postgresql

to maximize the compatibility of message bodies in the form of sql and nosql, avoiding the use of multiple database storage methods to increase the complexity of the system [20].

In order to prevent data loss problems, it is necessary to create the corresponding subscription message data table in advance when the data is first accessed to achieve data persistence. An example of an automated data table creation description fragment for the device data subscription topic is shown in Fig. 5.

```
<equipmentdb>
    <sub>oldTopic/device01</sub>
    <type>jsonField</type>
    <table>device01</table>
</equipmentdb>
<equipmentdb>
    <sub>oldTopic/device02</sub>
    <Type>customField</type>
    <table>device02</table>
    <field type = "VARCHAR(255)">humidity</field>
    <field type = "VARCHAR(255)">content_co</field>
</equipmentdb>
```

Fig. 5. Example of automated data table creation description fragment for device data subscription topic

When executing data subscription stream tasks, the framework regularly reads the dynamic configuration in taskResource. The table executor performs two types of operations through the type tag value: when the type is customField, the above description fragment is first parsed as a table CREATE statement and takes the table tag value as the table name, the default necessary fields include id field incremental primary key and createtime field to indicate the current record insertion time; when the type of attribute "type" is When the type is JSON field, the ALTER statement is executed after the completion of the table build operation by adding a JSON type data field for storing complex message bodies.

Subscription Client Factory. The subscription client factory receives parameters such as the IP address of the subscription topic server and completes the instantiation of the client based on the python version of the MQTT client paho library. The factory first writes the subscription topic and the current process u number to the database, which is used by subsequent modules to perform dynamic additions and deletions to the client instance, and then instantiates the data parsing module and the data saving module. In the callback function on_connect, it subscribes to the corresponding topic, and in on_message, it calls the parsing and saving instances. Finally, the factory instantiates the client, assigns the callback function to the client instance and starts the MQTT client connection.

3.6 Data Parsing and Preservation Based on Pass-Through and Encryption Components

This framework receives data from users via MQTT messages, and IOT detection devices connected to the server may be located in an insecure network environment where

messages may be listened to and tampered with. Some IoT detection device providers safeguard the confidentiality, integrity, and authenticity of the data by encrypting the MQTT messages. Therefore, a parsing module is needed to decrypt specific device data. The framework builds several parsing components to adapt to various third-party encryption parsing according to the actual business situation, and the data decoding can be performed for the accessed data by adding the corresponding description in the workflow. The network layer where different devices are located is complex, so the control of data security is mainly carried out in the application and transport layers of the MQTT protocol, where data security can be protected by adding MQTT user names and passwords to the workflow.

Hexadecimal Pass-Through Parsing Component. Most of the communication methods of up and down computers, such as serial and CAN ports, and many Bluetooth modules, are based directly on hexadecimal hardware commands transmitted in the form of pass-through. Therefore, for devices that transmit data in hexadecimal format, we need to provide a set of hexadecimal parsing components to convert the hexadecimal strings in transmissive form into the commonly understood JSON format. This component is specified to take two parameters, the first being a hexadecimal string converted to utf¬8, and the second specifying the name of the field to be parsed and the number of bytes in the original message for a JSON format body.

Table 3. Body temperature sticker hexadecimal message format

Name	Number of Bytes	Meaning
Id	4	Request sequence number
Temperature	2	Temperature property value
Voltage	2	Voltage property value

The hexadecimal transmission format of a temperature sticker is shown in Table 3. For example, the data of a temperature sticker is 0x0022334400260003, according to the format description, the first four bytes 0x00223344 represent the id number 2241348, the next two bytes 0x0026 represent the temperature of 38 degrees, and the last two bytes 0x0003 represent the current voltage of 3 V. In the workflow configuration In the workflow configuration, the workflow engine automatically calls this parsing component after parsing the subscription into the parsing format of { “id”:4, “temperature”:2, “voltage”:2}. After parsing, this parsing component will be called automatically, and the message to be parsed and the parsing rules will be passed into the component, and finally the component will parse the message body { “id”:2241348, “temperature”:38, “voltage”:3} is returned to the subsequent workflow, so the parsing of hexadecimal format messages can be completed.

3.7 Machine Learning Automated Data Analytics

The main function of the data analysis module is to provide support for the health aging data analysis task and to satisfy the user to train and utilize the model using the workflow. The machine learning algorithm library module in this framework defaults to the user already having data suitable for model training, so the module starts from the consideration of the process after feature engineering and designs a series of components to meet the needs of tasks such as data loading, model and parameter selection, model training, and model saving.

Data Loader. The module abstracts the data loader DataExtract component, the function of the data loader is mainly to load the data and return the data needed by the algorithm to learn the classification, and try to be compatible with more file formats and data description methods. The instance method of the DataLoader component class receives the dataset file path parameters and target column parameters passed by the workflow. The component determines the file type by parsing the suffix string of the dataset file path, and then calls the appropriate method to read the file.

Algorithm Assembly Factory. The data analysis module provides ModelueSet algorithm assembly factory, through the reflection mechanism based on python to achieve dynamic loading of user-specified algorithm modules and classes, while the dynamic assembly of algorithm modules and classes also provides a strong ability to extend the framework to use algorithm libraries other than scikit-learn library. First, the instance method of ModelueSet class component receives two string real parameters for the module name and class name passed by the workflow, and uses the importlib() function in the python standard library to dynamically import the required algorithm module by string, then calls hasattr() and getattr() functions to get the specific class name under the module, thus The algorithm classes selected by the user are loaded and assembled.

Model Training Component. After the training data and the specified algorithm module are dynamically loaded, the model needs to be trained. The training module receives the training data, assemblies, and the algorithms, hyperparameters, and model save names passed through the workflow.

Hyperparametric Components. For hyperparametric search problems, grid search and stochastic search can generally be used to solve them. The stochastic search method trades a small reduction in the efficiency of the low-dimensional space for a large increase in the efficiency of the high-dimensional search space, thus avoiding violent optimization search for a large number of parameters. Using the stochastic search method, the Ml_RSCV hyperparametric component provides its concrete implementation.

AutoML Component. If the user of the framework is very little knowledge of machine learning, it is still difficult to operate, so the framework combined with the recent rise of AutoML technology, the introduction of an automatic machine learning library ATM, and its packaging as AutomlTrain components. ATM all the algorithms are based on scikit-learn while supporting the popular classification algorithms, so can be well integrated with this framework. By combining with the workflow engine, the framework can be used to make calls to ATM through user-defined workflows.

```
<task taskType = "AutoMLTrain" taskName = "step2">
    <parm parmName = "url">D:\heart_disease\heart.csv</parm>
    <parm parmName = "joblib">heart_aTM1.pkl</parm>
</task>
```

Fig. 6. Example of workflow

When the user gives the following workflow as shown in Fig. 6.

The instance method of the AutoML component class receives the file path and target column name parameters and the model save path, then calls ATM and passes in the relevant parameters, completes the model and hyperparameter search, outputs the best model and parameters, and saves the model to the specified path.

4 Experiments and Results

4.1 Experiment Preparation

This section utilizes the proposed framework for validation based on the above requirements proposed in the healthy aging scenario. The experimental environment consists of a pilot community service center and a separate unit room, with the service center and the unit room each connected to the network through fiber optics. A server is deployed in the community service center with a hardware configuration of CPU Intel Xeon 5220 memory 32G and operating system of 64-bit windows server system.

4.2 Cases Based on Behavioral Anomaly Detection Scenarios

The proposed framework is validated to give a case study based on an abnormal behavior detection scenario. Activity detection sensors are installed in the home environment of the elderly, activity data such as activity points and durations of the elderly are recorded, and the labeled data are classified by labeling the normal and abnormal activity data of the elderly over a period of time, the labeled data are learned, and after the training is completed, the saved model is invoked to detect the abnormal data of the elderly user for the latest period of time according to the requirements of the elderly user.

The body detection sensor connects to its own gateway via the Zigbee protocol and forwards MQTT messages outwards. The MQTT message body of the device itself is: {“battery”: “100”, “mac”: “30:ae: 7b:e2:e5:53”, “value”: “0”,…}.

To add the function of recording the start time and duration of the person’s activity, the nodered project based on js was used to logically determine and forward the messages of the device, adding the number of hours the person’s activity started and the number of minutes the activity lasted. The new MQTT message body is {“start_time”: “12”, “duration”: “15”, “mac”: “30:ae:7b:e2:e5:53”, “create_date”: “2022-05-01”}.

Data Subscription Task Flow. The user first defines a workflow that is executed at regular intervals and contains a data subscription task flow. The data subscription task dynamically subscribes to the device data by reading the corresponding resources from the taskResource dynamic resource configuration. Since the device data does not require

special parsing, there is no need to define the parsing part and then start the data subscription task flow to complete the data access. The original message structure of the device is shown below. The dynamic resource configuration of the human activity detector is shown in Fig. 7.

```
<IotWF>
    <taskResourse resourceName = "RN_0002">
        <equipmentsub>
            <server>101.43.***.**</server>
            <port>18**</port>
            <subject db = "0" flag = "0">t</subject>
            <username></username>
            <password></password>
        </equipmentsub>
        <equipmentdb>
            <sub>t</sub>
            <type>customField</type>
            <table>pir</table>
            <field type = "VARCHAR(255)">start_time</filed>
            <field type = "VARCHAR(255)">duration</filed>
            <field type = "VARCHAR(255)">create_date</filed>
        </equipmentdb>
    </taskResourse>
</IotWF>
```

Fig. 7. Dynamic resource allocation of human activity detection sensors

```
<IotWF>
    <TaskFlow taskFlowName = "FN_003" TaskFlowType = "AnalysisFlow" scheduleType = "Once" scheduleValue = "0">
        <task taskName = "t1" taskType = "SQLExtract">
            <parm parmName = "SQL">D:\ecgPytorch\heart_disease\pir.csv</parm>
            <parm parmName = "targetCol">3</parm>
        </task>
        <task taskName = "t2" taskType = "ModuleSet">
            <parm parmName = "module">sklearn.neighbors</parm>
            <parm parmName = "class">KNeighborsClassifier</parm>
        </task>
        <task taskName = "t3" taskType = "M1_TRAIN">
            <parm parmName = "alg">KNeighborsClassifier</parm>
            <parm parmName = "p">{n_neighbors = 5, weights = 'distance'}</parm>
            <parm parmName = "joblib">knn01.joblib</parm>
        </task>
        <task taskName = "t4" taskType = "SQL">
            <parm parmName = "SQL">SELECT start_time, duration FROM pir where create_date > '2022-05-05'</parm>
            <parm parmName = "outgoing">t5</parm>
        </task>
        <task taskName = "t5" taskType = "M1Predict">
            <parm parmName = "joblib">knn01.joblib</parm>
        </task>
    </taskFlow>
</IotWF>
```

Fig. 8. Data Analysis Task Definition in Behavior Anomaly Detection Scenario

Data Analysis Task Flow. After visualizing the elderly behavior data, we can find that the normal activity time of the elderly is mainly concentrated around 13:00 to 16:00, and the activity length is mainly concentrated around 10 to 20 min, and there are four abnormal activities around 0:00 to 5:00, which may be related to the elderly's health condition and need to find medicine urgently. The data were therefore pre-labeled and formed into a dataset before training. As shown in Fig. 8, this data analysis task flow first performed the data loading task in lines 3–6 and loaded this dataset. Lines 7–10 define an algorithm assembly task calling the algorithm assembler to dynamically assemble the sklearn.neighbors module and classes. Lines 11–16 define a machine learning training

task that uses manual determination of model parameters and saves the generated KNN machine learning model. Lines 17–20 define a data query task that selects data in the pir table after May 5. Finally, a model execution task is defined to call the specified model from the model repository to detect the above data and return the model results.

5 Conclusion and Future Work

This paper provides an integration framework that presents an overall solution for complex healthcare devices integration, data storage, and machine learning analysis. It improves the scalability of the framework through flexible configuration, improves ease of use, and realizes "model-as-a-service" in healthcare scenarios. Based on the current work, this framework can further integrate technologies such as streaming data analysis, deep learning automation, etc. In the future to perform more complex tasks in different scenarios such as real-time data analysis for healthy aging and medical diagnosis.

Acknowledgment. This work is supported by the National Natural Science Foundation of China (61832014,61972276,62032016), and the Foundation of Jiangxi Educational Committee (GJJ210338).

References

1. Mutlag, A.A., Abd Ghani, M.K., Arunkumar, N.A., Mohammed, M.A., Mohd, O.: Enabling technologies for fog computing in healthcare IoT systems. Futur. Gener. Comput. Syst. **90**, 62–78 (2019)
2. Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y.: Performance evaluation of MQTT and CoAP via a common middleware. In: 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), April, pp. 1–6. IEEE (2014)
3. Sworna, N.S., Islam, A.M., Shatabda, S., Islam, S.: Towards development of IoT-ML driven healthcare systems: a survey. J. Netw. Comput. Appl. **196**, 103244 (2021)
4. Kulkarni, A., Sathe, S.: Healthcare applications of the internet of things: a review. Int. J. Comput. Sci. Inf. Technol. **5**(5), 6229–6232 (2014)
5. Santos, J.: E-service quality: a model of virtual service quality dimensions. Managing Serv. Qual.: an Int. J (2003)
6. Laplante, P.A., Laplante, N.L.: A structured approach for describing healthcare applications for the internet of things. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 621–625. IEEE (2015) December
7. Alfian, G., Syafrudin, M., Ijaz, M.F., Syaekhoni, M.A., Fitriyani, N.L., Rhee, J.: A personalized healthcare monitoring system for diabetic patients by utilizing BLE-based sensors and real-time data processing. Sensors **18**(7), 2183 (2018)
8. Yang, Z., Zhou, Q., Lei, L., Zheng, K., Xiang, W.: An IoT-cloud based wearable ECG monitoring system for smart healthcare. J. Med. Syst. **40**(12), 1–11 (2016)
9. Laport, F., Dapena, A., Castro, P.M., Vazquez-Araujo, F.J., Iglesia, D.: A prototype of EEG system for IoT. Int. J. Neural Syst. **30**(07), 2050018 (2020)
10. Yacchirema, D., de Puga, J.S., Palau, C., Esteve, M.: Fall detection system for elderly people using IoT and big data. Procedia Comput. Sci. **130**, 603–610 (2018)

11. Kadarina, T.M., Priambodo, R.: Monitoring heart rate and SpO2 using thingsboard IoT platform for mother and child preventive healthcare. In: IOP Conference Series: Materials Science And Engineering, vol. 453, No. 1, p. 012028. IOP Publishing (2018) November
12. Otto, C., Milenković, A., Sanders, C., Jovanov, E.: System architecture of a wireless body area sensor network for ubiquitous health monitoring. J. Mob. Multimedia **1**, 307–326 (2006)
13. Xiao, F., Zhang, W.H., Wang, D.H.: Overview of workflow technology in scientific process. Appl. Res. Comput. **28**(11), 4013–4019 (2011)
14. Hollingsworth, D., Hampshire, U.K.: Workflow management coalition: the workflow reference model. Document Number TC00-1003 19(16), 224 (1995)
15. Light, R.A.: Mosquitto: server and client implementation of the MQTT protocol. J. Open Source Softw. **2**(13), 265 (2017)
16. Russel, S.J., Norvig, P.: Artificial Intelligence–A Modern Approach, pp. 736–741. Person Education Inc., New Jersey (2003)
17. McCracken, D.D., Reilly, E.D.: Backus-naur form (bnf). In: Encyclopedia of Computer Science, pp. 129–131 (2003)
18. Mishra, B., Kertesz, A.: The use of MQTT in M2M and IoT systems: a survey. IEEE Access **8**, 201071–201086 (2020)
19. Babu, B.S., Ramanjaneyulu, T., Narayana, I.L., Srikanth, K., Sindhu, D.H.: Smart vehicle management through IoT. Int. J. Emerg. Trends Technol. Comput. Sci. (IJETTCS) **5**(3), 26–31 (2016)
20. Young, M.: An automated framework to derive model variables from open transport data using R, PostgreSQL and OpenTripPlanner (2016)