IET Networks

Research Article

IET Journals
The Institution of Engineering and Technology

AItalk: a tutorial to implement AI as IoT devices

ISSN 2047-4954
Received on 8th October 2018
Revised 8th November 2018
Accepted on 7th December 2018
E-First on 7th December 2018
doi: 10.1049/iet-net.2018.5182
www.ietdl.org

Yun-Wei Lin[1] ✉, *Yi-Bing Lin*[1], *Chun-You Liu*[1,2]

[1]*Department of Computer Science, National Chiao Tung University, Hsinchu 30010, Taiwan*
[2]*Computational Intelligence Technology Centre, Industrial Technology Research Institute, Hsinchu 31057, Taiwan*
✉ *E-mail: jyneda@gmail.com*

Abstract: In one of the recent trends of Internet of Things (IoT), the IoT data are manipulated by Artificial Intelligence (AI) techniques for smart applications. By including AI into existing IoT application programs, significant coding effort is required. This paper proposes a solution called AItalk to resolve this issue. Unlike traditional AI-based IoT applications that tightly integrate the AI mechanism within the network applications, the novel idea of AItalk is to treat the machine learning mechanism as a cyber IoT device. Our solution allows seamless inclusion of machine learning capability to the existing IoT applications without any programming effort. The advantage of this approach is that we can decompose a complex AI application into simplified distributed modules connected by using the IoT technology, and therefore the AI solution can be built more effectively. Also, in our approach, data can be easily processed in real time for an AI application. Supervised machine learning naturally fits the IoT applications, where the sensors provide the features to the AI algorithms, and the remote controllers serve as the labels. We show an example that the overhead of the IoT communication in AItalk is less than 30 ms and the AI prediction time is less than 2 ms.

1 Introduction

Internet of Things (IoT) connects devices to each other through the Internet, which is considered as the 'next big thing'. Many useful data obtained from the IoT devices are sent to the network for processing to provide, for example, money flow, logistics flow, people flow, interactive art design, and so on. Recently, the IoT data are manipulated by Artificial Intelligence (AI) techniques for smart applications [1–5]. By integrating AI into existing IoT applications, significant programming effort is required. Many open-source AI tools are available (see Table 1), which can be freely downloaded from the Internet [6]. However, it is tedious to use the AI tools to build IoT applications. To resolve this issue, the authors propose AItalk that allows a developer to easily add the AI mechanism to the existing IoT applications through a graphical user interface (GUI). AItalk aims to provide a user-friendly tool to easily include AI into IoT applications without programming efforts.

The proposed framework offers several features such as regression models, automatic data collection, user-friendly GUI, real-time prediction, model training, highly scalable architecture, and automatic hyper-parameter tuning. The authors integrate the above framework with an IoT platform called IoTtalk [7–9] to rapidly develop and expand the smart IoT applications.

AItalk addresses several issues including how to connect the sensors, how to control the actuators, how to use the machine learning algorithm, how to train the models, how to validate accuracy, precision and recall of the model, and how to adjust the setting. Besides validation, the authors also consider the real-time execution complexity of the system.

Fig. 1 illustrates a simplified functional block diagram of IoT-based machine-learning mechanism for a smart home application, which is implemented by the IoT application designer. In this example, the house owner (the user) originally holds a remote controller (Fig. 1(1)) to control an actuator (Fig. 1(2)), for example, a fan. To automatically control the fan, several sensors (e.g. for temperature, humidity and so on; see Fig. 1(3)) produce the sensor data in real time.

These data are first sent to the feature extraction module (Fig. 1(4)) of machine learning. In this module, the designer can select specific extraction methods (to be discussed at Steps 2–6 in Section 2.2) to extract the desired characteristics of the sensor data

Table 1 Open source AI tools

Type	Library name	Commits
core libraries	Numpy	15,980
	SciPy	17,213
	Pandas	15,089
visualisation	Matplotlib	21,754
	Seaborn	1699
	Bokeh	15,724
	Plotly	2486
machine learning	SciKit-Learn	21,793
	Keras	3519
	TensorFlow	16,785
	Theano	25,870
NLP	NLTK	12,449
	Gensim	2878

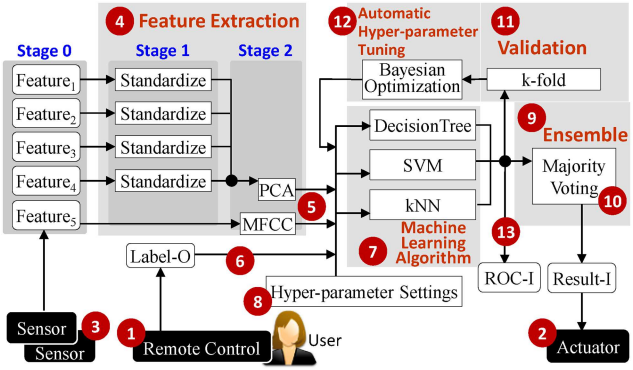

Fig. 1 *Machine learning for smart home application*

to form a feature vector (Fig. 1(5)). To train a machine-learning model, both the feature vectors and the user's control signals (i.e. the labels; see Fig. 1(6)) are sent to the machine-learning algorithm module (Fig. 1(7)). This module allows the designer to select appropriate algorithms. The designer then sets the hyper-parameters for the algorithms (Fig. 1(8)) to conduct the model training process, which will generate a prediction result for each of the algorithms. These results are sent to the ensemble module (Fig. 1(9)). By selecting an appropriate ensemble method in this module, the authors anticipate to produce the best prediction result (Fig. 1(10)) among those machine-learning algorithms. To enhance the performance of the AI mechanism, the results are also used to conduct validation (Fig. 1(11)) to tune the hyper-parameters (Fig. 1(12)) and then fed back to the machine-learning algorithm. The authors will show how the mechanism in Fig. 1 is implemented in AItalk. Useful statistics of machine learning are also produced (Fig. 1(13)) to adjust the models. The details will be elaborated later.

The main contribution here is to show how to nicely integrate an IoT platform with the existing state-of-the-art AI models. Specifically, the authors subtly implement the AI mechanism as a cyber IoT device to support seamless integration of AI and IoT. Such integration has not been found in the literature. To authors' understanding, no existing solutions can automatically integrate IoT with AI. The interaction between IoT and AI was done offline [10, 11] or required intensive re-programming in the existing IoT platforms [12, 13].

The paper is organised as follows. Section 2 describes how AItalk can be used to provide machine-learning mechanism in an IoT application. Section3 shows how the IoT mechanism can be used to build an AI application for human occupancy prediction and discusses the time complexity of AItalk. Section4 summarises our findings.

2 Building a machine learning based IoT application

This section shows how the machine-learning mechanism can be easily included in an existing IoT application developed in AItalk. Our approach (Fig. 2(a)) integrates an IoT device management platform called IoTtalk (Fig. 2(b)) [7–9] with the Python open-source tool called scikit-learn [14]. In IoTtalk, every IoT device is installed with a software module called device application (DA; see Fig. 2(c)–(e)). The DA is responsible for communication with the IoTtalk engine (Fig. 2(b)).

The AItalk server consists of two components: the GUI for machine learning (ML-GUI; Fig. 2(f)) and the IoTtalk engine. The IoTtalk engine provides MQTT- and HTTP-based RESTful application programming interfaces (APIs) for a DA to deliver/retrieve the IoT information to be manipulated by the IoTtalk engine. ML-GUI provides a friendly web-based user interface to quickly set up the hyper-parameters for the machine-learning mechanism. The GUI then establishes the connections and meaningful interactions among the sensors and the actuators through machine learning.

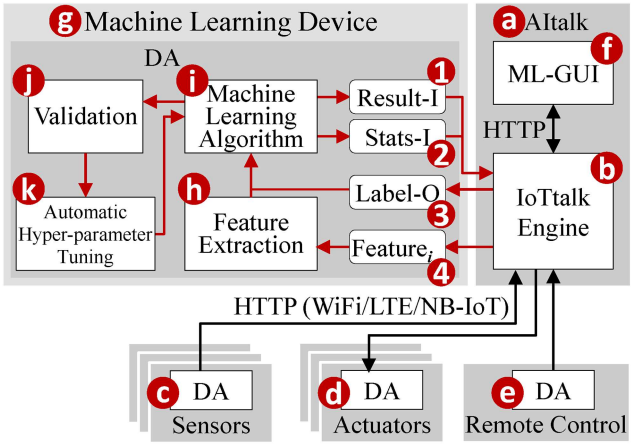

Fig. 2 *AItalk architecture*

2.1 Provisioning machine learning to interact with sensors and actuators

In a typical AI-based IoT application, the IoT devices are connected to a network application that executes the AI algorithms. AItalk allows an application designer to conveniently configure the solution for an AI problem through ML-GUI. In this GUI, the designer instructs the IoTtalk engine to automatically generate network applications to execute desired tasks that create or set up device features, functions, and connection configurations.

In our approach, the machine-learning mechanism in Fig. 1 is considered as a 'cyber' IoT device (Fig. 2(g)) managed by IoTtalk. The DA of this machine-learning device (ML_device) consists of four components (Fig. 2(h)–(k)) that implements the machine-learning mechanism in Fig. 1(4)–(13).

In AItalk, every device is defined as a set of input and output 'device features' (DFs). An input DF (IDF) can be a sensor (such as a magnetometer) or a controller (such as a button). An output DF (ODF) is an actuator (such as a motor). In ML_device, the prediction produced by the machine-learning algorithm (Fig. 2(i)) is sent from its IDF called Result-I (Fig. 2(1)) to control the actuators (Fig. 2(d)) through the IoTtalk engine. ML_device also produces statistics (Stats-I; see Fig. 2(2)) that are typically illustrated in a display actuator for the designer to tune the hyper-parameters of the machine-learning model.

On the other hand, the Label-O ODF (Fig. 2(3)) receives the label from the remote control (Fig. 2(e); see also Fig. 1(6)), and the Feature$_i$ ODFs of ML_device (Fig. 2(4)) receive the data from the sensors (Fig. 2(c)) through the IoTtalk engine. Therefore, every IoT device consists of two parts: the input device (the set of IDFs) and/or the output device (the set of ODFs).

In ML-GUI, the IoT devices and the DFs are illustrated as icons. The icons on the left-hand side are called input devices (e.g. Google Home in Fig. 3(1)) and the icons on the right-hand side are called the output devices (Fan in Fig. 3(2)). Inside a device icon, there are one or more small icons that represent the DFs (e.g. Microphone in Fig. 3(3) and Switch-O in Fig. 3(4)). Note that for the readability purpose, if an IDF and an ODF have the same name, ML-GUI will append the IDF name with '-I' and the ODF name with '-O'.

In IoTtalk, every network application is mapped to an icon in the GUI. Therefore, it provides a simple way to connect the IoT devices to the network application, which is inherited by AItalk. For example, the authors have integrated ArduinoYun [15] with IoTtalk, and develop an automatic procedure to accommodate any sensors installed in the input pins of ArduinoYun [9]. Similarly, the actuators to be controlled can be installed in the output pins of ArduinoYun or through IoTtalk-RC [8] that provides infrared remote control to aftermarket home appliances. To connect a sensor to an actuator, the authors only need to drag a line between the corresponding IDF icon and the ODF icon (Join 1 in Fig. 3(5)), and IoTtalk automatically creates the network application to handle the interaction between them.

Fig. 3 *Fan control in AItalk GUI without machine learning*
(a) A fan controlled by Google Home, ***(b)*** Connection in AItalk GUI

Unlike traditional AI-based IoT applications that tightly integrate the AI mechanism within the network applications, the novel idea of AItalk is to treat the machine-learning mechanism as a cyber IoT device called ML_device. Our solution allows seamless inclusion of machine-learning capability to the existing IoT applications without any programming effort. Consider the IoT-based Fan control application in Fig. 3*a*. The AItalk configuration of this application without machine learning is shown in Fig. 3*b*. Suppose that the authors want to use the indoor and outdoor humidity and the temperature sensors to control the fan through the machine-learning mechanism. To do so, the authors extend the application in Fig. 3*b* by adding the indoor and the outdoor sensors. The input sensors are grouped in an input device called IndrSnsr (abbreviated for Indoor Sensors; see Fig. 4(1)) and the outdoor sensors are grouped in the OutdrSnsr device (abbreviated for Outdoor Sensors; see Fig. 4(2)). Through the voice instructions sent from Google Home (Fig. 4(3)), the voice characteristics are also considered as a feature, and the AI mechanism uses this feature to distinguish the instructions come from different users in the room (Join 5). To include AI in this application, the authors first create a cyber ML_device called SmartFan that consists of an output device (Fig. 4(4)) and an input device (Fig. 4(5)). The output device has five Feature ODFs (Fig. 4(6)) and one ODF for labelling (i.e. Label-O; see Fig. 4(7)). The input device has two IDFs called Result-I and ROC (Receiver Operating Characteristic curve). The indoor temperature and the humidity sensors (Fig. 4(1)) are connected to SmartFan through Joins 1 and 2. The outdoor temperature and the humidity sensors (Fig. 4(2)) are connected to SmartFan through Joins 3 and 4. Besides producing the labels to Label-O through Join 6, the Microphone IDF of Google Home is also connected to SmartFan through the Feature5 ODF. Based on the voice characteristics, SmartFan can make prediction for different users in the room. By connecting Result-I to Switch-O through Join 7, SmartFan gives instruction to operate the fan intelligently. The above example shows that AItalk provides a transparent way to reconfigure the existing IoT application in Fig. 3*b* into the AI-based application in Fig. 4. Such task can be completed in <10 min (excluding the ML device setup to be elaborated in the next subsection).

AItalk GUI provides the 'Model' pulldown list (Fig. 5(1)) to select the device icons. By selecting the ML_device item in this list (Fig. 5(2)), the GUI window pops up the device feature setup

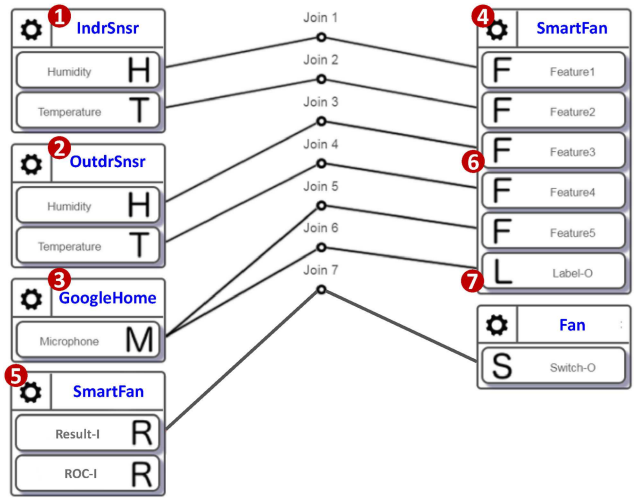

Fig. 4 *AItalk GUI setup for the ML device of fan control*

Fig. 5 *Creating the ML_device icon*

window for ML_device (Fig. 5(3)). The designer then selects two IDFs, that is, Result-I and ROC-I (Fig. 5(4)), 5 Feature ODFs and one ODF called Label-O (Fig. 5(5)). When the 'Save' button (Fig. 5(6)) is clicked, the input and the output ML_device icons (Fig. 4(4) and (5)) automatically show up in the GUI window. At this point, the DA of ML_device are created with the default set-ups for SmartFan (Fig. 2(h)–(k)). Other device icons (Fig. 4(1)–(3)) can be created following the same procedure. A nice feature of AItalk is that after an AI expert has configured an AI application, the ML_device can be saved, where the settings in the ML_device become the default values. A non-AI expert can reuse the ML_device to build new AI applications with the default set-ups.

2.2 Configuring ML_device

If the default set-up of ML_device does not fit the need of an application, the four components of ML_device can be conveniently reconfigured as follows. Suppose that the authors want to configure the SmartFan with the machine-learning set-up illustrated in Fig. 1, where the remote control (Fig. 1(1)) is GoogleHome (Fig. 3(1)) and the output device (Fig. 1(2)) is Fan (Fig. 3(2)). The authors first click the ä icon at the top left corner of the ML_device icon (Fig. 4(4)) to pop up the ML set-up window (Fig. 6). In the ML set-up window, the authors first give the name of the device, that is, SmartFan (Fig. 6(1)). Then, the authors set up the configuration in Fig. 1 with the following steps.

Step 2–6. Feature Extraction: In Fig. 1(4), two-stage feature extraction is used, which is set up in AItalk by selecting the stage number '2' in the ML set up window (Fig. 6(2)). Five IDFs created in (Fig. 5(5)) are treated as Stage 0, which are the inputs of Stage 1 (Fig. 6(3)). Stage 1 is a 5×5 matrix where the diagonal elements are checked to represent that Feature i of Stage 0 is connected to Feature i of Stage 1 (where $1 \leq i \leq 5$). In this matrix, Features 1 and 2 are indoor temperature and humidity. Features 3 and 4 are outdoor temperature and humidity. Since the scales of different sensors are not the same, the data for these sensors must be standardised (Fig. 6(4)). Feature 5 is a microphone, and the voice data are bypassed (not processed at Stage 1). At Stage 2, two

extraction algorithms are used (Fig. 6(5)). The characteristics for the standardised data of the first four features are extracted by Principal Component Analysis (PCA). For Feature 5, Mel-scale Frequency Cepstral Coefficients (MFCC) is used to extract the voice characteristics (Fig. 6(6)). Therefore, Stage 2 is a 2×5 matrix to represent the mappings between the outputs of Stage 1 to the inputs of Stage 2. Currently, AItalk implements all feature extraction algorithms in [14] including Bypass, Scaling, PCA, FFT, MFCC etc.

Step 7. Algorithm Selection: The machine-learning algorithm module (Fig. 6(7)) classifies the algorithms into three categories: classification, regression, and user defined. The classification algorithms include Support Vector Machine (SVM), kNN, Decision tree, Random Forest, and so on, which can be selected and configured as a multi-stage network of algorithms just like the feature extraction module. In the SmartFan example, the SVM, the kNN, and the Decision Tree algorithms are selected for model training (Fig. 1(7)). This configuration is set up by selecting one stage with five inputs. At Stage 1, three algorithms are selected (Fig. 6(7)), and a 3×5 matrix is used to represent this configuration.

Step 8. Ensemble: If multiple machine-learning algorithms are selected at Step 7, then the ensemble module needs to be set up to find the best result based on the predictions produced by these algorithms. The ensemble technique [1, 5, 16] uses multiple learning algorithms with variability to achieve better prediction result as compared to any of the individual algorithms. In the SmartFan example, the authors select 'Majority Voting' in the ML set up window (Fig. 6(8)). This ensemble method outputs the results produced by most of the algorithms.

Step 9. Validation: In Fig. 6(9), the k-fold method is used for cross-validation. Other validation methods such as Leave One Out, Leave P Out, and Stratified k-fold are also implemented in AItalk.

Step 10. Automatic Hyper-parameter Tuning: In Fig. 6(10), the Bayesian optimisation method is used for automatic hyper-parameter tuning. This method follows a sequential model-based approach, which prescribes a prior belief over the possible objective functions and then sequentially refines the model as data are observed via Bayesian posterior updating. Other methods such as Grid Search and Random Search are also implemented in AItalk.

Step 11. Training Data Selection: By clicking the 'Training data' button in the ML set up window (Fig. 6(11)), a webpage pops up for the designer to select the real-time data or the historical data to train the model (Fig. 7(1)). The designer can further specify the start and the end dates (Fig. 7(2)). He/she can also specify the days in a week (Fig. 7(3)) and time in a day (Fig. 7(4)).

Step 12. Model Training: When the designer clicks the 'Train Model' button (Fig. 6(12)), SmartFan starts collecting data and training the model following the setup of Steps 2–11. The historical data or the real-time data are sent to SmartFan for execution. During the training process, SmartFan automatically tunes the hyper-parameters as follows. Through k-fold cross-validation, the authors obtain accuracy, precision, and recall for SVM, Decision Tree, and kNN based on the current hyper-parameters. Then Bayesian optimisation is used to adjust the hyper-parameters. AItalk automatically generates the ROC curves, which can be displayed through ROC-I of SmartFan (see Fig. 4(5)). This curve plots the true positive rate against the false positive rate for various hyper-parameters. In the test, any increase in sensitivity will be accompanied by a decrease in specificity, and the ROC curve shows the trade-off between sensitivity and specificity.

This section showed that supervised machine learning naturally fits the IoT applications, where the sensors provide the features to the AI algorithms, and the remote controllers serve as the labels. Since the authors implement AItalk as an IoT system, it inherits the properties of IoTtalk, including connectivity, availability, and scalability. Other IoT approaches such as Alljoyn [12] and OM2M [13] need to intensively re-programme the applications to accommodate the AI capability.

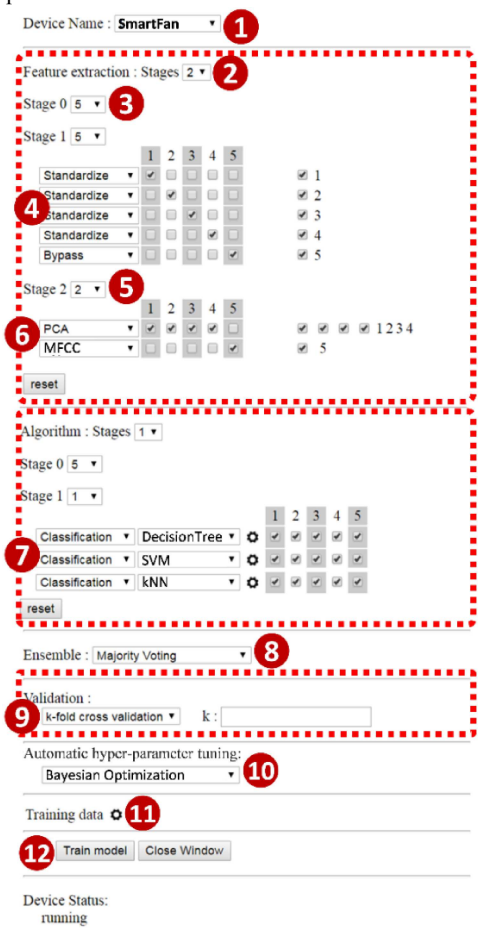

Fig. 6 *ML setup window for SmartFan*

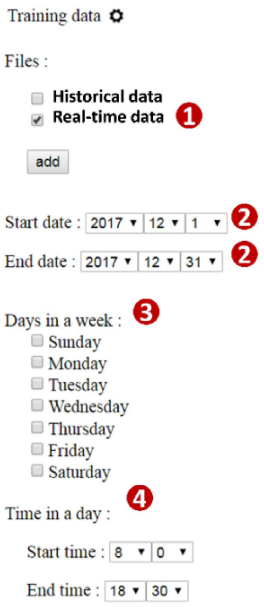

Fig. 7 *Selection of training data*

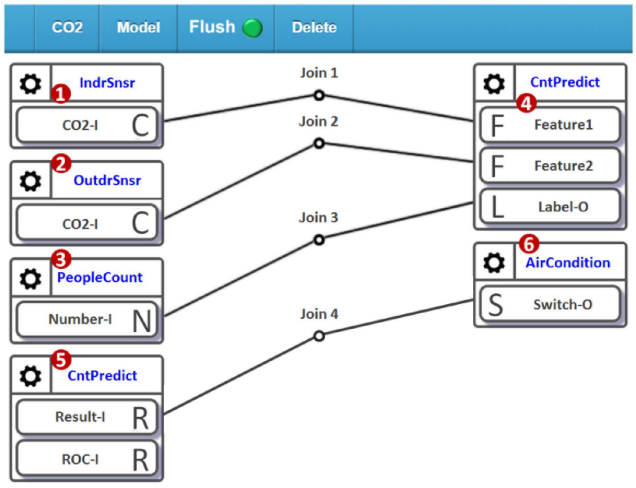

Fig. 8 *AItalk configuration for real-time estimation of human occupancy*

Fig. 9 *AItalk demo room in Room 311 in MIRC building, NCTU*

3 Altalk for indoor human occupancy counting

Typical building maintenance manages heating, ventilation, and air conditioning (HVAC) [17]. A building management system utilises human occupancy information for automate HVAC control. Such information is obtained from sensors in a closed space. Early studies (see [11] and the references therein) used simulation models to reduce energy consumption. These approaches may not reflect the actual environment and are not adaptable to different types of rooms and buildings. Some approaches utilised radio-based sensors such as infrared sensors to detect people entering/leaving the rooms. These radio-based approaches require extra sensor hardware installation. Other studies utilised carbon dioxide (CO_2) sensors to estimate the number of people (see [10] and the references therein). The advantage of the CO_2-based approaches is that CO_2 sensors are already installed with the ventilation equipment, and therefore, there is no extra hardware cost. For these approaches, the best results were obtained in [10], where the authors modelled various regression algorithms to predict human occupancy. A zero pattern adjustment model was proposed with additive decomposition to reconstruct the prediction value to increase the accuracy. The study showed prediction accuracy of 94.68%. These existing CO_2-based studies cannot dynamically predict the changes of people count. For example, in a theatre, [10] predicts the number of audience in the theatre, but does not predict when people enter or leave.

Based on AItalk, this section shows how to develop an automatic real-time HVAC control by utilising human occupancy prediction obtained from a CO_2 sensor in a closed space. The purpose of the prediction information is two folds. First, when the room is empty but the HVAC resources (e.g. for air conditioning, heating and so on) are still turned on, the prediction will suggest to turn off these resources. In National Chiao Tung University (NCTU), this mechanism is used by the room attendants to determine when the guest rooms are empty and to clean the rooms without interrupting the guests. Second, the prediction provides information to reallocate the room resources. For example, if there are too few people in the room in a certain period, then the drinking machine in that room is turned off (to save electricity) and AItalk guides people to use the drinking machines in other rooms.

Fig. 8 shows the human occupancy prediction system configured in the AItalk GUI. The configuration consists of the following devices. The IndrSnsr device (Fig. 8(1)) includes an indoor CO_2 sensor and the OutdrSnsr device (Fig. 8(2)) includes an outdoor CO_2 sensor. The PeopleCount device (Fig. 8(3)) provides accurate people count through real-time electronic sign-in and sign-out. In this example, ML_device is implemented as CntPredict with the input and the output devices. The output CntPredict device (Fig. 8(4)) has two features connected to the indoor and the outdoor CO_2 sensors through Joins 1 and 2. The Number-I IDF of PeopleCount is connected to the Label-O ODF of CntPredict through Join 3 to serve as the ground truth of the prediction, which is used for data training. The input CntPredict device (Fig. 8(5)) produces the results used to control the air conditioner (Fig. 8(6)). The indoor and the outdoor CO_2 sensors were deployed at an IoT demo room in NCTU since year 2017 (see Fig. 9).

3.1 Implementation of CntPredict

Design of CntPredict is illustrated in Fig. 10. Based on this design, CntPredict is implemented in AItalk through the ML set up window similar to those in Figs. 7 and 8.

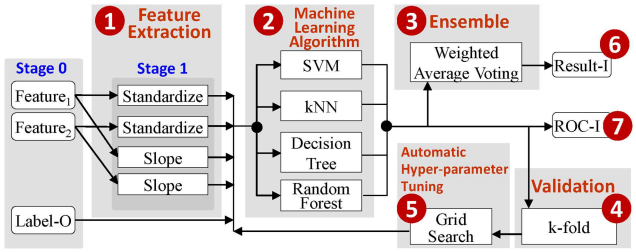

Fig. 10 *Design of CntPredict*

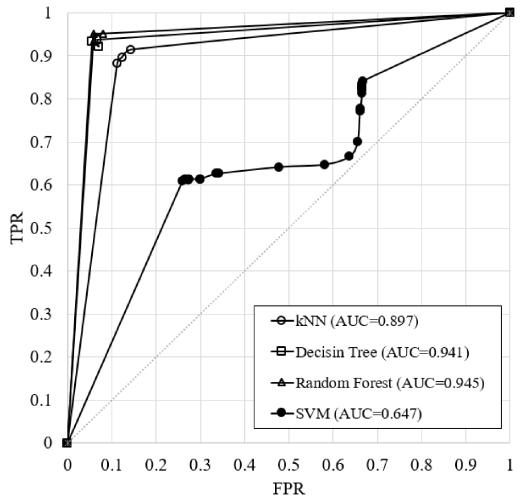

Fig. 11 *ROC curve*

Feature Extraction: In Fig. 10(1), one-stage feature extraction is used. Two IDFs (CO_2-Is in both IndrSnsr and OutdrSnsr) are the inputs, where the data for the indoor and the outdoor CO_2 sensors are standardised. In this module, the slope function is also used to compute the slope of the most recent five data items of each feature. This extraction function computes the rate of the CO_2 concentration change, which is assumed to be related to the change of people count.

Algorithm Selection: In the machine-learning algorithm module (Fig. 10(2)), SVM, kNN, Decision Tree, and Random Forest algorithms [2–5, 18, 19] are selected for model training. These algorithms are reported as the foremost classifiers that produce high accuracies [20].

Ensemble: The ensemble module (Fig. 10(3)) is set up to find the best result based on the predictions produced by all machine-learning algorithms. The authors select 'Weighted Average Voting' (WAV) that averages the predictions of the algorithms. The combined prediction is usually better than any of the individual predictions because its variance is reduced.

Validation: In Fig. 10(4), the k-fold method is used for cross-validation. This method randomly splits data into k parts (we set $k = 5$), and select one of these parts as testing data (20%), the remaining parts as training data (80%). After repeating k times, the authors can verify the average performance.

Automatic Hyper-parameter Tuning: Validation error is calculated as the effect of a set of hyper-parameters for each of the selected machine-learning algorithms. These errors are sent to the hyper-parameter optimiser (i.e. the automatic hyper-parameter tuning module; see Fig. 10(5)), where Grid Search is used to exhaustively search through a specified subset of the hyper-parameter space of the selected machine-learning algorithms. That is, based on the validation errors, better choices are searched within the hyper-parameter space to update the hyper-parameters. Model training continues until it reaches the stop condition of the hyper-parameter optimiser.

Training Data Selection: The authors have collected the data of the indoor and the outdoor CO_2 sensors at an IoT demo room in NCTU from April 2017 to September 2018. There are visitors for demo activities and student meetings, and the concentration of CO_2 is changed due to changing of the population in the room. The authors have logged the numbers of people in the room as the ground truth.

Model Training: When the designer clicks the 'Train Model' button (Fig. 6(12)), CntPredict starts collecting data and training the models following the setup of Steps 2–9. Through ROC-I of CntPredict (see Fig. 10(7)), AItalk automatically generates the ROC curves illustrated in Fig. 11. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various hyper-parameters. For example, TPR measures the probability that when the ground truth is 1 count, and CntPredict predicts 1 count. On the other hand, FPR measures the probability that when the ground truth is for example, NOT 1 count, CntPredict predicts 1 count. The best possible prediction method would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 100% TPR (no false negatives) and 0% FPR (no false positives). Therefore, the higher the AUC (Area Under the Curve) value, the better the model is, and the (0,1) point is called a perfect classification. A random guess would give a point along a diagonal line from the left bottom to the top right corners. The diagonal line divides the ROC space. The points above the diagonal line represent good classification results (better than random); the points below the line represent bad results (worse than random). Fig. 11 shows that Random Forest has best performance where the AUC value is 0.945. The AUC values of Decision Tree, kNN, and SVM are 0.941, 0.897, and 0.647, respectively. The ROC curve indicates that SVM is just a little bit better than random guessing.

3.2 Accuracy of CntPredict

We consider two scenarios to investigate the accuracy of CntPredict. Scenario 1 uses indoor sensor only. Scenario 2 uses both indoor and outdoor sensors. The authors also consider two cases for WAV ensemble. WAV1 involves four algorithms. WAV2 considers three algorithms by excluding SVM. Table 2 shows that, in Scenario 2, 95.7% accuracy is achieved for both WAV1 and WAV2. Therefore, the ROC curves suggest removing SVM to reduce CntPredict's computation cost. The execution time of the feature extraction stage is 14.4 ms in Scenario 2. In the algorithm stage, each algorithm searches 15 hyper-parameter combinations, and the expected execution times are 931.5 ms for SVM, 66.3 ms for kNN, 21.4 ms for Decision Tree, and 482.5 ms for Random Forest. The training times for WAV1 and WAV2 are 22,539.8 ms and 8567.0 ms, respectively. Therefore, WAV2 reduces 62.0% training overhead as compared with WAV1.

In Scenario 1, the prediction results are affected by the natural oscillation of the CO_2 concentration. When the CO_2 concentration is less than 450 ppm, the algorithms cannot distinguish whether the changes of CO_2 concentration are caused by people or not, and may make wrong predictions. To resolve this issue, Scenario 2 includes the outdoor CO_2 sensor so that AItalk can mitigate the natural oscillation effect by comparing indoor and outdoor CO_2 sensor values and improve prediction accuracy.

Fig. 12 shows the curves for the indoor/outdoor CO_2 concentration, the actual number of people as the ground truth, and the predicted number of people in Scenario 2 with WAV2 (measured in September 14, 2017). The figure indicates that the CO_2 sensor sometimes outputs oscillation values in a short period, which cause wrong prediction; for example, there are outliers at 0:04am, 0:26am, 1:23am, and so on. To remove these outliers, the authors design a cyber IoT device called 'Last-n average'. The authors use the last-n average device (Fig. 13(1) and (2)) to generate the final result from the last n data using plurality voting. This device keeps the last $n-1$ data. When the device receives a new datum, it chooses the most frequently occurring value among these n data and outputs this result. This mechanism immunises the short-term oscillation effect to enhance the performance of prediction. However, the n-value should be carefully selected. If a large n is selected, AItalk will be too insensitive to predict when the number of people is changed. Fig. 14 shows the accuracy

Table 2 TPR, FPR, and accuracy of machine learning ($n = 1$)

Algorithm	TPR, %	FPR, %	Scenario 1 accuracy, %	TPR, %	FPR, %	Scenario 2 accuracy, %
SVM	79.1	29.9	80.7	80.1	28.1	81.6
kNN	84.5	20.6	85.5	89.6	13.8	90.3
decision tree	80.8	20.0	81.5	94.6	4.9	94.7
random forest	83.3	22.2	84.4	94.8	6.3	95.1
WAV1	84.9	20.2	85.8	95.4	5.7	95.7
WAV2	84.4	19.7	85.3	95.4	5.7	95.7

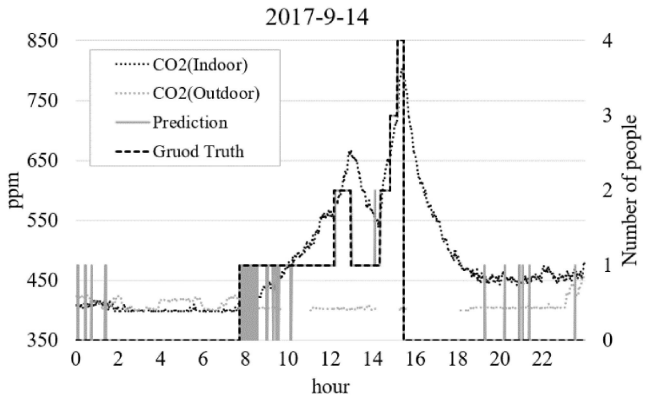

Fig. 12 *Design of CntPredict*

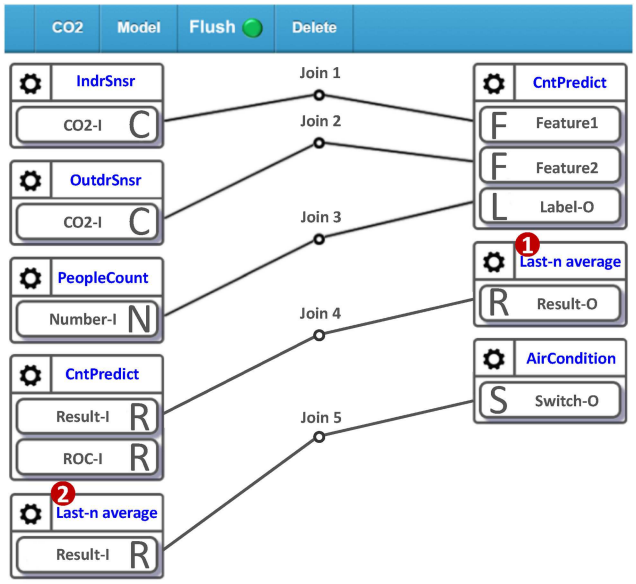

Fig. 13 *Adding the last-n average device*

curves against n. The best performance occurs at $n = 6$ where the accuracy is 96.5% as listed in Table 3. In this example, AItalk can quickly configure the outdoor temperature and humidity sensors as the features to ML_device to calibrate the indoor temperature and humidity sensors, which significantly increase the accuracy of prediction. Previous approaches [10, 11] did not explore such possibility because they did not have good tools like AItalk to configure already existed sensors as inputs to the AI model in real time.

3.3 Time complexity of AItalk

AItalk nicely integrates the distributed IoT structure with the AI applications. Through AItalk GUI, this distributed structure allows the developer to conveniently add and remove input data sources and the setup of machine-learning models. However, the distributed structure may incur extra communication delays. This subsection investigates AItalk's communication and execution overheads.

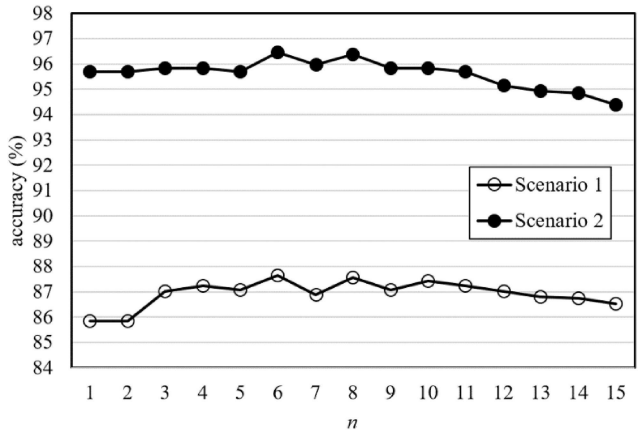

Fig. 14 *Accuracy of the last-n average mechanism (WAV1)*

AItalk decomposes an AI application into three groups of components: the IoTtalk engine (Fig. 2(b)), the input devices (Fig. 2(c), (e), and (g)), and the output devices (Fig. 2(d) and (g)).

We design three configuration cases to place these three groups of components. In Case 1, all components are located in one building in NCTU and are connected by one-hop WLAN links. In Case 2, the IoTtalk engine is located in the private cloud of the computer centre in NCTU, and the input and output devices are distributed on campus. In Case 3, the IoTtalk engine is located in the commercial cloud of Chunghwa Telecom (the largest telecom company in Taiwan), and the input and output devices are distributed on NCTU campus. Using the IoT interaction protocols, AItalk incurs two extra delays in Case i ($1 \le i \le 3$): the delay $d_{1,i}$ from an input device to the IoTtalk engine (i.e. paths (c)→(b), (e)→(b) or (i)→(b) in Fig. 2) and the delay $d_{2,i}$ from the IoTtalk engine to an output device (i.e. paths (b)→(g), (b) →(h)/(i)).

We have conducted experiments to measure d_i for Case i ($1 \le i \le 3$). In our experiments, each of the sensors, the IoTtalk server and the actuators is installed a timer module to synchronise with the Network Time Protocol (NTP). For every case, the authors conducted 1200 measurements in the commercial operation environments; that is, the involved networks operated with commercial background traffic [8]. Fig. 15 illustrates the histograms of $d_i = d_{1,i} + d_{2,i}$, where $\mathrm{E}[d_1] = 13.2$ ms, $\mathrm{E}[d_2] = 21.5$ ms, and $\mathrm{E}[d_3] = 28.4$ ms. The variances are $V[d_1] = 0.028E[d_1]^2$, $V[d_2] = 0.012E[d_2]^2$, and $V[d_3] = 0.001E[d_3]^2$. These results indicate that the delay time complexities of AItalk at a virtual machine in the cloud are slightly larger than that of a local AItalk server. The variances of all three cases are small.

We have also conducted 10,000 experiments to measure the computation times d_c for prediction, where $\mathrm{E}[d_c] = 1.386$ ms, $V[d_c] = 0.003E[d_c]^2$. Fig. 16 illustrates the d_c histogram.

4 Conclusion

Names and affiliations should immediately follow.

This paper proposed a solution called AItalk to integrate AI into existing IoT applications. Unlike traditional AI-based IoT approaches that tightly integrate the AI mechanism within the network applications, AItalk treats the machine-learning mechanism as a cyber IoT device. Our solution allows seamless inclusion of machine-learning capability to the existing IoT

Table 3 TPR, FPR and accuracy of machine learning with last-*n* averaging (*n* = 6)

Algorithm	TPR, %	FPR, %	Scenario 1 accuracy, %	TPR, %	FPR, %	Scenario 2 accuracy, %
SVM	78.3	30.4	79.9	79.5	28.8	81.0
kNN	86.7	18.3	87.6	90.1	13.2	90.8
decision tree	85.3	16.1	86.0	95.2	4.7	95.4
random forest	85.8	19.4	86.8	95.4	5.6	95.8
WAV1	86.7	17.6	87.6	96.2	4.4	96.5
WAV2	86.2	16.5	87.3	96.2	4.4	96.5

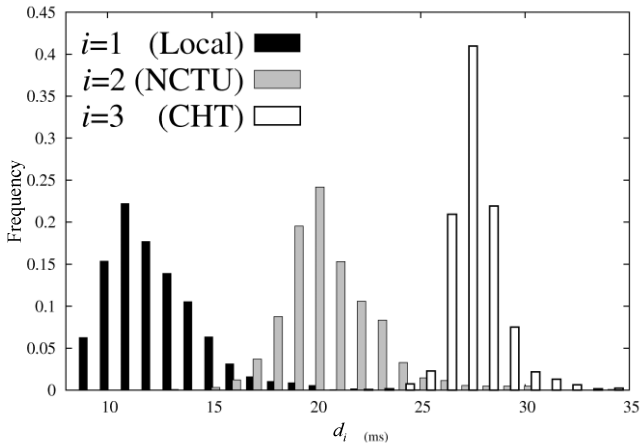

Fig. 15 *Histograms of* d_i $(1 \leq i \leq 3)$

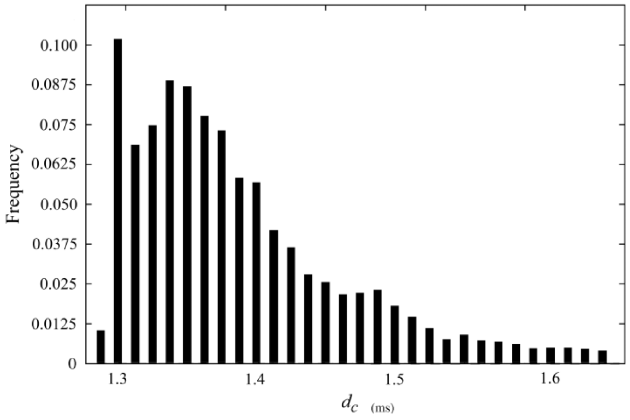

Fig. 16 *Histogram of* d_c

applications without any programming effort. AItalk decomposes a complex AI application into simplified distributed modules connected using the IoT technology such that real-time data can be transparently included in the AI applications without extra programming effort. Supervised machine learning naturally fits the IoT applications, where the sensors provide the features to the AI algorithms, and the remote controllers serve as the labels. Since the authors implement AItalk based on IoTtalk, it inherits the properties of IoTtalk, including connectivity, availability, and scalability.

We used indoor human occupancy counting as an example to demonstrate that AItalk can detect the population size in real time with CO_2 sensors, which cannot be achieved in the previous approaches with the same sensors. To support real-time computation of an AI application, the authors show that the overhead of the IoT communication in AItalk is <30 ms. Authors' study proves that AItalk provides a user-friendly environment for AI application development and execution with reasonable cost. In the future, the authors will implement new ML_device that can nicely accommodate TensorFlow [21], Google Cloud AutoML [22], and Microsoft Azure ML [23] to add powerful AI capability to AItalk.

5 Acknowledgments

This work was supported in part by Ministry of Science and Technology (MOST) 106-2221-E-009-049-MY2, 107-2221-E-009-039, and "Center for Open Intelligent Connectivity" of National Chiao Tung University and Ministry of Education, Taiwan, R.O.C.

6 References

[1] Gomes, H.M., Barddal, J.P., Enembreck, F., *et al.*: 'A survey on ensemble learning for data stream classification', *ACM Computing Surveys (CSUR) Journal*, 2017, **50**, (2), pp. 1–36
[2] Jindal, A., Dua, A., Kaur, K., *et al.*: 'Decision tree and SVM-based data analytics for theft detection in smart grid', *IEEE Trans. Ind. Inf.*, 2016, **12**, (3), pp. 1005–1016
[3] Ahmad, A.S., Hassan, M.Y., Abdullah, M.P., *et al.*: 'A review on applications of ANN and SVM for building electrical energy consumption forecasting', *Renew. Sustain. Energy Rev.*, 2014, **33**, pp. 102–109
[4] Belgiu, M., Drăguţ, L.: 'Random forest in remote sensing: a review of applications and future directions', *ISPRS J. Photogramm. Remote Sens.*, 2016, **114**, pp. 24–31
[5] Aburomman, A.A., Reaz, M.B.I.: 'A novel SVM-kNN-PSO ensemble method for intrusion detection system', *Appl. Soft Comput.*, 2016, **38**, pp. 360–372
[6] Bobriakov, I.: 'Top 15 python libraries for data science in 2017'. Available at https://activewizards.com/blog/top-15-libraries-for-data-science-in-python, accessed June 2017
[7] Lin, Y.-B., Lin, Y.-W., Huang, C.-M., *et al.*: 'IoTtalk: a management platform for reconfigurable sensor devices', *IEEE Internet Things J.*, 2017, **4**, (5), pp. 1152–1562
[8] Lin, Y.-W., Lin, Y.-B., Hsiao, C.-Y., *et al.*: 'IoTtalk-RC: sensors as universal remote control for aftermarket home appliances', *IEEE Internet Things J.*, 2017, **4**, (4), pp. 1104–1112
[9] Lin, Y.-W., Lin, Y.-B., Yang, M.-T., *et al.*: 'ArduTalk: an arduino network application development platform based on IoTtalk', *IEEE System Journal*, 2017, pp. 1–9
[10] Arief-Ang, I.B., Salim, F.D., Hamilton, M.: 'CD-HOC: indoor human occupancy counting using carbon dioxide sensor data', *Computing Research Repository*, 2017, abs/1706.05286, pp. 1–24
[11] Saelens, D., Parys, W., Baetens, R.: 'Energy and comfort performance of thermally activated building systems including occupant behavior', *Build. Environ.*, 2011, **46**, (4), pp. 835–848
[12] AllSeen Alliance, 2016. Available at https://allseenalliance.org/, accessed 13 July 2017
[13] oneM2M: 'Standards for M2M and the internet of things', 2016. Available at http://www.onem2m.org/, accessed 13 July 2017
[14] 'Scikit-learn, Machine Learning in Python'. Available at http://scikit-learn.org, accessed 2018
[15] 'Arduino, Getting Started with the Arduino Yún'. Available at https://www.arduino.cc/en/Guide/ArduinoYun, accessed 2018
[16] Dietterich, T.G.: 'Ensemble methods in machine learning'. Multiple Classifier Systems, Cagliari, Italy, 21–23 June, December 2000 (LNCS, **1857**), pp. 1–15
[17] DOE: 'Building energy databook'. Technical report, US Department of Energy, 2011
[18] Breiman, L.: 'Random forests', *Mach. Learn.*, 2001, **45**, (1), pp. 5–32
[19] Geurts, P., Ernst, D., Wehenkel, L.: 'Extremely randomized trees', *Mach. Learn.*, 2006, **63**, (1), pp. 3–42
[20] Noi, P. T., Kappas, M.: 'Comparison of random forest, k-nearest neighbor, and support vector machine classifiers for land cover classification using sentinel-2 imagery', *Sensors*, 2018, **18**, (1), p. 18
[21] Abadi, M., Agarwal, A., Barham, P., *et al.*: 'Tensorflow: large-scale machine learning on heterogeneous systems', 2015. Available at tensorflow.org
[22] Li, F.-F., Li, J.: 'Cloud AutoML: making AI accessible to every business'. Available at https://www.blog.google/topics/google-cloud/cloud-automl-making-ai-accessible-every-business/, accessed 29 Mar. 2018
[23] Mund, S.: '*Microsoft Azure machine learning*' (Packt Publishing, Birmingham, UK, 2015)