machines

MDPI

Article

A Cyber Manufacturing IoT System for Adaptive Machine Learning Model Deployment by Interactive Causality-Enabled Self-Labeling

Yutian Ren *, Yuqi He, Xuyin Zhang, Aaron Yen and Guann-Pyng Li *

California Institute for Telecommunications and Information Technology, University of California, Irvine, CA 92697, USA; yuqih20@uci.edu (Y.H.); xuyinz1@uci.edu (X.Z.); ahyen@uci.edu (A.Y.)
* Correspondence: yutianr@uci.edu (Y.R.); gpli@calit2.uci.edu (G.-P.L.)

Abstract: Machine learning (ML) has been demonstrated to improve productivity in many manufacturing applications. To host these ML applications, several software and Industrial Internet of Things (IIoT) systems have been proposed for manufacturing applications to deploy ML applications and provide real-time intelligence. Recently, an interactive causality-enabled self-labeling method has been proposed to advance adaptive ML applications in cyber–physical systems, especially manufacturing, by automatically adapting and personalizing ML models after deployment to counter data distribution shifts. The unique features of the self-labeling method require a novel software system to support dynamism at various levels. This paper proposes the AdaptIoT system, comprising an end-to-end data streaming pipeline, ML service integration, and an automated self-labeling service. The self-labeling service consists of causal knowledge bases and automated full-cycle self-labeling workflows to adapt multiple ML models simultaneously. AdaptIoT employs a containerized microservice architecture to deliver a scalable and portable solution for small and medium-sized manufacturers. A field demonstration of a self-labeling adaptive ML application is conducted with a makerspace and shows reliable performance with comparable accuracy at 98.3%.

Keywords: adaptive machine learning; industrial IoT; software system; manufacturing system; smart manufacturing

check for updates

Academic Editors: Huosheng Hu, Kai Cheng, Angelos P. Markopoulos, Panagiotis Karmiris-Obratański and Emmanouil-Lazaros Papazoglou

Received: 6 February 2025
Revised: 24 March 2025
Accepted: 2 April 2025
Published: 8 April 2025

Citation: Ren, Y.; He, Y.; Zhang, X.; Yen, A.; Li, G.-P. A Cyber Manufacturing IoT System for Adaptive Machine Learning Model Deployment by Interactive Causality-Enabled Self-Labeling. *Machines* **2025**, *13*, 304. https://doi.org/10.3390/machines13040304

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The integration of real-time machine learning (ML) technology into cyber–physical systems (CPSs), such as smart manufacturing, requires a hardware and software platform to orchestrate sensor data streams, ML application deployments, and data visualization to provide actionable intelligence. Contemporary manufacturing systems leverage advanced cyber technologies such as Internet of Things (IoT) [1] systems, service-oriented architectures [2], microservices [3], and data lakes and warehouses [4]. ML applications can be integrated with existing tools to support and enable smart manufacturing systems. For example, Yen et al. developed a software-as-a-service (SaaS) framework for managing manufacturing system health with IoT sensor integration that can facilitate data and knowledge sharing [5]. Mourtzis et al. proposed an IIoT system for small and medium-sized manufacturers (SMMs), incorporating big data software engineering technologies to process the generation and transmission of data at the terabyte-scale monthly for a shop floor with 100 machines [6]. Liu et al. designed a service-oriented IIoT gateway and data schemas for just-enough information capture to facilitate efficient data management and transmission

in a cloud manufacturing paradigm [7]. Sheng et al. proposed a multimodal ML-based quality check for CNC machines deployed using edge (sensor data acquisition) to cloud (Deep Learning compute) collaboration [8]. Morariu et al. designed an end-to-end big data software architecture for predictive scheduling in service-oriented cloud manufacturing systems [9]. Paleyes et al. [10] summarized the challenges in deploying machine learning systems in each stage of the ML lifecycle. For manufacturing companies, especially SMMs, the relatively outdated IT infrastructure, lack of IT expertise, and heterogeneous nature of manufacturing software and hardware systems complicate ML application deployment [11]. While systems in the literature have demonstrated various ML applications, they lack support for adaptive ML.

A major component of the cyber manufacturing paradigm is actionable intelligence, providing users with critical information to act at the right time and place. Manufacturers significantly favor personalized intelligence for its ability to adapt to their specific use cases. However, barriers exist to the development and deployment of personalized ML systems in manufacturing environments. The cost of manually collecting and annotating a training dataset slows the democratization of ML-enhanced smart manufacturing systems, especially in SMMs [11]. Recently, the development of adaptive machine learning, which autonomously adapts ML models to diverse deployment environments, has become a viable solution to lower the entry barrier to ML for SMMs. Several types of adaptive ML methods, including pseudo-labels empowered by semi-supervised learning (SSL) [12,13], delayed labels [14,15], and domain knowledge-enabled learning [16,17], have been proposed. These adaptive ML methods can be deployed to achieve personalized intelligence in manufacturing via the way of semi-supervised learning.

Recently, a novel interactive causality-based self-labeling method has been proposed to achieve adaptive machine learning and has been demonstrated in manufacturing cyber–physical system applications [18,19]. This method utilizes causal relationships extracted from domain knowledge to enable an automatic post-deployment self-labeling workflow to adapt ML models to local environments. The self-labeling method works in real time to automatically capture and label data and is able to effectively utilize limited pre-allocated or public datasets. Self-labeling is a coordinated effort between three types of computational models, namely task models, effect state detectors (ESDs), and interaction time models (ITMs), to execute the self-labeling workflow for adapting task models after deployment. An overview of this method is provided in Section 2. The merit of the self-labeling method is in its ability to fully leverage the unique properties of ML applications in CPS contexts, including scenarios with rich domain knowledge, dynamic environments with time-series data and possible data shifts, and diverse environments with limited pre-allocated datasets, to fulfill the needs of personalized solutions at the edge.

To support and execute the interactive causality based self-labeling (SLB) method, especially for SMMs, the system infrastructure must support the following requirements: (1) real-time timestamped data transfer of sensor, audio, and video data from from heterogeneous services and devices; (2) a causality knowledge base that manages the interaction between models to facilitate self-labeled ML between causally related nodes; (3) a core self-labeling service that connects the ML services, routes data streams, executes the self-labeling workflow, and retrains and redeploys ML models autonomously at the edge; and (4) a scalable architecture to easily accommodate new edge, ML, and SLB services. Due to the unique needs of interactive causality, a novel software system is required to realize self-labeling functionality for various ML models. This software system harnesses real-time IoT sensor data, ML, and self-labeling services to enable self-labeling adaption of models to ever-changing environments.

Given the unique mechanism of self-labeling in terms of temporal causality, the deployment of the self-labeling method requires a novel ML feature logging and model serving paradigm that executes several models to label data streams in real time guided by existing domain knowledge.In this paper, we propose and implement the AdaptIoT system as a platform to develop cyber manufacturing applications with adaptive ML capability. The AdaptIoT platform employs mainstream software engineering practices to achieve an affordable, scalable, actionable, and portable (ASAP) solution for SMMs. AdaptIoT defines an end-to-end IoT data streaming pipeline that supports high-throughput ($\geq$100 k msg/s) and low-latency ($\leq$1 s) sensor data streaming via HTTP and defines a standard interface to integrate ML applications that ingest sensor data streams for inference. The most important feature of AdaptIoT is its inherent support for self-labeling, managing various computational models (e.g., ML models) to automatically execute a flexible self-labeling workflow to collect and annotate data without human intervention to retrain and redeploy ML models. A causality knowledge base is incorporated to store and manage the virtual interactions among computational models for self-labeling. AdaptIoT employs a scalable micro-service architecture that can easily integrate future capabilities such as data shift monitoring. We deploy AdaptIoT in a small-scale makerspace to simulate its application in SMMs and develop a self-labeling application using the AdaptIoT platform, demonstrating the applicability and the adaptive ML capability of AdaptIoT in real-world environments. Part of the platform source code is open-sourced at https://github.com/yuk-kei/iot-platform-backend (accessed 30 March 2025). AdaptIoT is established on the successful demonstration of the self-labeling method and is designed to support its real-time deployment in SMMs. The unique requirements of self-labeling described before set up the novelty of AdaptIoT, which is by design to satisfy all the needed functions..

2. Overview of Interactive Causality and Self-Labeling Method

Cyber manufacturing systems for AI applications have been discussed a lot in the literature. For example, Yen et al. developed a software-as-a-service (SaaS) framework for managing manufacturing system health with IoT sensor integration that can facilitate data and knowledge sharing [5]. Mourtzis et al. proposed an IIoT system for small and medium-sized manufacturers (SMMs), incorporating big data software engineering technologies to process the generation and transmission of data at the terabyte-scale monthly for a shop floor with 100 machines [6]. Liu et al. designed a service-oriented IIoT gateway and data schemas for just-enough information capture to facilitate efficient data management and transmission in a cloud manufacturing paradigm [7]. Sheng et al. proposed a multimodal ML-based quality check for CNC machines deployed using edge (sensor data acquisition) to cloud (Deep Learning compute) collaboration [8]. Morariu et al. designed an end-to-end big data software architecture for predictive scheduling in service-oriented cloud manufacturing systems [9]. Paleyes et al. [10] summarized the challenges in deploying machine learning systems in each stage of the ML lifecycle. These existing methods do not consider the special use case of self-labeling that requires real-time data stream cross-labeling with a predictable time lag, and therefore will require high customization in the self-labeling scenario. This paper aims to address this issue and provides a generic platform solution for self-labeling applications.

The interactive causality-enabled self-labeling (SLB) method is developed to achieve fully automatic post-deployment adaptive learning for ML systems such that deployed ML models can adapt to local data distribution changes (e.g., concept drift [20].) This section includes a brief review of the self-labeling technique.

Self-labeling begins with selecting two causally connected nodes within a dynamic causal knowledge graph (KG) [21], which can be obtained from domain knowledge and

ontology. In the minimum case where the selected nodes are adjacent, the cause and effect events are related by an interaction time between their occurrence. This interaction time can vary but typically has a correlation with the effect state transient [18].

SLB requires monitoring one or more data streams so that the cause and effect state transitions can be observed. In Figure 1, the cause and effect time-series data streams are collected at nodes o_1 and o_2, respectively. The first of the three models required for SLB is the effect state detector (ESD), which monitors the data streams that provide effect data and is responsible for identifying effect state transients, including classification. The interaction time model (ITM) intakes the effect data within windows selected by the ESD, optionally including the ESD output, and predicts the interaction time (i.e., causal time lag [22]) between the cause and effect state transitions. The relevant portion of the cause data stream is extracted using the effect state transition timestamp determined by the ESD and the interaction time output by the ITM. With the cause data associated with effect state transitions, we use effect transitions as the label and cause data as the input features to train the task model.

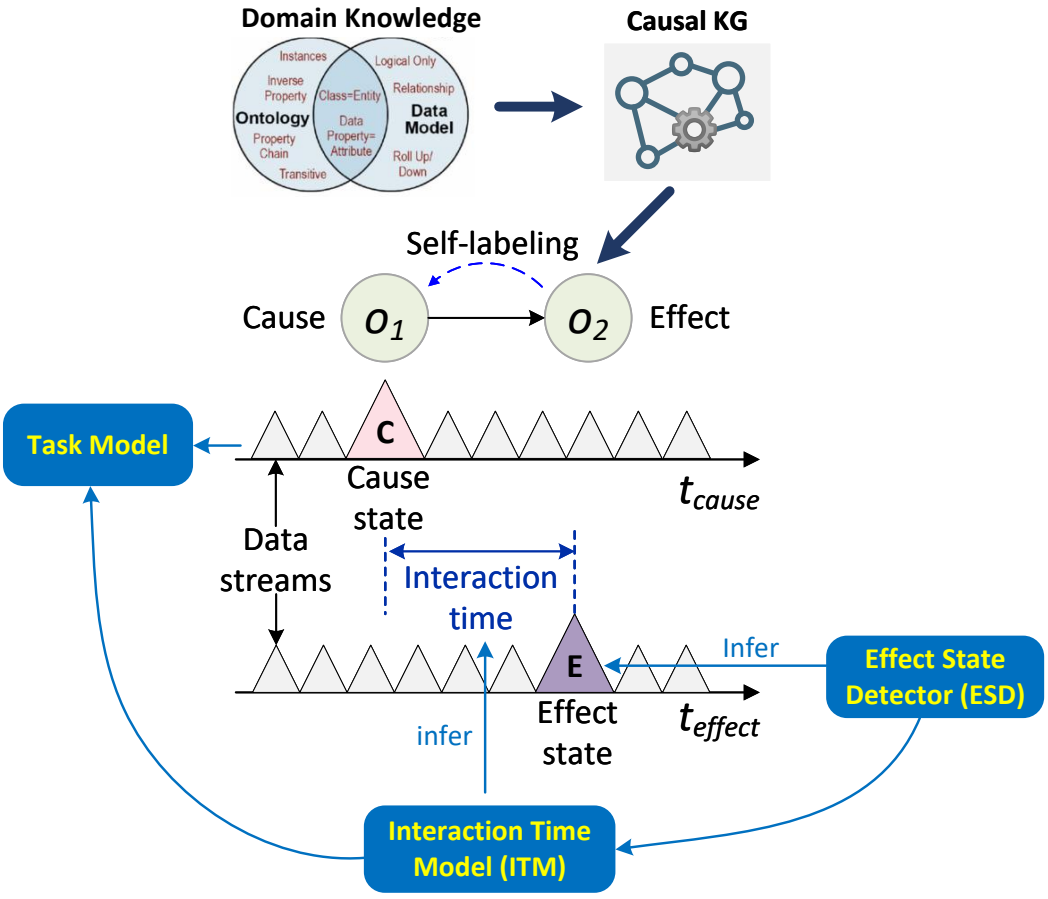


Figure 1. An illustration of the overall procedure of self-labeling.

The task model is our primary decision model, enriched by SLB through continual learning. Continual learning through self-labeling is particularly beneficial in scenarios where the input and/or output data distributions shift from their values during initial training. The relationship between cause and effect is resilient to drifts in data, and this resiliency is inherited by the self-labeling method to provide a basis for continual learning. Time-series data streams are collected for each system, with the causal relationship defining a cause and an effect system. A key advantage of the self-labeling method is its ability to independently detect and label the effect system and propagate said label to the relevant time-series data in the cause system, automating the continual learning described above.

This allows for a robust predictive classifier to be implemented without necessitating human intervention to facilitate continual learning.

3. Software Architecture of AdaptIoT for Self-Labeling

In this section, we describe AdaptIoT's modular software architecture and illustrate the specialized modules for self-labeling applications.

3.1. Module-Level Architecture

To meet the unique requirements of self-labeling applications as stated in Section 1, a high-level system block diagram is illustrated in Figure 2. The major functional modules of the system are composed of edge services, the Data Streaming Manager (DSM), databases for storage, and clusters of ML services (note that even though named as ML services for generalizability, some of them can be simple data/signal processing or statistical models), the Interaction Causality Engine (ICE), and a frontend Graphical User Interface (GUI) handler. The edge services comprise sensors, edge computing devices, external applications, and machines on the factory floor. The in situ edge services stream data via the DSM to databases and applications, including the ICE and ML services. The DSM is the main message broker, routing high-throughput streaming data generated by edge, ML, and ICE services to their destination. The DSM serves as the backend for data streaming and service management.

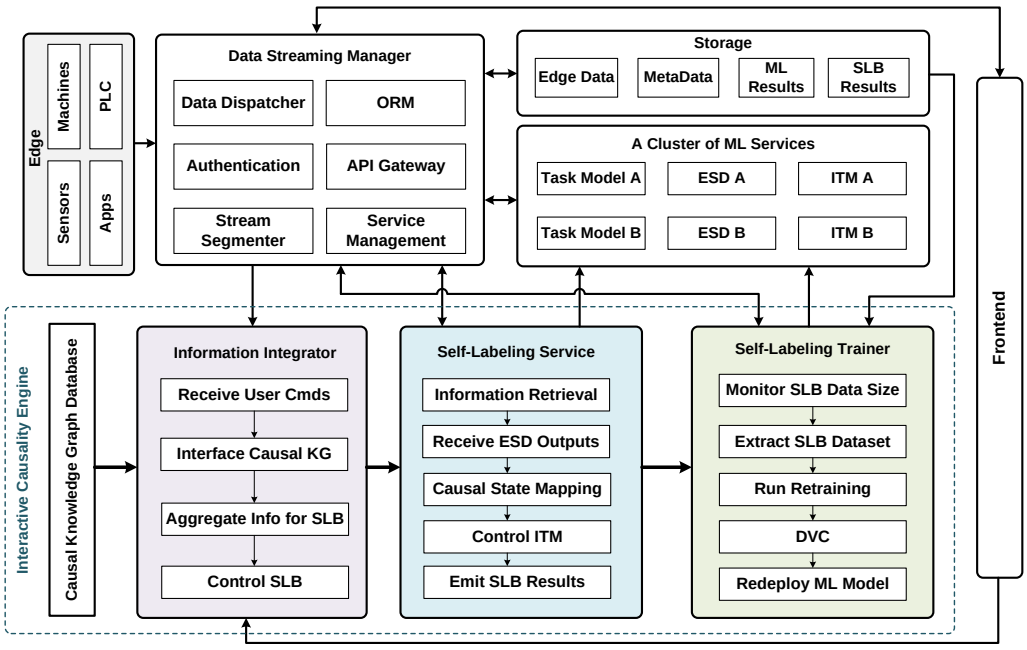


Figure 2. A high-level block diagram of the proposed IIoT system for self-labeling applications.

To efficiently store various types of data, multiple database types are implemented, including time-series databases, SQL, and no-SQL databases. The databases store raw timestamped sensor data, metadata for services and devices, processed ML results, and self-labeling results. In addition, a cluster of ML services, including task models, ESDs, and ITMs, runs to provide actionable intelligence while participating in the self-labeling workflow.

3.2. Interactive Causality Engine

The Interactive Causality Engine (ICE) is the core engine enabling adaptability for deployed ML task models. The ICE consists of a causal knowledge graph database, an information integrator, a self-labeling service, and a self-labeling trainer. The four com-

ponents undertake different tasks and jointly execute the self-labeling workflow in an automatic manner.

The causal knowledge graph database stores multiple KGs with directional links that represent the interactivity and underlying causality among the linked nodes. These KGs are extracted and reformulated from existing domain knowledge. A simplified KG sample of a 3D printer is shown in Figure 3a. A direct link in this graph represents interactivity, and connections between nodes suggest the possibility of causality between the nodes. The links in the KG can be bidirectional, differing from many causal graph model definitions (such as structural causal models [23]) as the low-level causal relationships between connected nodes are broken down into state-level representations at the temporal scale. In simple terms, at a high level two bilaterally linked nodes can be mutually causally related, but at finer temporal resolutions, in any instance one side serves exclusively as the cause and the other as the effect. Given two connected nodes, the state transition mappings (i.e., logical relations) of two nodes are represented as a dynamic and temporal state machine, as exemplified in Figure 3b. We represent this state machine with corresponding state transition relationships by using a truth table.

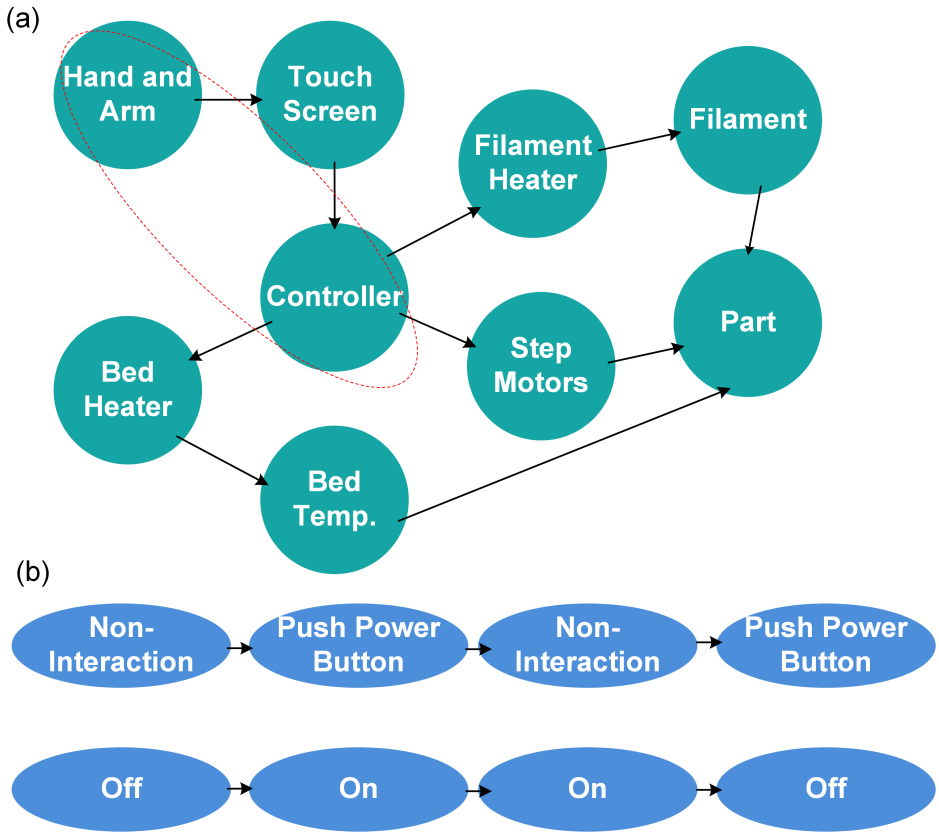


Figure 3. (**a**) A simplified knowledge graph for a 3D printer use case; (**b**) corresponding state transition relationships of the causally related Hand and Arm node and Controller node.

The information integrator bridges the causal KG database, the self-labeling service, sensor metadata, ML services, and users to ingest and integrate needed information and control the self-labeling. Through the information integrator, users can start or stop a self-labeling workflow among the causally linked nodes. The information integrator also scrutinizes the information completeness for running a self-labeling service.

The self-labeling service receives inputs from the information integrator and initiates a self-labeling workflow by coordinating the raw data streams from sensors, corresponding ML services, and the self-labeling trainer. When a self-labeling service starts, the following

functions will be executed: (1) receive control signals from the information integrator to start or stop a self-labeling workflow; (2) receive inputs from the information integrator, including the selected causal nodes, the truth table representing the causal logical relations, the URLs of corresponding sensor streams and ML services, and the output paths (URLs); (3) receive outputs from ESDs and execute causal state mapping to find consistent cause states; (4) assemble inputs for ITMs and route them to corresponding ITMs; (5) receive ITM outputs, combine ITM outputs with corresponding cause states, and emit them to a database for storage; and (6) optionally select corresponding data segments from cause streams based on the information in Step 5. Note that since the actual interaction time of each effect can be very different, Step 3 inspects whether self-labeling the causes needs to wait for additional effect states to be detected. The self-labeling service can run multiple self-labeling workflows in parallel for various nodes in KGs.

The self-labeling trainer is an independent and decoupled service that constantly monitors the number of self-labeled samples, receives users' commands via the information integrator, retrains task models, and redeploys the task models. It is designed to be separate from the self-labeling service for reusability and extensibility. The self-labeling trainer will schedule a training session at non-peak hours when the number of self-labeled samples reaches the requirements with user approval. In addition, data version control (DVC) [24] is applied to version self-labeled datasets and trained weight files for MLOps to efficiently manage the continuous retraining empowered by self-labeling. After retraining, users can choose whether to redeploy the task model with the new weights.

3.3. Unit Service Model

To connect and scale to heterogeneous edge services and ML services, an abstract layer-wise unit service model is designed to work as the fundamental architecture for a single service in the proposed AdaptIoT system. The unit service model is designed to accommodate and standardize all types of services in the system that generates data and sends generated data to a storage place. This layer-wise architecture for a single service ensures the scalability and homogeneity of downstream interfaces. The unit service model is abstracted into four layers from the bottom up: the asset layer, data generation (DataGen) layer, service layer, and API layer.

The asset layer defines an abstraction of the independent components connected to the system, such as hardware (e.g., sensors and machines), external applications (e.g., proprietary software), or external data sources (e.g., external database). A key uniqueness of this layer is that the system can interface with the independent components to receive data or run applications but cannot control or access their sources.

The data generation layer encapsulates a software that generates one data sample upon being called once. This layer performs the core function of data generation by interacting with the asset layer. A higher-level abstraction of heterogeneous edge applications is achieved in this layer by defining uniform class attributes and functions. For example, the sensor firmware as the asset layer communicates with the DataGen layer to retrieve one data sample per call. The inference function of an ML model using various ML frameworks, e.g., scikit-learn, PyTorch, or Tensorflow, is unified with the same interface to interact with the service layer. To receive data generated by external applications, we define a *Receiver* function using REST API to accept a POST request from external applications and data sources. The POST request after scrutinization is rerouted in the *Receiver* for the DataGen layer to use GET to acquire samples individually.

The service layer integrates necessary functions as a microservice on top of the data generation. It handles receiving inputs from upper API layers, i.e., inputs needed for ML inference in the DataGen layer. It integrates the inputs and the DataGen layer to generate

data in a discrete or continuous way. Upon the generation of new data, an emitter function is executed to send out data to the following pipeline. Besides interacting with the DataGen layer, the service layer integrates other auxiliary functions, including control (i.e., start, stop, and update), service registration, and metadata management, for the API endpoints in the API layer. Up to this layer, all the heterogeneous applications from the bottom are consolidated with a homogeneous interface.

The top layer is the API layer, where the API endpoints are defined using a web framework. This layer handles all the API-level I/O and interactions with other services by calling functions defined in the service layer.

Besides the four basic layers, an orchestration layer is designed to moderate the same types of services with the same or different configurations operating on the same hardware. This layer is optional, depending on the actual needs of service orchestration.

4. System Implementation and Analysis

This section details the system implementation of AdaptIoT, including the software and hardware infrastructure, and provides an example implementation of a self-labeling service hosted on AdaptIoT.

4.1. Hardware Infrastructure of the Cyber Makerspace

Figure 4 illustrates a complete implementation of the proposed AdaptIoT system for self-labeling applications and the hardware infrastructure, including manufacturing equipment. This system is deployed to a cyber makerspace lab with common manufacturing equipment, including two 3D printers, two CNC machines (Mill and Lathe), a collaborative robot, and one TIG welding machine. We heavily instrument each machine and the entire space with sensors to collect multimodal signals, including cameras, power meters, vibration, acoustic, distance, environmental, and other specialized sensors. The sensors are installed at multiple locations and critical components of the machine. In addition, the CNC machines and the robot are controlled by programmable logic controllers (PLCs) that are interfaced to acquire information about machine running status directly.

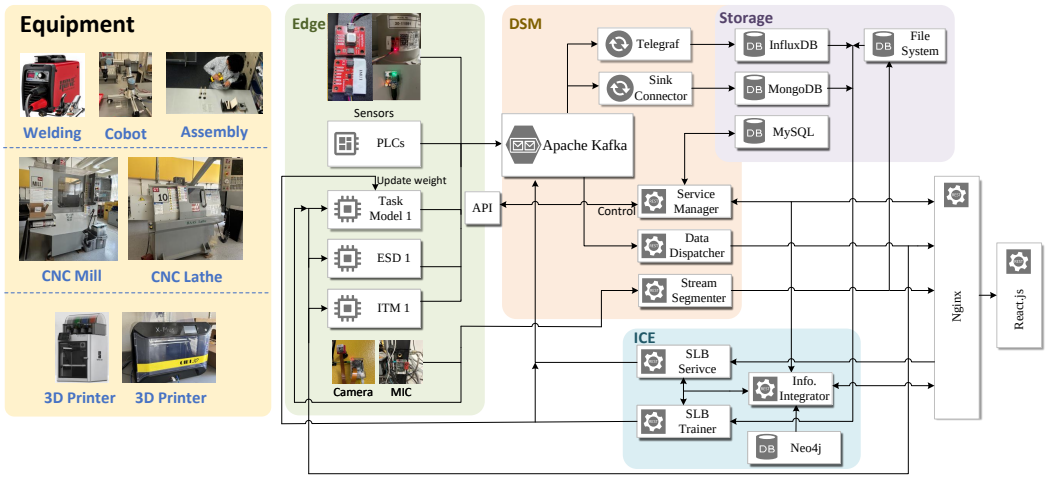


Figure 4. Hardware and software infrastructure. Each block represents a containerized software service.

4.2. AdaptIoT System Implementation

The implemented services and software components are shown in Figure 4 with examples of edge services. Figure 5 shows an example of the web-based GUI. Each block in Figure 4 represents an independent dockerized [25] web service running on various hardware, which executes one or more functions, as described in Figure 2. Except for databases, all other communications among services are via REST API using the lightweight

Flask web framework. Four services, service manager, data dispatcher, stream segmenter, and information integrator, are exposed to the React frontend via Nginx. The internal APIs are hidden behind the four previously mentioned services.

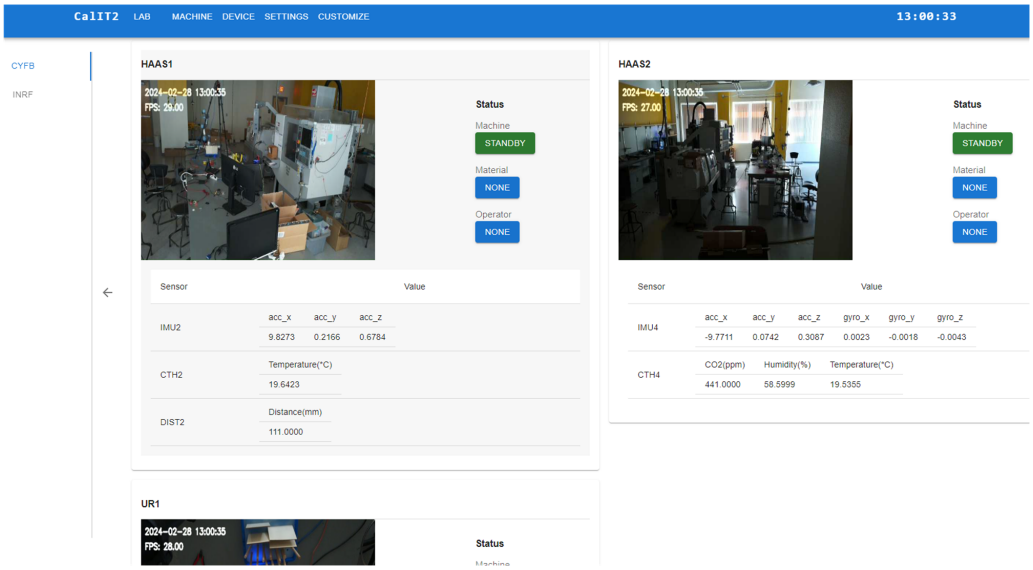


Figure 5. The web GUI to display real-time data and ML results where users can also apply control to the system.

4.2.1. Software Components

Message queue. A message queue is a communication method used in distributed systems and computer networks for asynchronous communication between various components or processes. The key feature of a message queue is that it decouples the producers and consumers in terms of time and space. Producers and consumers do not need to run simultaneously or on the same machine. This decoupling is useful in building scalable and flexible systems, as components can communicate without being directly aware of each other. Due to these features, we choose a message queue as the main message broker in the DSM. Popular message queue systems include Apache Kafka, RabbitMQ, and Apache ActiveMQ [26,27]. This study uses Kafka due to its outstanding horizontal scalability and high throughput.

Database and storage. Several types of data need to be stored, and accordingly, several types of storage are chosen. We consider factors including data structure, throughput, size, access frequency, and scalability. A MySQL database stores static metadata for all the services and users. For example, the relational metadata for a sensor service includes its factory locations, associated machines, vendor information, and URL for obtaining data. An IoT system with streaming sensors requires a continuous high data throughput (e.g., $\geq$10 k samples/s), which puts additional demand on the database ingestion speed. Time-series databases are typically designed to handle high throughput, especially in scenarios with a continuous influx of timestamped data. For storing high-throughput sensor data, a time-series database InfluxDB [28] is chosen. Regarding the results generated by ML services, we use both MongoDB and MySQL, depending on the data types. In addition, a graph database Neo4j [29] is chosen to store the causal knowledge graph. Video and audio data are stored in file systems only.

Implementation of the unit service model. Figure 6 describes the detailed implementation of a unit service model in connection with an external application. Starting from the asset layer, external applications in various languages and platforms can post an authenticated JSON-format message via REST API to the *Receiver* from which an asset layer function *GetDataViaWeb* can obtain data. The *Receiver* connects with the API gateway

to serve as the system-level data ingestion service for external data sources. The *DataGen* class calls *GetDataViaWeb* continuously to acquire the JSON message and wrap it into a defined standard message format. The *Service* class in the service layer integrates *DataGen* and input sources *GetStream* to execute service-level functions. The received inputs from *GetStream* in JSON format are transmitted downwards to the data generation layer and possibly asset layer for processing. In class *Service*, the generated data are sent to an *Emitter* that manages all the data transmission to Kafka. An *Orchestrator* stays on top of multiple *Services* of the same type for control and interaction with other unit services. A Flask API layer provides a lightweight web server for each unit service and defines the API endpoints on the top.

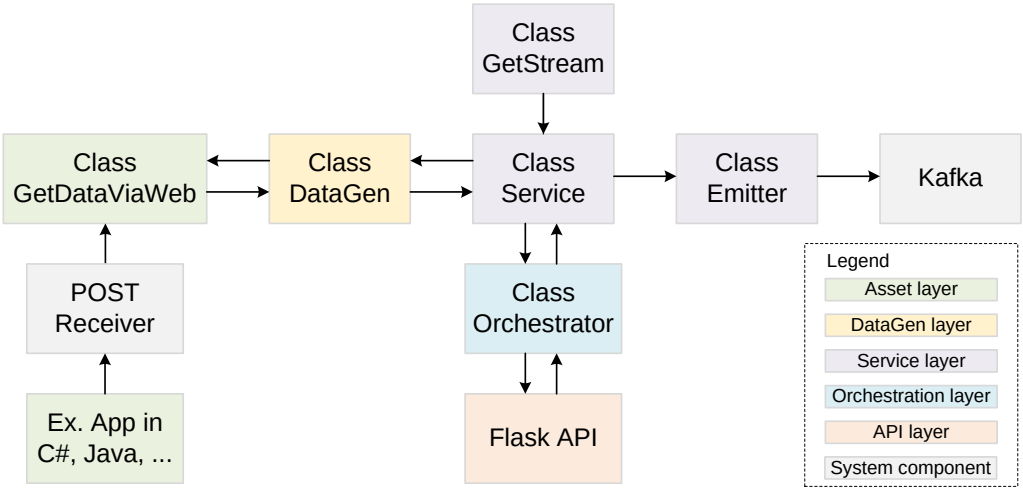


Figure 6. A unit service example of receiving data from an external application.

4.2.2. Data Flow

As an illustration, we will describe a complete data flow in AdaptIoT from an edge sensor to an ML service. An edge sensor encapsulated in a unit service model generates a sample and emits it to the Kafka cluster. In the Kafka cluster, the sample is allocated to a partition for processing, after which this sample is routed to two places. First, the sample is routed by Telegraf to the InfluxDB for persistence. In the meantime, due to the unique requirement of many ML applications that need continuous data processing, the **data dispatcher** is implemented to route received individual samples into an HTTP data streaming via Server-Sent Events (SSEs) [30] and a query interface via REST API. ML services that need this data stream can use the standard HTTP method to receive the stream. The inferred ML results are emitted to Kafka again and routed to the corresponding MongoDB and data dispatcher. The React frontend queries the APIs for visualization.

4.3. ICE Implementation

Two types of data structures are used to represent the causality among nodes in a KG and the exact causal logical relations between any selected nodes. For the causal knowledge graph, we use a graph database Neo4j [29] to represent the nodes, attributes of nodes, and the directional relationships among nodes. The truth table is used to represent various causal logical relations among arbitrary nodes. The truth tables are stored in MongoDB in key-value pairs.

We define a standard class *SlbService* that can apply the self-labeling method on any causally related ML services given relevant parameters. The outputs of self-labeling are three key values obtained by fusing the outputs of the ESD and ITM, including a corresponding cause state, a timestamp of the end of the cause state, and the duration of the cause state. To partition the cause data streams based on self-labeling results, the

system supports operation in two modes. Mode 1 saves the raw self-labeling outputs in MongoDB, which are used to generate a retraining dataset by the SLB trainer afterward. Mode 2 creates self-labeled data samples on the fly when *SlbService* is running to provide immediate feedback for users. Both modes can be turned on at the same time. The SLB trainer independently monitors the number of self-labeled samples by querying the database at a constant frequency and manages ML training scripts for retraining ML models.

Negative Samples. Similar to other natural label-based systems, i.e., social media recommendation systems [31] where users' interactions (likes, views, and comments) are used as positive labels, in many cases, the ESD can only provide positive labels when there are state transitions different from the background distribution. The acquisition of negative samples from the background data distribution follows the same strategy as recommendation systems via negative sampling or more advanced importance sampling. The negative sampling is undertaken by each ESD, since the ESD keeps a buffer of its own historical states. The ESD randomly samples the background distribution as the negative labels and sends them to the self-labeling service for processing.

SLB Implementation. A detailed implementation of the self-labeling service is described in Figure 7. A non-trivial situation of self-labeling is to cope with multiple asynchronous effects for causal state mapping to self-label the corresponding cause states. The SLB service needs to wait for the delayed effects in order to jointly or individually execute ITMs, or neglects detected effects if no other effects arrive in the given time period and the received effects are unable to determine a unique cause state. We apply a first-in-first-out (FIFO) queue to cache the arrived effects. The causal state mapping module regularly scans the FIFO and determines if the current effect states are enough to determine an unambiguous cause state referring to the retrieved causality from KG. In addition, the causal state mapping module monitors the timespan and evict effects states that cannot formulate a deterministic cause state due to the lack of necessary effects in a given time period. After the causal state mapping, ITMs are triggered to infer the interaction time by using the assembled effect states as inputs, after which the self-labeling results are compiled and emitted.

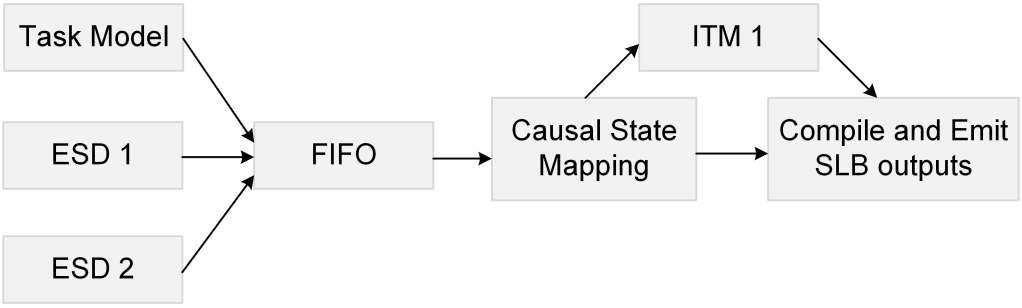


Figure 7. Self-labeling modular structures for multiple effects.

Figure 8 illustrates the interactions among ML services with the self-labeling mechanism guided by the causal knowledge graph. Initially, five ML services operate independently for state change detection of the corresponding nodes in the KG (Figure 8a). By choosing some of the causally linked nodes for self-labeling, as colored in Figure 8b, the corresponding ML services start interacting with each other via the defined self-labeling workflow. In turn, self-labeling improves detection accuracy, and the ML services can infer again independently.

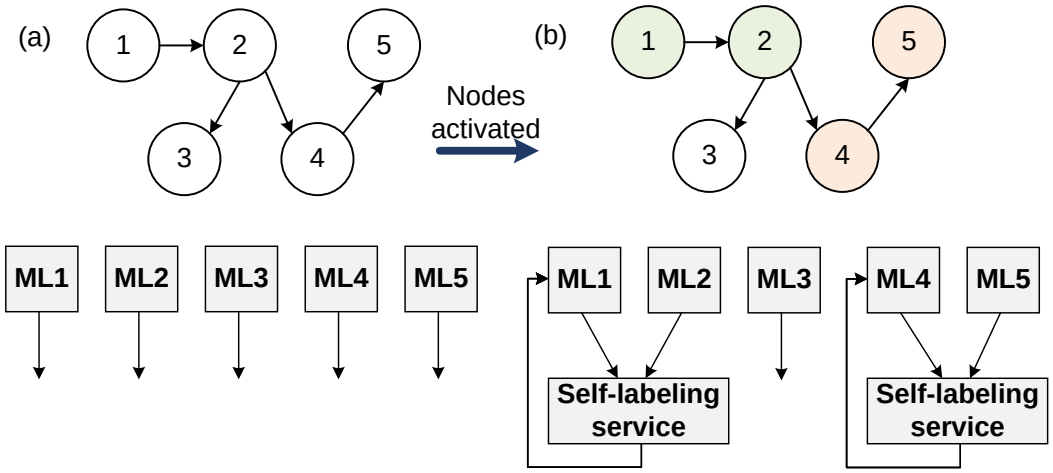


Figure 8. An illustration of virtual interactions among ML services due to the initialization of pair-wise self-labeling.

4.4. System Characterization

A system characterization of several key performance indicators is conducted to evaluate the performance of the proposed AdaptIoT system. The system's backend and frontend applications are deployed on a workstation with a 20-core Intel Xeon W-2155 (Intel Corporation, Santa Clara, CA, USA) at 3.30 GHz. The workstation's Ethernet data transfer rate is 1 Gbps.

A standard configuration of a sensor node for characterization purposes consists of one host processor, one 6-DOF IMU sensor, one CTH (CO_2, temperature, and humidity) sensor, and one distance (time of flight) sensor, while it can also be customized freely. For a single edge node with one IMU, one CTH, and one distance sensor, the average end-to-end timing performance from the data generator to the database is evaluated, and the results are shown in Table 1. We use Raspberry Pi 3B with 1G RAM and 300 Mbps Ethernet as the host processor for multiple time-series sensors installed on machines. Depending on sensor types, the sampling frequency of each sensor ranges from a few hundred Hz to 0.2 Hz.

Table 1. Test results of a single edge node.

Mean Throughput	Mean msg Size	Mean Delay	Max Delay
284 msg/s	250.2 byte	31 ms	64 ms

As for camera streaming, Raspberry Pi 4B with 8G RAM and 1 Gbps Ethernet is chosen. Correspondingly, Raspberry Pi Camera Module 3 with 1080p resolution and 30 fps is used. Each camera produces streams simultaneously in two modes: preview and full HD resolution. The preview mode streams at 240p resolution for GUI display only. The average end-to-end delay is 39 ms. The full HD mode streams at 1080p to the video segmenter for self-labeling and the corresponding ML services for inference. This design ensures that the acquired video dataset and ML inference can use high-quality images while reducing bandwidth requirements for GUI users. The average frame size is 69 KB, and theoretically, using 1 Gbit, the system can support about 60 cameras simultaneously.

To provide a baseline for characterization, we detail the system configuration below. First, a mock test is accomplished to evaluate the maximum capacity of a single Kafka producer and consumer. We use a laptop with an AMD Ryzen 7 6800H 16-core CPU and 1 Gbps network as the transmitter for hosting the mock sensor. An Apache Kafka cluster with three nodes and 10 partitions is used. The Kafka single producer test result is shown in Table 2. The maximum throughput of a single consumer is 388k msg/s, which is equivalent

to 92.5 MB/s. Note that the test is accomplished with only one producer, one consumer, and three Kafka nodes. Due to the horizontal scalability of Kafka, higher performance can be realized by proper scaling.

Table 2. Kafka single producer throughput test results.

Max Number of Messages	Mean Delay	Max Delay
182 k	679 ms	979 ms

Additionally, a realistic system capability test is conducted on the real deployment. We start with three standard sensor nodes, nine additional power meters with a 1 Hz data rate, and three additional edge sensor services, including a Tiny ML board, an IMU sensor, and a data query service for the UR3e robot, while eight camera streamings run in the background. In total, 29 edge services are actively running; on average, 108 million messages are generated daily. We monitor the data ingestion speed of InfluxDB, and the average data ingestion rate is 1259 messages per second (msg/s). Based on a single consumer with an average receiving speed of 92.5 MB/s and a time-series edge producer with an average data rate of 41 msg/s, the theoretically maximum number of supported time-series edge services is 13.2 k.

5. A Real Self-Labeling Experiment Running on AdaptIoT

To demonstrate the applicability of the proposed system for self-labeling applications, a self-labeling application is developed and deployed on AdaptIoT to demonstrate the system's efficacy. The self-labeling application utilizes the example in [19] and replicates the adaptive worker–machine interaction detection on a 3D printer. The experiment uses the concept of interactive causality to design the self-labeling system for adapting a worker action recognition model. The cause side uses cameras to detect body gestures as an indication of worker–machine interactions. The effect side uses a power meter to detect machine responses in the form of energy consumption.

The developed self-labeling application is driven by a causal knowledge graph that describes the extracted domain knowledge. This KG representing the causality embedded in the 3D printer operation among people, machines, and materials is built and loaded into the graph database with corresponding metadata, as shown in Figure 3. As an illustration, we implement five sensors for five nodes in the KG, with five corresponding ML services to detect the state change of each node. Among the five implemented nodes, two interactive nodes are chosen to conduct self-labeling, as highlighted in the red circle. The two nodes represent a worker's body movement and the machine's status change.

The implementation details are shown in Figure 9. The worker action is defined as binary, namely interaction and non-interaction. The interaction is defined as when workers push the power button to turn on/off the 3D printer, which corresponds to a change in the machine's power consumption as an effect. An ML service composed of a cascaded OpenPose [32] and graph convolution network (GCN) [19] is implemented to recognize worker action as the task model. To detect a machine's power change, a machine state recognition algorithm composed of an event detector and classifier is implemented as the ESD [33]. The ITM uses a lookup table and statistical Gaussian model to infer the interaction time. This entire self-labeling application runs on the proposed AdaptIoT system.

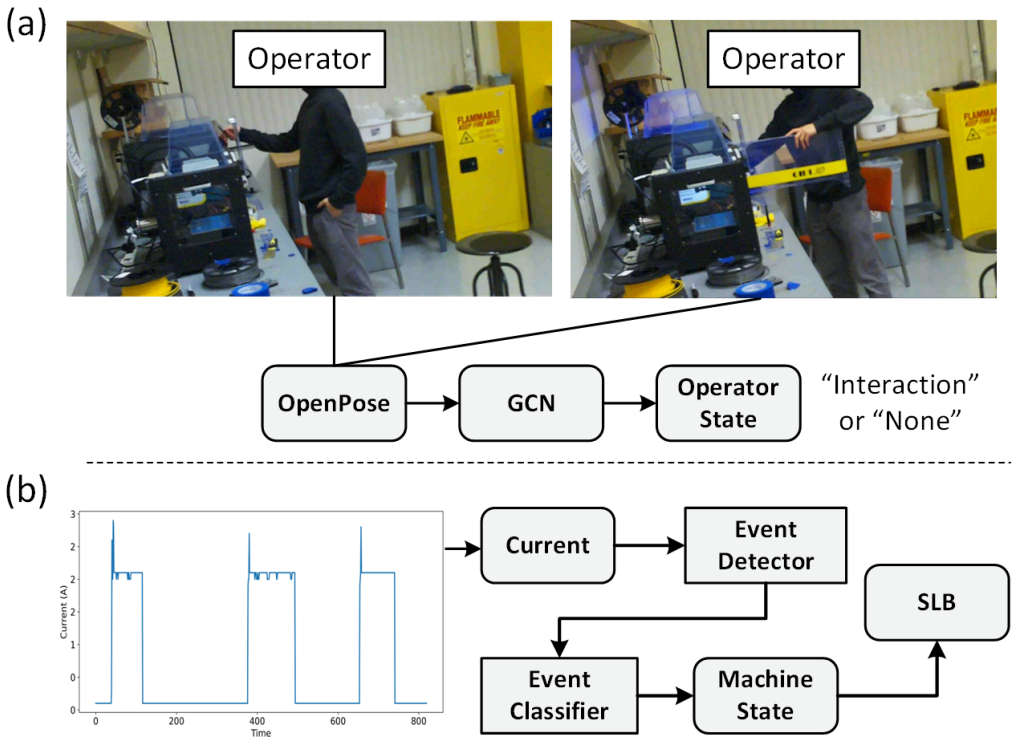


Figure 9. Experimental setup of the 3D printer self-labeling use case. (**a**) Illustrates the task model data processing pipeline, and (**b**) describes the ESD pipeline for current signals.

To demonstrate effectiveness, we manually collected and labeled a dataset of 400 samples as the validation and test sets. Through the experiment, a self-labeled dataset composed of 200 samples is automatically collected and labeled using the AdaptIoT system over three weeks of 3D printer usages. Table 3 summarizes the accuracy compared with several semi-supervised approaches. The results show the mean and standard deviation derived over the training with 10 random seeds. By default, all other semi-supervised approaches apply the temporal random shift as the data augmentation. It can be observed that the self-labeling method consistently outperforms other semi-supervised methods with a smaller standard deviation, indicating a more stabilized training, which demonstrates the applicability of the proposed AdaptIoT system for the self-labeling applications. According to the theory in [18], the self-labeling and semi-supervised methods show comparable performance when there is no observable data distribution shift, as in the situation of this experiment. The merit of self-labeling over traditional semi-supervised methods mainly manifests in the scenario of data distribution shifts, which has been demonstrated by previous studies and is not the scope of this study.

Table 3. Model accuracy (%) trained on the experiment dataset.

Method	Accuracy
PseudoLabel [34]	90.05 (±2.72)
SimMatch [35]	96.18 (±1.48)
SoftMatch [36]	98.22 (±0.68)
FreeMatch [37]	97.90 (±0.72)
Pretrain only	95.21 (±2.72)
SLB (w/ data aug.)	98.24 (±0.57)
SLB (w/o data aug.)	98.33 (±0.22)

An interesting observation is found from the experiment results, which is that the temporal random shift as the data augmentation adversely affects the self-labeling accuracy. Ref. [38] proposes a qualitative explanation of the impact of the uncertain interaction time on the self-labeling and model retraining performance. The authors use the concept of motion smoothness to explain that even though the ITM may infer the interaction time at a deviated timestamp, the natural motion smoothness alleviates the adverse effect of deviated interaction time. Hypothetically, the inaccuracy caused by interaction time inference is equivalent to the temporal random shift. The adverse effect of adding the temporal random shift to the self-labeling shown in Table 3 partially reveals this fact but requires deeper research in the future.

Discussion

The proposed system runs based on existing domain knowledge encoded into a knowledge graph. However, in real manufacturing environments, abnormal events can always occur, which presents new challenges to the self-labeling system. To accommodate and adapt to unknown events, i.e., new classes from the ML perspective, human involvement or large language models is one direction to be explored in future work.

In addition, as more and more decisions are made autonomously by AI agents, such as cobot control, operator security becomes another concern in real-world deployment. One way to leverage the proposed AdaptIoT to alleviate the security concern is to program the knowledge graph to include nodes related to security fallbacks so that fallbacks can be executed when meeting unknown or interruptive conditions. Furthermore, these added security nodes can be incorporated into the self-labeling loop to improve their ML recognition accuracy.

6. Conclusions

In this study, an IoT system, AdaptIoT, is designed and demonstrated to support the interactive causality-enabled self-labeling workflow for developing adaptive machine learning applications in cyber manufacturing. AdaptIoT is designed as a web-based microservice platform for both manufacturing IoT digitization and intelligentization, with an end-to-end data streaming component, a machine learning integration component, and a self-labeling service. AdaptIoT ensures high-throughput and low-latency data acquisition and seamless integration and deployment of ML applications. The self-labeling service automates the entire self-labeling workflow to allow real-time and parallel task model adaptation. A university laboratory as a makerspace is retrofit with the AdaptIoT system for future adaptive learning cyber manufacturing application development. Overall, more adaptive ML applications in cyber manufacturing are envisioned to be developed in the future based on the proposed AdaptIoT system.

Author Contributions: Conceptualization, Y.R. and G.-P.L.; Methodology, Y.R., Y.H., X.Z. and G.-P.L.; Software, Y.R., Y.H., X.Z. and A.Y.; Validation, Y.R., Y.H., X.Z. and A.Y.; Investigation, Y.R. and A.Y.; Data curation, Y.H. and X.Z.; Writing—original draft, Y.R.; Writing—review & editing, G.-P.L.; Visualization, Y.R.; Supervision, Y.R. and G.-P.L.; Project administration, Y.R. and G.-P.L.; Funding acquisition, G.-P.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Calit2 and Broadcom Foundation.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank Broadcom Foundation and Calit2 for the funding support.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Lu, Y.; Cecil, J. An internet of things (iot)-based collaborative framework for advanced manufacturing. *Int. J. Adv. Manuf. Technol.* **2016**, *84*, 1141–1152. [CrossRef]
2. Tao, F.; Qi, Q. New it driven service-oriented smart manufacturing: Framework and characteristics. *IEEE Trans. Syst. Man, Cybern. Syst.* **2019**, *49*, 81–91. [CrossRef]
3. Thramboulidis, K.; Vachtsevanou, D.C.; Kontou, I. Cpus-iot: A cyber-physical microservice and iot-based framework for manufacturing assembly systems. *Annu. Rev. Control* **2019**, *47*, 237–248. [CrossRef]
4. Rudack, M.; Rath, M.; Vroomen, U.; Bührig-Polaczek, A. Towards a data lake for high pressure die casting. *Metals* **2022**, *12*, 349. [CrossRef]
5. Yen, I.-L.; Zhang, S.; Bastani, F.; Zhang, Y. A framework for IoT-based monitoring and diagnosis of manufacturing systems. In Proceedings of the 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), San Francisco, CA, USA, 6–9 April 2017; pp. 1–8.
6. Mourtzis, D.; Vlachou, E.; Milas, N. Industrial big data as a result of iot adoption in manufacturing. *Procedia CIRP* **2016**, **55**, 290–295. [CrossRef]
7. Liu, C.; Su, Z.; Xu, X.; Lu, Y. Service-oriented industrial internet of things gateway for cloud manufacturing. *Robot. Comput.-Integr. Manuf.* **2022**, *73*, 102217. [CrossRef]
8. Sheng, Y.; Zhang, G.; Zhang, Y.; Luo, M.; Pang, Y.; Wang, Q. A multimodal data sensing and feature learning-based self-adaptive hybrid approach for machining quality prediction. *Adv. Eng. Inform.* **2024**, *59*, 102324. [CrossRef]
9. Morariu, C.; Morariu, O.; Răileanu, S.; Borangiu, T. Machine learning for predictive scheduling and resource allocation in large scale manufacturing systems. *Comput. Ind.* **2020**, *120*, 103244. [CrossRef]
10. Paleyes, A.; Urma, R.-G.; Lawrence, N.D. Challenges in deploying machine learning: A survey of case studies. *ACM Comput. Surv.* **2022**, *55*, 1–29. [CrossRef]
11. Davis, J.; Malkani, H.; Dyck, J.; Korambath, P.; Wise, J. Cyberinfrastructure for the democratization of smart manufacturing. In *Smart Manufacturing*; Elsevier: New York, NY, USA, 2020; pp. 83–116.
12. Yan, H.; Guo, Y.; Yang, C. Augmented self-labeling for source-free unsupervised domain adaptation. In Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Virtual, 6–14 December 2021.
13. Zhou, P.; Xiong, C.; Yuan, X.; Hoi, S.C.H. A theory-driven self-labeling refinement method for contrastive representation learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 6183–6197.
14. Grzenda, M.; Gomes, H.M.; Bifet, A. Performance measures for evolving predictions under delayed labelling classification. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; pp. 1–8.
15. Gomes, H.M.; Grzenda, M.; Mello, R.; Read, J.; Nguyen, M.H.L.; Bifet, A. A survey on semi-supervised learning for delayed partially labelled data streams. *ACM Comput. Surv.* **2022**, *55*, 1–42.
16. Ratner, A.J.; Sa, C.M.D.; Wu, S.; Selsam, D.; Ré, C. Data programming: Creating large training sets, quickly. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3574–3582.
17. Stewart, R.; Ermon, S. Label-free supervision of neural networks with physics and domain knowledge. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
18. Ren, Y.; Yen, A.H.; Li, G.-P. A self-labeling method for adaptive machine learning by interactive causality. *IEEE Trans. Artif. Intell.* **2023**, *5*, 2093–2102. [CrossRef]
19. Ren, Y.; Li, G.-P. An interactive and adaptive learning cyber physical human system for manufacturing with a case study in worker machine interactions. *IEEE Trans. Ind. Inform.* **2022**, *18*, 6723–6732. [CrossRef]
20. Lu, J.; Liu, A.; Dong, F.; Gu, F.; Gama, J.; Zhang, G. Learning under concept drift: A review. *IEEE Trans. Knowl. Data Eng.* **2018**, *31*, 2346–2363. [CrossRef]
21. Huang, H. Causal relationship over knowledge graphs. In Proceedings of the 31st ACM International Conference on Information & Knowledge Management—CIKM '22, Atlanta, GA, USA, 17–21 October 2022; Association for Computing Machinery: New York, NY, USA, 2022; pp. 5116–5119.
22. Gollob, H.F.; Reichardt, C.S. Taking account of time lags in causal models. *Child Dev.* **1987**, *58*, 80–92.
23. Pearl, J. *Causality: Models, Reasoning, and Inference*; Cambridge University Press: Cambridge, UK, 2009.
24. Kreuzberger, D.; Kühl, N.; Hirschl, S. Machine learning operations (mlops): Overview, definition, and architecture. *IEEE Access* **2023**, *11*, 31866–31879.
25. Merkel, D. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.* **2014**, *239*, 2.
26. Dobbelaere, P.; Esmaili, K.S. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-Based Systems, Barcelona, Spain, 19–23 June 2017; pp. 227–238.

27. Fu, G.; Zhang, Y.; Yu, G. A fair comparison of message queuing systems. *IEEE Access* **2020**, *9*, 421–432. [CrossRef]
28. Rinaldi, S.; Bonafini, F.; Ferrari, P.; Flammini, A.; Sisinni, E.; Bianchini, D. Impact of data model on performance of time series database for internet of things applications. In Proceedings of the 2019 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Auckland, New Zealand, 20–23 May 2019; IEEE: New York, NY, USA, 2019; pp. 1–6.
29. Fernandes, D.; Bernardino, J. Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. In *DATA 2018: Proceedings of the 7th International Conference on Data Science, Technology and Applications, Porto, Portugal, 26–28 July 2018*; Science and Technology Publications: Setubal, Portugal, 2018; Volume 10.
30. Vinoski, S. Server-sent events with yaws. *IEEE Internet Comput.* **2012**, *16*, 98–102. [CrossRef]
31. Covington, P.; Adams, J.; Sargin, E. Deep neural networks for youtube recommendations. In Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, 15–19 September 2016; pp. 191–198.
32. Cao, Z.; Simon, T.; Wei, S.-E.; Sheikh, Y. Realtime multi-person 2D pose estimation using part affinity fields. In Proceedings of the 2017 CVPR, Honolulu, HI, USA, 21–26 July 2017.
33. Ren, Y.; Li, G.P. A contextual sensor system for non-intrusive machine status and energy monitoring. *J. Manuf. Syst.* **2022**, *62*, 87–101.
34. Lee, D.-H. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In Proceedings of the ICML 2013 Workshop: Challenges in Representation Learning (WREPL), Atlanta, GA, USA, 16–21 June 2013; Volume 3, p. 896.
35. Zheng, M.; You, S.; Huang, L.; Wang, F.; Qian, C.; Xu, C. Simmatch: Semi-supervised learning with similarity matching. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), New Orleans, LA, USA, 18–24 June 2022; pp. 14471–14481.
36. Chen, H.; Tao, R.; Fan, Y.; Wang, Y.; Wang, J.; Schiele, B.; Xie, X.; Raj, B.; Savvides, M. Softmatch: Addressing the quantity-quality trade-off in semi-supervised learning. *arXiv* **2023**, arXiv:2301.10921.
37. Wang, Y.; Chen, H.; Heng, Q.; Hou, W.; Fan, Y.; Wu, Z.; Wang, J.; Savvides, M.; Shinozaki, T.; Raj, B.; et al. Freematch: Self-adaptive thresholding for semi-supervised learning. *arXiv* **2023**, arXiv:2205.07246.
38. Ren, Y.; Yen, A.; Saraj, S.; Li, G.P. Interactive causality-enabled adaptive machine learning in cyber-physical systems: Technology and applications in manufacturing and beyond. In *Principles and Applications of Adaptive Artificial Intelligence*; IGI Global: Hershey, PA, USA, 2024; pp. 179–206.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.