MASTER: Machine Learning-Based Cold Start Latency Prediction Framework in Serverless Edge Computing Environments for Industry 4.0

Muhammed Golec, Sukhpal Singh Gill, Huaming Wu, Talat Cemre Can, Mustafa Golec, Oktay Cetinkaya, Felix Cuadrado, Ajith Kumar Parlikad, and Steve Uhlig

***Abstract*—The integration of serverless edge computing and the Industrial Internet of Things (IIoT) has the potential to optimize industrial production. However, cold start latency is one of the main challenges in this area, resulting in resource waste. To address this issue, we propose a new machine learning-based resource management framework called MASTER which utilizes an extreme gradient boosting (XGBoost) model to predict the cold start latency for Industry 4.0 applications for performance optimization. Furthermore, we created a new cold start dataset using an IIoT scenario (i.e. predictive maintenance) to validate the proposed MASTER framework in serverless edge computing environments. We have evaluated the performance of the MASTER framework using a real-world serverless platform, Google Cloud Platform for single-step prediction (SSP) and multiple-step prediction (MSP) operations and compared it with existing frameworks that used deep deterministic policy gradient (DDPG) and long short-term memory (LSTM) models. The experimental results show that the XGBoost-based resource management framework is the most successful model in predicting cold start with mean absolute percentage error (MAPE) values of 0.23 in SSP and 0.12 in MSP. It has been also identified that the Linear Regression model (utilized in the MASTER framework) has the least computational time (0.03 seconds) as compared to other deep learning and machine learning models considered in this work. Finally, we compare the energy consumption and CO_2 emissions of all models to emphasize resource awareness.**

***Index Terms*—Cold start latency, edge computing, Industry 4.0, predictive maintenance, serverless computing.**

Manuscript received 16 January 2024; revised 25 February 2024; accepted 29 April 2024. Date of publication 2 May 2024; date of current version 23 May 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62071327 and in part by the Tianjin Science and Technology Planning Project under Grant 22ZYYYJC00020. The work of Muhammed Golec was supported by the Ministry of Education of the Turkish Republic. The work of Felix Cuadrado was supported by HE ACES Project under Grant 101093126. Recommended by Lead Guest Editor Yuemin Ding and Guest Editor Kan Yu. *(Corresponding author: Huaming Wu.)*

Muhammed Golec is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, E1 4NS London, U.K., and also with Abdullah Gul University, Kayseri 38080, Türkiye (e-mail: m.golec@qmul.ac.uk).

Sukhpal Singh Gill and Steve Uhlig are with the School of Electronic Engineering and Computer Science, Queen Mary University of London, E1 4NS London, U.K. (e-mail: s.s.gill@qmul.ac.uk; steve.uhlig@qmul.ac.uk).

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Talat Cemre Can is with TFI TAB Food Investments, Istanbul 34349, Türkiye (e-mail: talatcemrecan@gmail.com).

Mustafa Golec is with the Faculty of Engineering Computer Engineering, Dumlupınar University, Kütahya 43100, Türkiye (e-mail: mustafagolec36@gmail.com).

Oktay Cetinkaya is with the Oxford e-Research Centre (OeRC), Department of Engineering Science, University of Oxford, OX1 2JD Oxford, U.K. (e-mail: oktay.cetinkaya@eng.ox.ac.uk).

Felix Cuadrado is with the School of Telecommunications Engineering, Technical University of Madrid (UPM), 43100 Madrid, Spain (e-mail: felix.cuadrado@upm.es).

Ajith Kumar Parlikad is with the Institute for Manufacturing, Department of Engineering, University of Cambridge, CB2 1TN Cambridge, U.K. (e-mail: aknp2@cam.ac.uk).

Data is available online at https://github.com/MuhammedGolec/Cold-Start-Dataset-V2.

Digital Object Identifier 10.1109/JSAS.2024.3396440

I. Introduction

THE rapid developments in sensor technologies have resulted in the spread of the Internet of Things (IoT) applications in many different areas, including civil, military, healthcare, and education [1], [2]. One of the IoT application areas that have attracted attention in recent years is the Industrial Internet of Things (IIoT), which aims to optimize industrial processes and increase efficiency [3]. IIoT allows industrial devices to share data through sensors and networks. Analyzing these data aims to make production processes more efficient [4]. To better understand the impact of IIoT on production processes, predictive maintenance applications in Industry 4.0, which enables the integration of digital technologies into industrial areas, can be given as an example [5]. Predictive maintenance is an application that analyzes data collected through sensors from industrial machines and production processes, offering the following advantages:

1) reducing downtime;
2) increasing the reliability of machines;
3) providing strategies for maintenance [6].

These IIoT-based applications also mean vast amounts of data that must be processed in real time. New developments with low latency and high processing capacity are needed to process these data [7]. Serverless edge computing may be a promising solution to meet this need.

Serverless edge computing is a new paradigm that extends the advantages of serverless computing to the network's edge [2]. This paradigm aims to benefit from the following main advantages of serverless and edge computing [8], [9].

1) *Dynamic Scalability:* System resources can be automatically scaled up or down in line with incoming demands [10]. Thanks to this feature, the system can respond to users quickly, even when transaction demand is high.
2) *Low Latency:* The latency is much lower than on central servers since the data will be processed at the edge [11]. This provides an excellent advantage for real-time operations where response time is critical, such as IIoT and Industry 4.0.
3) *Bandwidth Saving:* Since data are processed at the edge, server usage and network congestion are minimized.
4) *Easy Infrastructure Management:* It encourages code developers to focus only on coding and business logic, abstracting from the control and management level.
5) *Economic Model:* Customers are priced only for the duration of resource usage. This model is known as pay-as-you-go and allows customers to optimize costs.

Besides the advantages of serverless computing, it also has challenges, such as security, privacy, platform dependency, and cold start latency [12]. This article focuses on the cold start problem in serverless edge computing, which can cause latency in real-time IIoT applications [13].

In serverless edge computing, functions are executed by assigning them to containers [14]. After the execution, idle containers are deleted to avoid unnecessary energy and resource consumption. This process is called scale to zero and is the main reason for a cold start [15]. Because the deleted containers may be needed again in line with the increasing demand, and it will take time for these containers to be rebuilt. This preparation time causes a cold start. Another reason for a cold start is when a container receives more requests than it can handle [16]. New containers will be launched to meet this excessive demand, causing cold start delays. Cold start latency has adverse effects, such as user experience, scalability, and cost in serverless edge computing [17].

1) *User Experience:* In scenarios where response time is critical, cold start latency should be minimized for a smooth user experience [18].
2) *Scalability:* One of the essential features of serverless edge computing is its ability to scale resources up and down for variable workloads. An increase in cold start latency will mean an increase in the creation time of containers, so the execution of incoming requests will be delayed, negatively affecting the scalability feature [19].
3) *Cost:* In serverless, with the pay-as-you-go model, only the resources used are charged. In a serverless environment with a high cold start, a short-term and heavily used function will cause unnecessary costs [20].

A. Motivation and Contributions

Industry 4.0 is the fourth industrial revolution that emerged with huge advantages, such as increasing efficiency and product quality by digitizing production processes [21]. Despite these advantages, it brings its own problems, such as investment costs (e.g., new equipment), data security, unsuitability of the existing infrastructure, and latency, which are still waiting to be addressed [22]. Managing latency is critical in real-time IIoT applications in Industry 4.0, as well as automotive and robotic applications [23], as these often require high speed, efficiency, and accuracy. Latencies may delay data processing and bring data integrity in the system [24]. In addition, although Industry 4.0 necessitates quick transactions, latencies may cause undesirable consequences, such as performance degradation and therefore, loss of competitive advantage. The main reasons for latencies in IIoT are 1) constant transfer of data created on IIoT devices to servers for processing (bandwidth waste and network congestion), and 2) time taken for data collected on IIoT devices to return after being processed on the server (response time). The serverless edge computing paradigm can be used to solve these concerns and improve the capabilities of IIoT [25]. Serverless edge computing reduces response time by bringing processing power closer to the network's edge and saves bandwidth because it reduces the data sent to the server. In this way, latency time in IIoT can be reduced [26]. On the other hand, in serverless edge computing, the cold start latency problem caused by the serverless paradigm continues. Few studies have been done in the literature to solve this problem, and most of these studies include solutions, such as "keeping container warm" that require resources to be idle [22]. Therefore, there is a need for approaches that can be the basis for new studies that solve the cold start problem by considering resource consumption.

In this article, we propose a new **MA**chine learning-based cold **ST**art latency prediction framework in S**ER**verless edge computing environments for Industry 4.0, i.e., *MASTER*, to predict the cold start latency in serverless edge computing environments for Industry 4.0 applications to optimize performance. In the MASTER framework, we have utilized two machine learning (ML) models, such as eXtreme gradient boosting (XGBoost) and linear regression (LR), and deep learning (DL) models, such as DeepAR, neural hierarchical interpolation for time series (NHITS), and temporal fusion transformer (TFT). The performance of the MASTER framework is compared with the state-of-the-art frameworks, such as ATOM [27] and two-layer adaptive (TLA) [28], to prove its novelty in predicting the cold start latency. ATOM framework used the deep deterministic policy gradient (DDPG) deep reinforcement learning (DRL) model whereas TLA used the long short-term memory (LSTM) model to predict cold start latency. MASTER used the

abovementioned ML and DL models [29] due to the following reasons.

1) *Capture complex patterns:* It is expected to be successful in nonlinear and complex patterned data, such as cold start.
2) *Automatic feature extraction:* It automatically extracts features in the cold start dataset. This eliminates the need for specialized feature engineering processes.
3) *Robustness against outliers:* It withstands noisy and outliers in cold start datasets and makes accurate predictions.

The main contributions of this work are as follows.

1) Proposing a new ML-based resource management framework called MASTER to predict the cold start latency in serverless edge computing environments. Thus, it is aimed at forming the basis for future resource-sensitive cold start prevention studies.
2) Creating a new cold start dataset based on an IIoT scenario, i.e., predictive maintenance, to validate the proposed MASTER framework in serverless edge computing environments. Thus, a public dataset is created for future cold start studies.
3) Incorporating two ML (XGBoost and LR) and three DL models (DeepAR, NHITS, and TFT) into the MASTER framework to predict the cold start latency, thereby determining the model with the best cold start prediction performance.
4) Comparing the performance of the MASTER framework to those of two baseline works, namely, ATOM [27] and TLA [28], in terms of cold start prediction performance. Thus, it demonstrates the MASTER framework's cold start prediction superiority.
5) Evaluating the performance of the MASTER framework using a real-world serverless platform, the Google Cloud Platform (GCP), for single-step prediction (SSP) and multiple-step prediction (MSP) operations.
6) Comparing the computational time, energy consumption, and CO_2 emission amounts of the abovementioned ML and DL models. In this way, it aims to raise awareness about CO_2 emissions, one of the main causes of global environmental problems.

B. Organization

The rest of this article is organized as follows. Section II discusses the related work of various existing solutions for the cold start problem. Section III presents the methodology, including main architecture, pseudocode, and dataset. Section IV discusses the experimental setup, workload details, evaluation metrics, and results. Finally, Section V concludes this article and highlights future directions.

II. Related Work

Cold start latency originating from the serverless paradigm is still a problem to be solved. The cold start latency can range from tens of milliseconds to a few seconds, causing an undesirable delay in time-sensitive scenarios [30]. When the literature is reviewed, the proposed solutions are generally grouped under two headings [28].

A. Studies on Reducing Cold Start Latency Time

These studies are aimed at making container preparation processes, such as runtime, library initialization, and function preload faster. Thus, the container preparation process takes less time and the cold start latency can be reduced. Solaiman and Adnan [31] aimed to reduce the cold start latency time by proposing a new container management called WLEC. The WLEC management architecture uses S2LRU++, an enhanced version of S2LRU Cache replacement policies. The preparation time is shortened in containers where functions are executed using S2LRU++. The authors tested WLEC on AWS-OpenLambda and a local virtual machine (VM). The results showed that the cold start latency time was reduced by up to 31%. Silva et al. [32] worked with a new technique, they proposed to decide when to create a snapshot in a function. The technique was prototyped using the Linux-based Checkpoint/Restore In Userspace application developer, and experiments were performed by comparing it with standard Unix process creation. Results show that the start-up time of function has improved between 40% and 70%. This way, as the runtime initialization time is shortened, the cold start latency time is also reduced.

B. Studies on Reducing the Frequency of Cold Start

It is about working to reduce the frequency of cold start by using methods, such as keeping the container warm [33]. Suo et al. [34] introduced HotC, a new lightweight container management framework that adjusts the runtime reuse to client requests. In HotC, it performs live container control using the exponential smoothing model and Markov chain models. Moreover, it reuses containers by selecting from the runtime pool according to user requests. Experiments on OpenFaaS show that HotC reduces the frequency of cold starts. Daw et al. [35] aimed to reduce the frequency of cold starts by recommending a tool called Xanadu. Xanadu prevents cold starts by providing speculative and just-in-time resources for serverless platforms. Experiments on Knative and Openwhisk platforms show that Xanadu reduces cold start occurrence by 10–18 times. They aim to reduce the frequency of cold start by using a "hot" container creation technique according to user requests suggested by Ristov et al. [36]. The authors tested their work on the Knative platform using their autoscaler technique, and the results show an 85% success rate. These are existing works such as warm-start containers (WSA) [37], and two-layer adaptive (TLA) [28] methods to reduce the cold start frequency. In the WSA method, authors used a reinforcement learning model to predict call functions and container patterns. In the second stage, the call time of a function is estimated using the LSTM model, and the number of containers to be heated is decided based on this prediction result. Similarly, there is a two-step approach in the TLA method. In the first step, a deep neural network model is used to estimate the number of idle containers (window length). In the second step, the number of requests is estimated using the LSTM model. In the ATOM framework [27], the authors used a DRL method [i.e., (DDPG)], which is effective in solving complex and nonlinear problems, to estimate the number of users using the server and the time of cold start occurrence in serverless

TABLE I
COMPARISON OF THE PROPOSED MASTER FRAMEWORK WITH EXISTING WORKS

Study	Mechanism	Monitoring	Serverless Platform	RA	MSP	Edge	Domain
Studies on Reducing Cold Start Latency Time							
[31]	WLEC	AWS, Local VM	✓	✗	✗	✗	Image Resizing
[32]	Prebaking	Standard Unix	✓	✗	✗	✗	Snapshots
Studies on Reducing the Frequency of Cold Start							
[27]	DRL	GCP Cloud Functions	✓	✓	✗	✓	Healthcare
[37]	WSA	AWS Lambda, Azure, OpenFaaS, Openwhisk	✓	✗	✗	✗	Function Invocation Patterns
[28]	TLA	Openwhisk	✓	✗	✗	✗	Number of Containers
[34]	HotC	OpenFaaS	✗	✗	✗	✗	Container Runtime Pool
[35]	Xanadu	Knative,Openwhisk	✗	✗	✗	✗	Sequence of Functions
[36]	Autoscaler	Knative	✗	✗	✗	✗	Parallel Loops
MASTER	ML and DL	GCP Cloud Functions	✓	✓	✓	✓	Industry 4.0

edge computing. As a result of their experiments, they obtained a root-mean-squared error (RMSE) value of 148.76 for cold start prediction. This framework, unlike previous studies, is a basis for energy-sensitive cold start prevention studies.

C. Critical Analysis

Table I compares the proposed MASTER framework with existing works. The columns in Table I and what they mean can be examined as follows.

1) "Mechanism" represents what techniques were used in the studies reviewed.
2) "Monitoring" represents which platforms/simulators were used in the reviewed studies.
3) "Serverless platform" represents whether a serverless-based platform is used in the studies reviewed.
4) "Resource-aware (RA)" represents whether an RA-based method is used in the studies reviewed.
5) "MSP" represents whether MSP was performed in the studies reviewed.
6) "Edge" represents whether the work under review was tested in an edge environment.
7) "Domain" represents which domain is targeted in the studies reviewed.

Existing methods focus on keeping the container warm and container pooling, which is not RA and requires the constant operation of resources. In addition, in current studies, no dataset has yet been created by considering the dynamically changing "function calls." In addition, none of these studies performed MSP for cold start using ML- and DL-based models. Only two studies use serverless edge computing environments for experiments, namely, ATOM [27] and our proposed framework (MASTER). Compared with ATOM, the MASTER framework provides a huge advantage in cold start detection, such as capturing long-term trends by estimating MSP. Thus, cloud providers can be informed up to 15–20 minutes earlier, and precautions can be taken for a cold start. In addition, the ATOM framework targets the healthcare domain, whereas the Industry 4.0 domain is targeted in MASTER. While the DRL-based algorithm is used to make cold start predictions in the ATOM framework, DL- and ML-based models are used in MASTER, which have advantages, such as capturing complex patterns and automatic feature extraction and also have higher prediction performance. More details on these will be provided in Section IV-E.

III. PROPOSED MASTER FRAMEWORK

In this section, first, the MASTER framework and its working mechanism are described in Section III-A. Then, the methodology is given in Section III-B, so that the reader can better understand the research stages. The datasets used in the article are explained under Section III-C.

A. Main Architecture

The structure of the MASTER framework with four layers is shown in Fig. 1. The first layer consists of assets, the second layer consists of an edge network, the third layer consists of a network, and the fourth layer consists of a serverless platform.

The asset layer forms the first layer in the MASTER framework. In the industry, all machines, sensors, and systems that are included in the production process and monitored in predictive maintenance applications are in this layer. These assets can consist of a variety of equipment, such as computer numerical control machining and heating, ventilation, and air-conditioning systems. The dataset used in the MASTER framework is obtained from a freeze machine [38]. A freeze machine is an industrial machine that can rotate around its own axis and shape various materials, such as metal and furniture, with the help of a cutting edge. Various types and numbers of sensors are used to collect relevant data from the freezing machine.

The edge network is the layer where IoT devices and end nodes, such as programmable logic controller and supervisory control and data acquisition systems, with limited processing powers, are located [39]. It also forms the first of the two main layers in serverless edge computing. This layer has a heterogeneous structure since it accommodates devices with different system features and processing capabilities. The edge network layer is closer to the data center than central servers and therefore can respond to the resource (assets) with lower latency. In the edge network, GCP-based Google Cloud Function is deployed. In this way, nodes can not only control the lifecycle of the function ($f(x)$) but also interact with each other. Edge network transmits $f(x)$ from assets to edge nodes in the edge network using different protocols (hyper-text transfer protocol

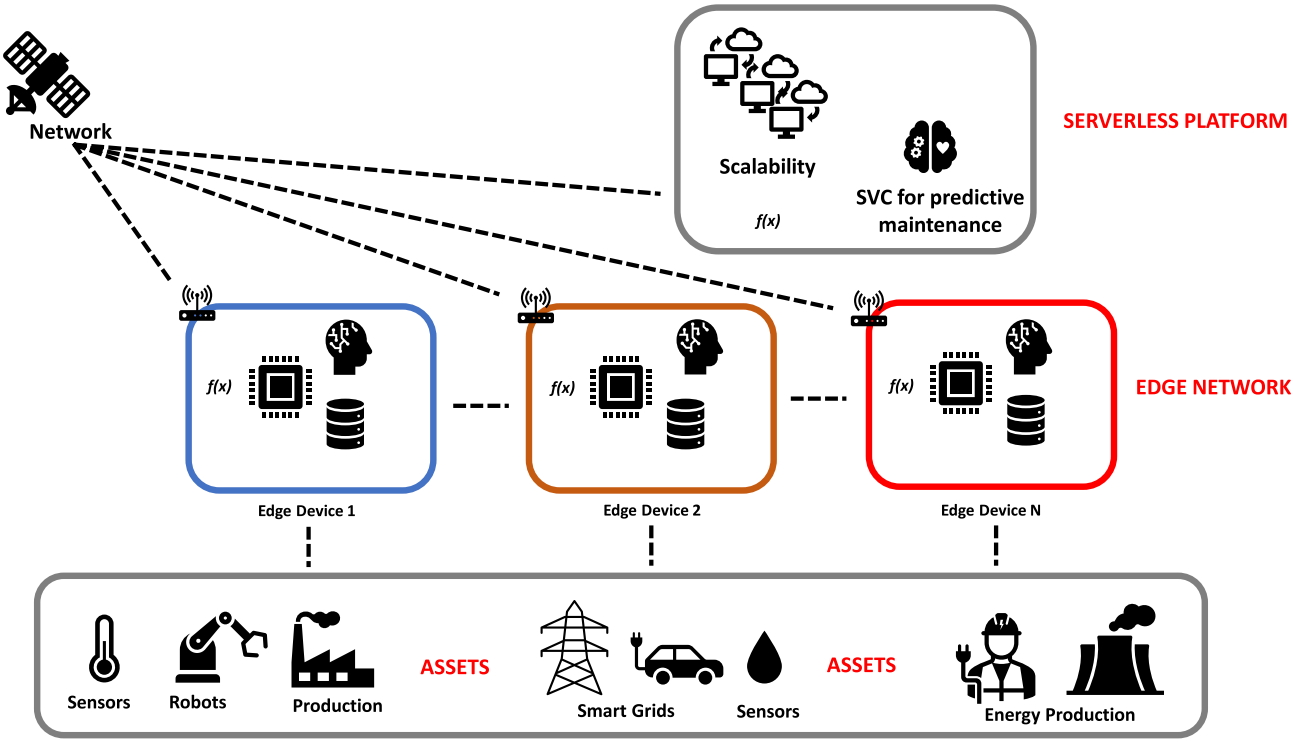

Fig. 1. MASTER framework.

(HTTP) is used in this work) with trigger logic. Edge nodes trigger the function on the serverless platform with this $f(x)$ and return a response to the asset.

Network layer is responsible for all interlayer data communication. It is especially critical in real-time applications. Satellite communications can be used for industrial production lines distributed over large geographies. In addition, the network layer may consist of various network systems, such as intranets, usually wireless. The serverless layer is used when high processing power and capacity are required for $f(x)$ sent from assets to the edge network. This is decided by the edge nodes in the edge network. The MASTER framework constantly monitors the network and detects the occurrence of a cold start. It uses XGBoost regressor and the reason for using an XGBoost regressor is explained in Section IV. The XGBoost model is trained using the cold start dataset. By using the time period in the dataset and the latency amounts corresponding to each time period, the latencies of future time periods are estimated. The MASTER framework has two different prediction modes. The first prediction mode is SSP, which makes a cold start prediction 5 min in advance, and the second prediction mode is MSP mode, which makes a cold start prediction 20 min in advance. Prediction results provide useful information for efforts to reduce cold start latency frequency. These results can be used by cloud providers in the future to reduce cold start frequency and provide smoother operations and cost savings for customers.

In Algorithm 1, the pseudocode of the MASTER framework is given for time-series prediction. The first phase involves creating a cold start dataset. The second stage shows the cold start prediction process of the trained model using this dataset. The MASTER framework is positioned between the client and server to monitor transaction information. Predictive maintenance data ($\psi_{1,2,\ldots,n}$) coming from the sensors, such as air temperature and rotational speed, are sent to the ML model [support vector classifier (SVC)] deployed on the serverless platform. The prediction result (Δ) made in the SVC model is sent back to the client. The MASTER framework saves Δ and transaction information (T_i). In this way, it creates a cold start dataset by monitoring the communication channel 24 h a day and five days a week. In the second stage, the ML model (XGBoost) in the MASTER framework is trained using the cold start dataset. The variables given as input are the amount of delay corresponding to each time period in the cold start dataset (τ), the loss function value used in the XGBoost (XGB) model (ι), the base learner value (g), and the number of subtrees (κ). As output, the model's prediction result for the cold start is returned ($\Re$). In the last part, the cold start in the system can be determined according to a previously determined λ value.

Time complexity: There are two loops in the algorithm, so the time complexity value is $\mathcal{O}(n^2)$. This means that the algorithm performance will deteriorate as the square of the number of elements increases.

B. Methodology

Fig. 2 is designed to better explain the MASTER workflow in technical terms.

1) In the first stage, the cold start dataset containing client–server communication information and the cold start statuses are created. To do this, a predictive maintenance

Algorithm 1: Pseudocode of MASTER for Time-Series Prediction.

Input: $\psi_{1,2,\ldots,n}$, $\tau \in (\tau_1, \tau_2, \ldots, \tau_n)$, ι(y,y'), g(X,μ), κ
Output: Δ, $\Re$
Variables:
Predictive maintenance data $\leftarrow \psi_{1,2,\ldots,n}$
Support Vector Classification $\leftarrow$ SVC
Prediction Result $\leftarrow \Delta$
Transaction Information $\leftarrow T_i$
XGBoost $\leftarrow$ eXtreme Gradient Boosting
Time Period $\leftarrow \tau$
Loss Function $\leftarrow \iota$
Base Learner Value $\leftarrow g$
the Number of Subtrees $\leftarrow \kappa$
Prediction result $\leftarrow \Re$
Threshold Value $\leftarrow \lambda$
Begin
for Day=1:5 do
    Cold Start Dataset Creation
    Send $\psi_{1,2,\ldots,n} \longrightarrow$ Serverless **ML**$(\sum_0^n \psi_{1,2,\ldots,n})$
    Return $\Delta \oplus T_i$
    Save T_i
Cold Start Prediction
for $\tau = 1{:}\kappa$ **do**
    Initialize $g_0(X_i) = \sum_{i=1}^{N} \iota(y_i, \text{p}))$
    Compute $\nabla g_t(\text{X})$
    Start New g g(X,μ)
    $\Re = \arg\min_p \sum_{i=1}^{N} \kappa(y_i, g'_{k-1}(X_i) + \text{pg}X_i, \mu_i)$
    If $\Re > \lambda$:
        Return Cold Start
End

application is deployed on a serverless platform. Then, the system is followed for 24 h a day and five days a week, as explained in the previous section.

2) After the cold start dataset is created, outliers are detected through preprocessing operations. Feature engineering operations, such as the standard scaler and lag features, are performed for artificial intelligence (AI)-based time-series models that will be used in cold start prediction. Our aim in doing this is to increase the prediction accuracy as much as possible.
3) SSP and MSP prediction processes are performed with ML- and DL-based time-series models.
4) In the last step, a performance evaluation for ML and DL models is performed.

C. Dataset

This section describes the two different datasets used in this research work. In particular, we used the predictive maintenance dataset to create the cold start dataset and then used the cold start dataset to train the AI-based time-series models.

1) Predictive Maintenance Dataset: The predictive maintenance dataset used in this article was produced by Matzka [38] and shared via Kaggle.[1] Modeled after a milling machine, this dataset contains 14 features and 10 000 data, and Table II explains what each feature means. Five fault errors in the dataset were added to the Failure Type variable. In addition, the "Machine Failure" variable has been named "Target" for convenience. First, feature engineering operations were performed on the dataset and meaningless data in the "Failure Type" variable was removed. Later, the variables "UDI," "Failure Type," and "Product ID" were removed because they would not be used in this experiment. The categorical variable "Type" was subjected to one-hot encoder processing, and numerical variables "Air temperature," "Process temperature," "Rotational speed," "Torque," and "Tool wear" were subjected to standard scalar processing. The variable "Target" was selected as the target variable. Logistic regression and SVC ML models, which are known to have high prediction performance for predictive

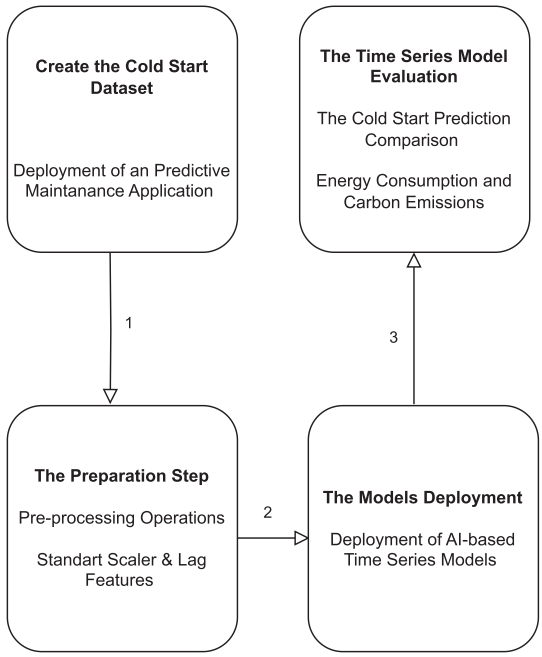

Fig. 2. Flowchart to show the workflow in MASTER.

TABLE II
PREDICTIVE MAINTENANCE DATASET

UID	Unique id numbers (1–1000)	Rotational speed	Rotation speed (in r/min)
Product ID	Item numbers	Torque	Torque value (N·m)
Type	Product quality (L, M, H)	Tool wear	tool wear value (5/3/2 min for H/M/L respectively)
Air temperature	Temperature (2-300 K)	Target	Indicates whether there is a machine malfunction
Process temperature	Process temperature	Machine failure	Shows Machine Failure Type

[1][Online]. Available: https://www.kaggle.com/datasets/stephanmatzka/predictive-maintenance-dataset-ai4i-2020

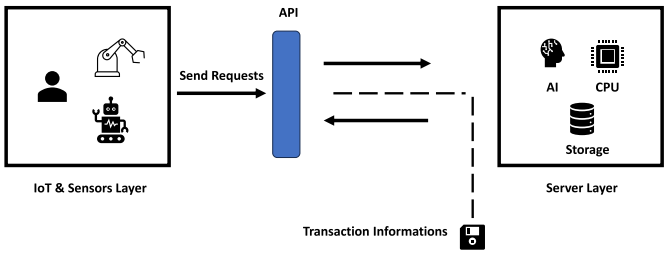

Fig. 3. Cold start dataset creation.

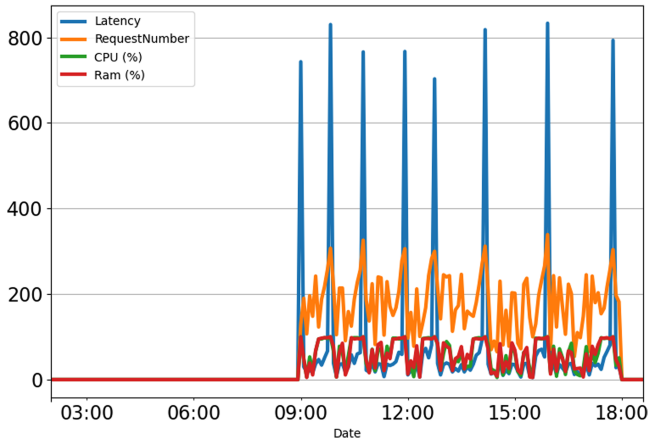

Fig. 4. Coldstart dataset.

maintenance, were compared. It was determined that the model with the highest accuracy rate was SVC with 97.78%.

2) Cold Start Dataset: This section explains how to obtain the cold start dataset that will be used to train the ML/DL model of the MASTER framework. The predictive maintenance scenario described in the previous section was deployed on GCP-Cloud Functions, a serverless platform as shown in Fig. 3.

The environment parameters for this instance are as follows: "Region": Europe-Southwest1-a, "Runtime": Python 3.10, "Function call format": HTTP, and "Memory": 512 MB. To create the workload, a varying number of simultaneous requests (1–350) are sent to the server using the Apache J-Meter application between 1 and 6 January, 2024. To simulate the production process in a factory, the system is set to send requests to the server five days a week between 09:00 and 18:00. Using the HTTP trigger mechanism in J-Meter, six variables are sent to the SVC model deployed on the server to obtain the prediction result and transaction information. Transaction information includes the following data: "Date," "Time," "Day," "Latency," "Request Number," "CPU (%)," and "Ram (%)." Using this information, the cold start dataset shown in Fig. 4 is created. Equation (8) is used to calculate cold start, and when the dataset is examined, it is seen that cold start occurs in the following three ways.

1) When the first request comes to the server, a cold start occurs because the environment parameters are loaded into the container for the first time.
2) *If there is no request to the server for more than 15 min:* In GCP-Cloud Functions, after function execution is completed, containers continue to run for a certain period of time (15 min) [40]. Similar measures are taken on other trading platforms to prevent cold start.
3) *In case more than 300 simultaneous requests are sent to the server:* This number is the threshold required to launch a new container for this scenario.

3) Data Preparation Steps: The following are the data preparation steps.

1) *Standard Scaler:* It scales the features in the dataset and converts it to a dataset with zero mean and unit variance. This aims to improve the performance of ML algorithms, which are affected by the size differences between features in the dataset. In addition, it is to prevent a single feature from dominating the learning process.
2) *Lag Features Class Created:* It shows the values from previous time steps in a time-series dataset. It helps capture the correlation between a variable and the values of past variables. Patterns that are useful in seasonality analysis can be easily detected.
3) *Window Features Class Created:* It calculates summary statistics of historical values. They provide helpful outputs, such as anomaly detection or trend analysis, by presenting information, such as moving averages and rolling standard deviation.
4) *Autocorrelation Function (ACF):* It is used to examine the correlation of a time series with its lagged values. It is generally used to detect seasonality in time series.
5) *Partial autocorrelation function:* It is a statistical tool used to examine the correlation between time series and delays as in ACF. Unlike ACF, it does not include intermediate delays in the correlation analysis.

IV. Performance Evaluation

This section discusses the experimental setup, workload, evaluation metrics, and results. Section IV-A discusses the experimental setup used to conduct experiments. Next, the workloads created throughout this article are introduced in Section IV-B. We discuss the baselines frameworks in Section IV-C, which are used to compare the cold start prediction performance with the proposed MASTER framework. We describe the evaluation metrics used in all performance comparisons in Section IV-D. In Section IV-E, the performance of the proposed MASTER framework is compared experimentally with the abovementioned baseline frameworks in terms of cold start prediction performance, energy consumption, computational time, and carbon emissions.

A. Experimental Setup

In this section, parameter information for all ML- and DL-based models used in the MASTER framework is given along with the system configuration details for the reproduction of this work in the future. All experiments were carried out on a system with "CPU": Intel Core i7-10750H, "clock speed": 2.6–5.0 GHz, "RAM": 16 GB, and "OS": Windows 10 Pro system.

TABLE III
HYPERPARAMETER SETTINGS FOR ML/DL MODELS FOR BOTH PROPOSED (MASTER) AND BASELINE (ATOM AND TLA) FRAMEWORKS

Framework	Model name	Hyperparameters
ATOM [27]	DDPG	Nf = 2, LRa = 0.0001, LRc = 0.01, Nah = 30, Nch = 30, MAXep = 100
TLA [28]	LSTM	epoch=50, activation="softmax", input shape = (10, 1), Dense =1
MASTER	XGB Regressor	"objective": "reg:squarederror", "n_estimators": 100, random_state': 33
	LR	"copy_X": True, "fit_intercept": True, "normalize": "deprecated', "positive": False
	DeepAr	training, learning_rate=0.1, log_interval=10, log_val_interval = 1,hidden_size = 32, rnn_layers = 2, optimizer = "Adam"
	NHITS	training, learning_rate=0.01, log_interval=10, log_val_interval=1, weight_decay=1e-2, backcast_loss_ratio=0.0, hidden_size=64, optimizer="Rprop"
	TFT	training_learning_rate=0.6, hidden_size=32, attention_head_size=2, dropout=0.3, hidden_continuous_size=8, loss=QuantileLoss(), log_interval=10, optimizer='Adadelta', loss='mse'.

TABLE IV
ENVIRONMENT PARAMETERS FOR GOOGLE CLOUD FUNCTIONS

Region	Europe-Southwest1-a
Runtime	Python 3.10
Function call format	HTTP
Memory	512 MB

The hyperparameter settings for all ML/DL models tested in this work are given in Table III. In addition, the environment parameters for Google Cloud Functions used when creating the cold start dataset are given in Table IV.

B. Workloads

One of the biggest obstacles to serverless edge computing and IIoT integration to make industrial processes more efficient is cold start latency. In this work, we propose an ML-based resource management framework called MASTER. In this way, it provides the basis for future cold start prevention studies by performing cold start prediction and monitoring in serverless edge computing environments.

First, the Industry 4.0 scenario, an IIoT application, was deployed using Google Cloud Functions. To create the workload, requests were sent to the server via JMeter, simulating a real industrial production process. This involves sending 1–350 HTTP requests to the server between 09:00 and 18:00 for five days. The cold start dataset was created using the responses and transaction information returned for all requests from the ML model deployed on the server. Second, ML/DL models were trained using this cold start dataset and all models were compared according to SSP and MSP to find the model with the most successful prediction result.

C. Baselines

In this section, we discuss briefly about baselines, which are used to compare the performance of the proposed MASTER framework. In serverless edge computing, each function is assigned to a new container for execution. Setting up environment parameters, such as requires a certain amount of time, which causes cold start latency. A new container is started in the following three cases.

1) When the first request comes to the server.
2) When the container is not used for a certain period of time. Idle containers are released to save energy (zero to scale). If a new request comes to the released container, the container must be restarted.
3) If the number of requests to the container exceeds the capacity of the container, a new container is started.

For this reason, the correlation between cold start occurrence and the number of requests sent to the server can give important clues. Another important correlation information is cold start delay patterns. Because, when delay patterns exceed a certain threshold value, action can be taken to prevent cold start occurrence. In this article, we compare the proposed framework (MASTER) concerning the performance of SSP and MSP with current cold start-based baselines: ATOM [27] and TLA [28].

1) *ATOM [27]:* In the proposed approach, cold start occurrence times and the number of requests to be sent to the server are determined by using a DRL-based model (DDPG). The authors chose a DRL-based model because it has proven to be successful for complex and nonlinear problems. In this way, it is aimed to provide a sustainable solution for future resource-sensitive cold start prevention studies.
2) *TLA [28]:* This approach model has two stages. In the first stage, how much longer the container will be kept warm is calculated using the actor–critic model. In the second layer, call times are determined by monitoring function patterns. By determining the function call times, the heating times of the containers are determined. Thus, cold start latency frequency and duration are tried to be reduced.

D. Evaluation Metrics and Formulations

The metrics and formulations used when evaluating DL and ML models are as follows.

1) *Accuracy rate:* It shows how accurately the ML model predicts [23]. It is obtained by dividing true positive and true negative by the total value. Accuracy is calculated as follows:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TN} + \text{FP} + \text{FN} + \text{TP}}. \quad (1)$$

2) *Precision:* It indicates how many of the samples predicted as positive in the ML model are actually positive [11]. Precision is calculated as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{FP} + \text{TP}}. \quad (2)$$

3) *Recall:* It gives how many of the situations that need to be predicted as positive are predicted positively using the ML model [10]. Recall is calculated as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \quad (3)$$

4) *F-Score:* It is used to find the harmonic mean between precision and recall [23]. F-Score is calculated as follows:

$$F_{\text{Score}} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{4}$$

5) *Mean absolute error (MAE):* It is calculated by averaging the absolute differences between the true value Y and the predicted value Y' [13]. It is another metric used to evaluate forecasting models in statistics

$$\text{MAE} = \frac{1}{N} \sum |Y - Y'|. \tag{5}$$

6) *Mean squared error (mse):* It is calculated by squaring the difference between the actual value Y and the predicted value Y' [11]

$$\text{mse} = \frac{1}{N} \sum (Y - Y')^2. \tag{6}$$

7) *Root-mean-squared error:* It is calculated by taking the square root of the mse [27]. It is used more than mse because the mse value can be very large in some comparison situations:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum (Y - Y')^2}. \tag{7}$$

8) *Cold start:* It originates from the serverless paradigm and is calculated using the formula below. Here, τ_i represents the response time for the first request, and τ_i represents the response time for the second request [8]

$$\zeta = \tau_i - \tau_{ii}. \tag{8}$$

Energy-efficient solutions are needed for edge computing, which has evolved into a net zero emission policy. These solutions contribute to net zero emissions by reducing global electricity use. Using the formulations explained below, energy consumption and CO_2 emissions can be calculated for all AI models examined in this article.

1) *Energy Consumption:* The following formula is used to find the energy consumption E used by the models [41]. Here, $\mathbb{P}$ represents the thermal design power of the processor. t is used to represent both the train and test time of the models

$$E = \mathbb{P} \times \frac{t}{100}. \tag{9}$$

2) *Carbon Emission:* Cloud providers provide services, such as storage and processing power, to users over the Internet through data centers. Operations that require electricity consumption, such as energy and cooling, to provide all these services contribute to carbon emissions [27]. Although calculating the amount of carbon emissions is a complex process, it is generally calculated as follows:

$$C_{\pounds} = P \times t \times C_{\text{IE}} \tag{10}$$

where $C_{\pounds}$ is the amount of carbon emissions, P is the power consumption, t is the train or test time for an AI model, and C_{IE} is the coefficient that varies regionally. In this research work, this coefficient is taken as 182 gCO2/kWh.[2]

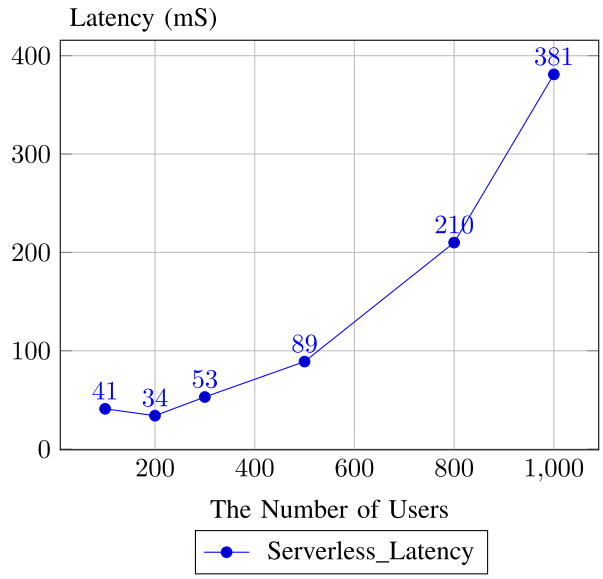

Fig. 5. Performance measurements in terms of latency while deploying the predictive maintenance dataset.

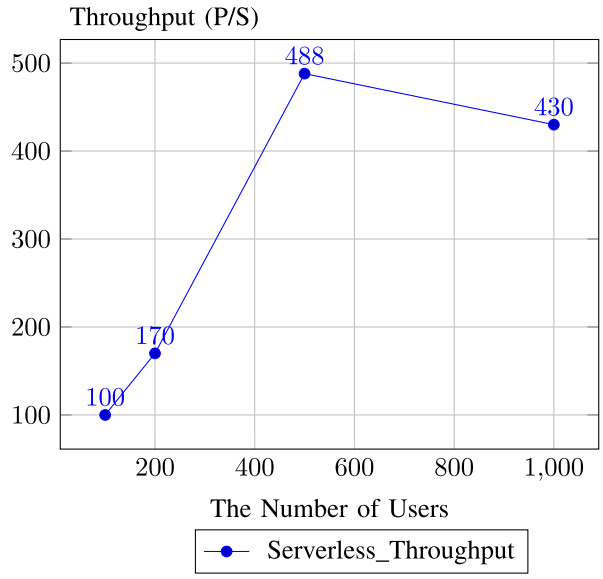

Fig. 6. Performance measurements in terms of throughput while deploying the predictive maintenance dataset.

E. Results

This section discusses the experimental results in terms of serverless platform performance, ML, and computing parameters.

1) Serverless Platform Performance: Figs. 5 and 6 show the latency and throughput values obtained in response to

[2][Online]. Available: https://carbonintensity.org.uk/

TABLE V
SSP PREDICTION PERFORMANCES ON TEST DATA FOR PROPOSED (MASTER) AND BASELINE FRAMEWORKS (ATOM AND TLA)

Work	Model	MAPE	MAE	RMSE	mse
ATOM [27]	DDPG	0.52	65.53	78.43	6151.26
TLA [28]	LSTM	0.48	60.37	83.40	6955.56
MASTER	XGBR	0.23	31.9	34.43	1185.54
	LR	0.45	60.74	68.35	4672.94
	NHITS	0.61	84	100.62	10126.37
	TFT	0.78	102.36	121.67	14804.22
	DeepAr	0.81	111.22	112.37	12627.87

TABLE VI
MSP PREDICTION PERFORMANCES ON TEST DATA FOR PROPOSED (MASTER) AND BASELINE FRAMEWORKS (ATOM AND TLA)

Work	Model	MAPE	MAE	RMSE	mse
ATOM [27]	DDPG	0.47	189.40	225.60	50895.36
TLA [28]	LSTM	0.33	170.66	257.43	66270.20
MASTER	XGBR	0.12	47.75	6.76	45.69
	LR	0.40	136.25	11.59	134.32
	NHITS	0.58	259.30	372.36	138614
	TFT	0.86	365.67	467.77	218808
	DeepAr	0.77	332.95	428.20	183355

the increasing number of users in Google Cloud Functions, respectively. Apache J-Meter application was used to create a workload on the server. The throughput value tends to increase in proportion to the increasing number of users. After the number of users reaches 500, the throughput starts to decrease gradually. The reason for this is the resource contention that occurs due to the use of common resources on the servers. Likewise, the amount of latency is expected to increase depending on the number of users. However, when Fig. 5 is carefully examined, it is seen that the latency of 100 users is higher than the latency of 200 users. This is because of cold start occurring on serverless platforms.

2) ML Parameters—Cold Start Prediction Performance: We have considered cold start prediction performance as an ML parameter. To measure the cold start prediction performance, we consider the latency and throughput of Google Cloud Functions, which are measured for an increasing number of requests. Then, the SSP and MSP performances of the five ML/DL models within the MASTER framework are compared to choose the best-performing model for cold start prediction. Then, the superiority of MASTER is demonstrated by comparing the SSP and MSP performances of the MASTER framework with two baselines [27], [28]. In the next experiment, the computational time of the MASTER framework is compared with baselines. In the last experiment, we evaluate the energy consumption and CO_2 emissions for proposed and existing frameworks.

Two different prediction processes were performed to find the model that made the most successful cold start estimation among all DL/ML models examined in this article. In the first prediction model, SSP, the cold start occurrence time is predicted five minutes in advance by monitoring the past 300 steps. Performance results for all models are given in Table V. The results show that the best model in SSP is the XGB regressor with a mean absolute percentage error (MAPE) ratio of 0.23. In the second prediction model, MSP, the cold start occurrence time was predicted 20 min in advance by monitoring the past 300 steps. Performance results for all models are given in Table VI. The results show that the best model in MSP is again the XGB regressor with a MAPE ratio of 0.12. Both SSP and MSP results show that the MASTER framework is more successful in cold start prediction than ATOM and TLA. Furthermore, it has been identified that the ML models performed much better than DL and DRL models due to the following reasons.

1) The size of the cold start dataset is small. DRL and DL models generally learn better on large datasets. Complex DRL and DL models do not perform well on small-size datasets.
2) DL models are more sensitive to the quality of data than ML models. Therefore, ML models perform better on datasets containing noisy data, such as the cold start dataset generated.

Fig. 7 shows the actual values for the cold start dataset and the prediction results of all DL/ML models.

3) Computing Parameters: We have considered computational time and energy consumption and carbon emissions as computing parameters.

1) Computational Time: It is very important to measure the computational time for ML/DL models with resource limitations. Fig. 8 shows the computational time for these models. It has been noted that the LR is the fastest model with 0.04 s for MSP and 0.017 s for SSP. The slowest model is the NHITS model with 3.16 s on the MSP and 0.79 s on the SSP. When compared in terms of training times, it is noted that the slowest model is the DRL model (DDPG) used in the ATOM framework. This is due to the exploration versus exploitation tradeoff for the DRL agent to learn, which means the agent has to make a lot of attempts to understand the environment and maximize the reward. The results showed that ML models are preferable in terms of practicality.

2) Energy Consumption and Carbon Emissions: In this section, energy consumption and CO_2 emission amounts are compared for MASTER with baselines [27], [28]. In this period, when concerns about environmental sustainability increase, it is of great importance to reduce operational costs by minimizing the carbon footprint. By carrying out these experiments, we aim to identify the most efficient models by emphasizing this awareness.

We compared the MASTER with baselines in terms of energy consumption, which is calculated using (9), as shown in Fig. 9. It has been noted that the LR consumes the least energy for training with 45 J, while the DDPG model consumes the most energy with 20 264 J. In DRL models, such as DDPF, agents learn by trial and error using an exploration and exploitation strategy. This form of learning takes longer than other models (ML and DL) and therefore results in higher energy consumption. The LR model uses the method of linearly relating input variables,

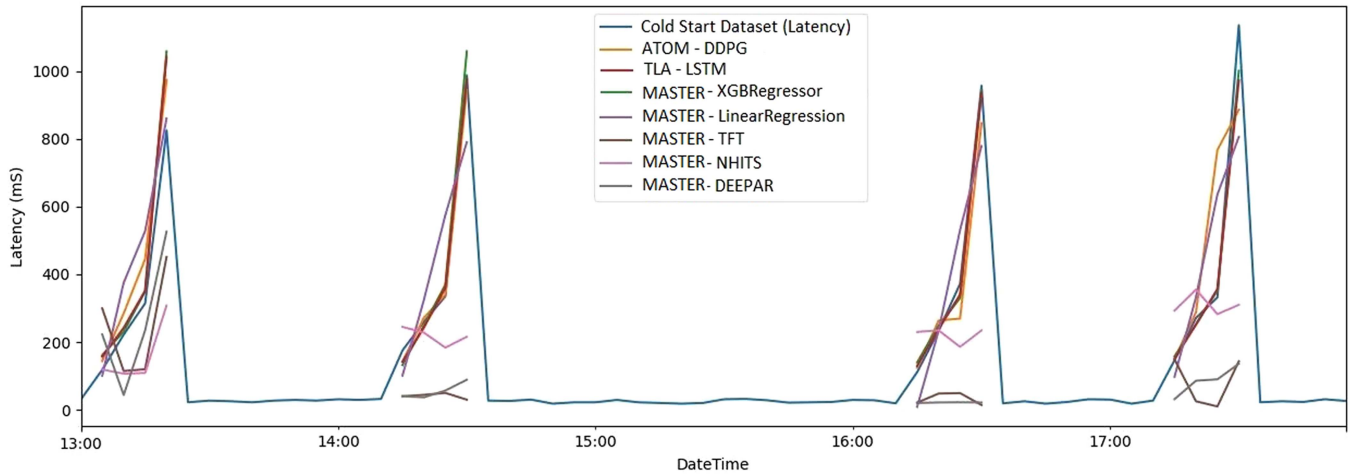

Fig. 7. Cold start prediction performance comparison in terms of latency for proposed (MASTER) and baseline frameworks (ATOM and TLA).

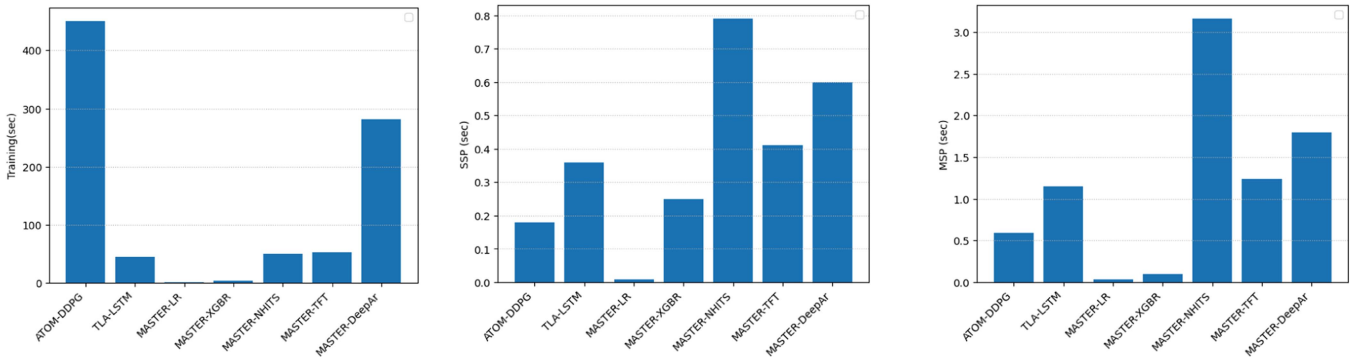

Fig. 8. Performance comparison in terms of latency for proposed (MASTER) and baseline frameworks (ATOM and TLA) in terms of computation time. (a) Training. (b) SSP. (c) MSP.

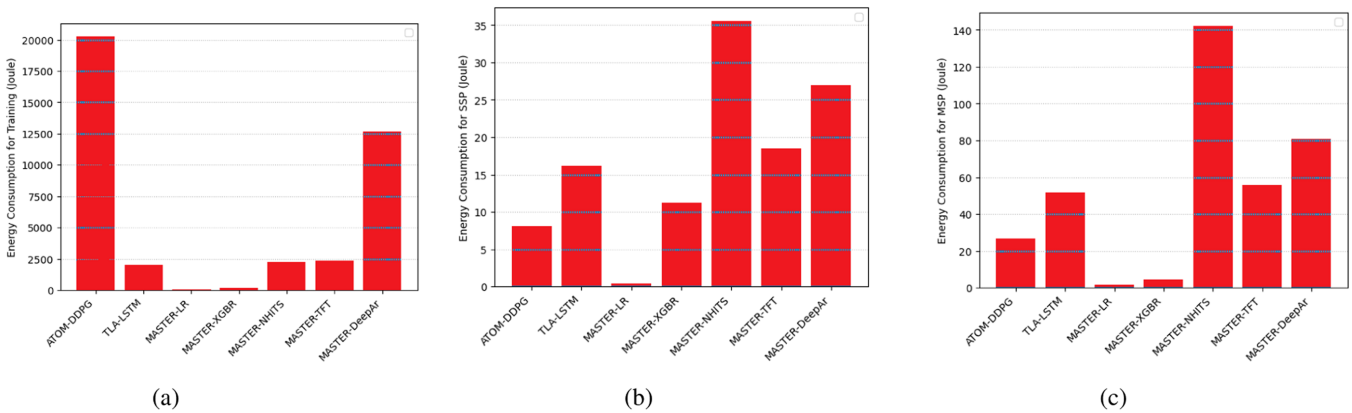

Fig. 9. Performance comparison in terms of latency for proposed (MASTER) and baseline frameworks (ATOM and TLA) in terms of energy consumption. (a) Training. (b) SSP. (c) MSP.

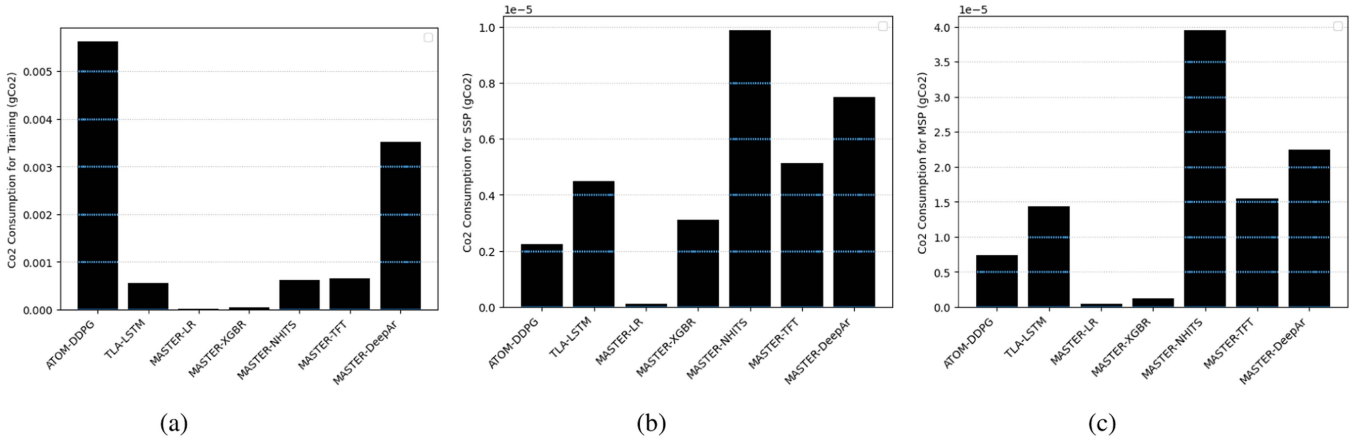

Fig. 10. Performance comparison in terms of latency for proposed (MASTER) and baseline frameworks (ATOM and TLA) in terms of CO_2 emission. (a) Training. (b) SSP. (c) MSP.

which is an uncomplicated learning model. In addition, it has a small number of hyperparameters. For all these reasons, it is faster and consumes less energy than other models. When looking at the SSP and MSP results, it is seen that the model that consumes the least energy is LR for this reason (0.45, 1.8), while the model that consumes the most energy is NHITS with values of 35.55 and 142.20 J. NHITS has an architecture consisting of several stacks and blocks to eliminate long-horizon forecasting and computational complexity, and therefore will bring higher energy consumption compared with other models.

Fig. 10 shows the emission amounts of CO_2 obtained using (10). Since the amount of CO_2 emission is directly proportional to the amount of energy consumption, similarly, for training, the least CO_2 emission belongs to the LR model with 1.25e-05 gCO_2 and the highest CO_2 emission belongs to the DDPG model with 0.005 gCO_2. For SSP and MSP, the lowest CO_2 emissions belongs to the LR model with 5e-07 and 1.25e-07 gCO_2, while the highest CO_2 emissions belongs to NHITS model with 9.87e-06 and 3.95e-05 gCO_2.

V. Conclusion

The integration of serverless edge computing and the IIoT is a promising approach that can make industrial processes more efficient in addition to the advantages that the serverless paradigm offers, such as an affordable pricing model and dynamic scalability, there is still a cold start latency problem waiting to be solved. This article explores the potential of AI models for predicting cold start latency. For this, we propose MASTER, an ML-based framework that performs cold start monitoring and prediction in serverless edge computing environments. The MASTER framework is positioned between the client and server, monitors all communication information, and creates a cold start dataset. It trains the ML algorithm in the MASTER framework using this cold start dataset. To evaluate the performance of the MASTER framework, we used the predictive maintenance application, which is an Industry 4.0 scenario. As a result of the experiments performed for the model to be used in the AI module of the MASTER framework, it was determined that the most successful model was XGBoost, with MAPE values of 0.23 in SSP and 0.12 in MSP. We also compared the performance of the MASTER framework in terms of cold start latency prediction with baselines, such as ATOM and TLA. In this article, the performance of time-series models was compared according to energy consumption and CO_2 emissions. The results showed that NHITS was the model with the highest computation time and CO_2 emissions.

This article demonstrates that ML models can accurately forecast cold start latency and hence hold significant promise for reducing cold start latency in the future. Further, additional resource management issues in serverless computing, such as execution cost and scalability, can be constructively addressed by extending generative AI models in future research. Cold start prediction accuracy can be enhanced using modern ML or DL models. It is also possible to mitigate the cold start latency problem in serverless settings by making extensions to the MASTER framework. Furthermore, public datasets containing multiple applications and functions offered by cloud service providers, such as Microsoft Azure, can be used for real-time predictions. As a result, many functions can be used to create a dataset, which can be utilized in the future. In addition, in settings with limited resources, a faster and less expensive system can be developed with the help of online ML.

Software Availability: The dataset in first footnote is publicly published for future researchers.

References

[1] G. Liu, "Frequency-switchable routing protocol for dynamic magnetic induction-based wireless underground sensor networks," *IEEE J. Sel. Areas Sensors*, vol. 1, pp. 1–8, 2024.

[2] X. Li, P. Russell, C. Mladin, and C. Wang, "Blockchain-enabled applications in next-generation wireless systems: Challenges and opportunities," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 86–95, Apr. 2021.

[3] H. Boyes, B. Hallaq, J. Cunningham, and T. Watson, "The Industrial Internet of Things (IIoT): An analysis framework," *Comput. Ind.*, vol. 101, pp. 1–12, 2018.

[4] K. Rose, S. Eldridge, and L. Chapin, "The Internet of Things: An overview," *The Internet Soc.*, vol. 80, pp. 1–50, 2015.

[5] S. S. Gill et al., "Modern computing: Vision and challenges," *Telematics Inform. Rep.*, vol. 13, 2024, Art. no. 100116.

[6] T. Zonta, C. A. Da Costa, R. da Rosa Righi, M. J. de Lima, E. S. da Trindade, and G. P. Li, "Predictive maintenance in the Industry 4.0: A systematic literature review," *Comput. Ind. Eng.*, vol. 150, 2020, Art. no. 106889.

[7] S. S. Gill, I. Chana, M. Singh, and R. Buyya, "RADAR: Self-configuring and self-healing in resource management for enhancing quality of cloud services," *Concurrency Computation: Pract. Experience*, vol. 31, no. 1, 2019, Art. no. e4834.

[8] M. Golec, G. K. Walia, M. Kumar, F. Cuadrado, S. S. Gill, and S. Uhlig, "Cold start latency in serverless computing: A systematic review, taxonomy, and future directions," 2023, *arXiv:2310.08437*.

[9] C. Tang, G. Yan, H. Wu, and C. Zhu, "Computation offloading and resource allocation in failure-aware vehicular edge computing," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 1877–1888, Feb. 2024.

[10] M. Golec, D. Chowdhury, S. Jaglan, S. S. Gill, and S. Uhlig, "AIBLOCK: Blockchain based lightweight framework for serverless computing using AI," in *Proc. IEEE Int. Symp. Cluster Cloud Internet Comput.*, 2022, pp. 886–892.

[11] M. Golec, S. Iftikhar, P. Prabhakaran, S. S. Gill, and S. Uhlig, "QoS analysis for serverless computing using machine learning," in *Serverless Computing: Principles and Paradigms*. Berlin, Germany: Springer, 2023, pp. 175–192.

[12] X. Liu et al., "FaaSLight: General application-level cold-start latency optimization for function-as-a-service in serverless computing," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, Jul. 2023, Art. no. 119.

[13] M. Golec, S. S. Gill, A. K. Parlikad, and S. Uhlig, "HealthFaaS: Ai-based smart healthcare system for heart patients using serverless computing," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 18469–18476, Nov. 2023.

[14] I. Baldini et al., "Serverless computing: Current trends and open problems," in *Research Advances in Cloud Computing*, Berlin, Germany: Springer, 2017, pp. 1–20.

[15] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," *Commun. ACM*, vol. 62, no. 12, pp. 44–54, 2019.

[16] J. M. Hellerstein et al., "Serverless computing: One step forward, two steps back," 2018, *arXiv:1812.03651*.

[17] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *Proc. IEEE 11th Int. Conf. Cloud Comput.*, 2018, pp. 442–450.

[18] P. K. Gadepalli, G. Peach, L. Cherkasova, R. Aitken, and G. Parmer, "Challenges and opportunities for efficient serverless computing at the edge," in *Proc. IEEE 38th Symp. Reliable Distrib. Syst.*, 2019, pp. 261–2615.

[19] M. Sewak and S. Singh, "Winning in the era of serverless computing and function as a service," in *Proc. IEEE 3rd Int. Conf. Convergence Technol.*, 2018, pp. 1–5.

[20] T. Elgamal, "Costless: Optimizing cost of serverless computing through function fusion and placement," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2018, pp. 300–312.

[21] Y. K. Teoh, S. S. Gill, and A. K. Parlikad, "IoT and fog-computing-based predictive maintenance model for effective asset management in Industry 4.0 using machine learning," *IEEE Internet Things J.*, vol. 10, no. 3, pp. 2087–2094, Feb. 2023.

[22] M. Shurrab, D. Mahboobeh, R. Mizouni, S. Singh, and H. Otrok, "Overcoming cold start and sensor bias: A deep learning-based framework for IoT-enabled monitoring applications," *J. Netw. Comput. Appl.*, vol. 222, 2024, Art. no. 103794.

[23] M. Golec, R. Ozturac, Z. Pooranian, S. S. Gill, and R. Buyya, "IFaaSBus: A security-and privacy-based lightweight framework for serverless computing using IoT and machine learning," *IEEE Trans. Ind. Inform.*, vol. 18, no. 5, pp. 3522–3529, May 2022.

[24] M. Javaid, A. Haleem, R. P. Singh, S. Rab, and R. Suman, "Significance of sensors for Industry 4.0: Roles, capabilities, and applications," *Sensors Int.*, vol. 2, 2021, Art. no. 100110.

[25] S. S. Gill et al., "AI for next generation computing: Emerging trends and future directions," *Internet Things*, vol. 19, 2022, Art. no. 100514.

[26] Z. Jan et al., "Artificial intelligence for Industry 4.0: Systematic review of applications, challenges, and opportunities," *Expert Syst. With Appl.*, vol. 216, 2023, Art. no. 119456.

[27] M. Golec et al., "ATOM: Ai-powered sustainable resource management for serverless edge computing environments," *IEEE Trans. Sustain. Comput.*, early access, Nov. 29, 2023, doi: 10.1109/TSUSC.2023.3348157.

[28] P. Vahidinia, B. Farahani, and F. S. Aliee, "Mitigating cold start problem in serverless computing: A reinforcement learning approach," *IEEE Internet Things J.*, vol. 10, no. 5, pp. 3917–3927, Mar. 2023.

[29] S. F. Ahmed et al., "Deep learning modelling techniques: Current progress, applications, advantages, and challenges," *Artif. Intell. Rev.*, pp. 1–97, 2023.

[30] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "Serverless programming (function as a service)," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 2658–2659.

[31] K. Solaiman and M. A. Adnan, "WLEC: A not so cold architecture to mitigate cold start problem in serverless computing," in *Proc. IEEE Int. Conf. Cloud Eng.*, 2020, pp. 144–153.

[32] P. Silva, D. Fireman, and T. E. Pereira, "Prebaking functions to warm the serverless cold start," in *Proc. 21st Int. Middleware Conf.*, 2020, pp. 1–13.

[33] S. Pan, H. Zhao, Z. Cai, D. Li, R. Ma, and H. Guan, "Sustainable serverless computing with cold-start optimization and automatic workflow resource scheduling," *IEEE Trans. Sustain. Comput.*, early access, Sep. 01, 2023, doi: 10.1109/TSUSC.2023.3311197.

[34] K. Suo, J. Son, D. Cheng, W. Chen, and S. Baidya, "Tackling cold start of serverless applications by efficient and adaptive container runtime reusing," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2021, pp. 433–443.

[35] N. Daw, U. Bellur, and P. Kulkarni, "Xanadu: Mitigating cascading cold starts in serverless function chain deployments," in *Proc. 21st Int. Middleware Conf.*, 2020, pp. 356–370.

[36] S. Ristov, C. Hollaus, and M. Hautz, "Colder than the warm start and warmer than the cold start! Experience the spawn start in FaaS providers," in *Proc. Workshop Adv. tools, Program. languages, PLatforms Implementing Evaluating Algorithms Distrib. Syst.*, 2022, pp. 35–39.

[37] A. Kumari, B. Sahoo, and R. K. Behera, "Mitigating cold-start delay using warm-start containers in serverless platform," in *Proc. IEEE 19th India Council Int. Conf.*, 2022, pp. 1–6.

[38] S. Matzka, "Explainable artificial intelligence for predictive maintenance applications," in *Proc. 3rd Int. Conf. Artif. Intell. Industries*, 2020, pp. 69–74.

[39] H.-H. Hsu et al., "A nonvolatile AI-edge processor with SLC–MLC hybrid ReRAM compute-in-memory macro using current-voltage-hybrid readout scheme," *IEEE J. Solid-State Circuits*, vol. 59, no. 1, pp. 116–127, Jan. 2024.

[40] P. Vahidinia, B. Farahani, and F. S. Aliee, "Cold start in serverless computing: Current trends and mitigation strategies," in *Proc. IEEE Int. Conf. Omni-Layer Intell. Syst.*, 2020, pp. 1–7.

[41] E. Kristianto, P.-C. Lin, and R.-H. Hwang, "Sustainable and lightweight domain-based intrusion detection system for in-vehicle network," *Sustain. Comput.: Inform. Syst.*, vol. 41, 2024, Art. no. 100936.