ATOM: AI-Powered Sustainable Resource Management for Serverless Edge Computing Environments

Muhammed Golec, Sukhpal Singh Gill, Felix Cuadrado, Ajith Kumar Parlikad, Minxian Xu, *Member, IEEE*, Huaming Wu, *Senior Member, IEEE*, and Steve Uhlig

***Abstract*—Serverless edge computing decreases unnecessary resource usage on end devices with limited processing power and storage capacity. Despite its benefits, serverless edge computing's zero scalability is the major source of the cold start delay, which is yet unsolved. This latency is unacceptable for time-sensitive Internet of Things (IoT) applications like autonomous cars. Most existing approaches need containers to idle and use extra computing resources. Edge devices have fewer resources than cloud-based systems, requiring new sustainable solutions. Therefore, we propose an AI-powered, sustainable resource management framework called ATOM for serverless edge computing. ATOM utilizes a deep reinforcement learning model to predict exactly when cold start latency will happen. We create a cold start dataset using a heart disease risk scenario and deploy using Google Cloud Functions. To demonstrate the superiority of ATOM, its performance is compared with two different baselines, which use the warm-start containers and a two-layer adaptive approach. The experimental results showed that although the ATOM required more calculation time of 118.76 seconds, it performed better in predicting cold start than baseline models with an RMSE ratio of 148.76. Additionally, the energy consumption and CO_2 emission amount of these models are evaluated and compared for the training and prediction phases.**

***Index Terms*—Serverless edge computing, cold start, Internet of Things, deep reinforcement learning, sustainable resource management.**

I. Introduction

THE proliferation of Internet of Things (IoT) applications has led to a substantial increase in the volume of data that requires processing [1]. IoT devices are inherently resource-constrained, and as such, cloud systems can address this limitation by providing the resources and processing power they offer. In cloud systems, data processing occurs exclusively on central servers, leading to issues such as bandwidth congestion and latency. For this reason, the concept of edge computing has emerged to address these challenges by conducting computations at the edge of the network [2]. In edge computing, the computing process is performed at the edge of the network using devices ranging from smartphones to programmable logic controllers (PLCs) [3]. Since the data produced by IoT devices is processed at the edge of the network, it provides great advantages in real-time applications (e.g., augmented reality) where latency is critical [4], [5]. Additionally, thanks to edge computing, bandwidth can be used more efficiently by reducing the amount of data that needs to be transmitted to the cloud.

A. Limitations and Challenges

In addition to the advantages edge computing provides, it has limited processing capacity compared to the traditional cloud and consists of heterogeneous devices [6]. For this reason, it brings challenges such as scaling resources when necessary and efficient resource management [4]. To overcome these challenges, serverless edge computing, a new concept that integrates edge and serverless computing, has emerged [5]. Serverless edge computing extends the advantages of edge computing and brings the advantages of the serverless paradigm such as dynamic scalability and efficient resource consumption [7]. In serverless edge computing, which works with event-driven architecture, containers are used to perform functions, and idle containers are scaled to zero, preventing unnecessary resource and energy consumption [8]. However, despite these advantages, the zero scaling feature remains the root cause of cold start lag that remains unresolved. Because it will take time for a stopped container to be restarted if needed, and this time is called cold

Manuscript received 21 September 2023; revised 7 November 2023; accepted 26 December 2023. Date of publication 29 December 2023; date of current version 10 December 2024. The work of Muhammed Golec would express his thanks to the Ministry of Education of the Turkish Republic, for their support and funding. This work was supported in part by the National Natural Science Foundation of China under Grants 62102408 and 62071327, in part by Shenzhen Science and Technology Program under Grant RCBS20210609104609044, in part by the Chinese Academy of Sciences President's International Fellowship Initiative under Grant 2023VTC0006, and in part by Tianjin Science and Technology Planning Project under Grant 22ZYYYJC00020. Recommended for acceptance by Zhongwen Guo, Hui Xia, Yu Wang, and Radhouane Chouchane. *(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 Computer Engineering Department, Abdullah Gul University, 38080 Kayseri, Turkey (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).

Felix Cuadrado is with the Technical University of Madrid (UPM), 28040 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).

Minxian Xu is with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China (e-mail: mx.xu@siat.ac.cn).

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

The dataset is publicly published for future researchers at: https://github.com/MuhammedGolec/ColdStart-Dataset.

Digital Object Identifier 10.1109/TSUSC.2023.3348157

start latency. Cold start latency is contrary to the low latency principle promised by edge computing. Another reason for the cold start is that the number of requests to the container exceeds the maximum number of requests the container can handle [9]. Serverless creates new containers to handle these requests; thus, new cold start latencies occur. In addition, it has been observed that the amount of CPU and RAM of the system and the software languages used to execute the function affect cold start latency [10].

The cold start problem is an undesirable issue in serverless edge computing. A cold start has adverse effects on performance and user experience, resource usage, and scalability, defined as follows:

- *Performance and User Experience:* Cold start negatively affects a serverless edge system's low latency and fast response capacity. In addition, cold start-based delays will undermine the customer's trust in serverless edge systems.
- *Resource Usage:* Since serverless functions (or containers) work with event-driven architecture, they are started by a trigger. For each startup process, resources like memory and CPU are consumed. Cold starts cause unnecessary resource consumption during these startups.
- *Scalability:* Edge devices connected to serverless edge computing have a heterogeneous structure. Therefore, each device may have a different processing capacity and storage amount. A cold start will negatively affect scalability by limiting the responsiveness of a serverless edge system in the event of a sudden increase in demand.

B. Motivation and Contributions

Serverless edge computing is a new paradigm that promises great promise in industry and academia with the advantages it offers. However, it brings some challenges that still need to be solved such as security, privacy, and resource management [11]. Cold start is one of the most important of these problems. Warm-up and function caching methods are generally used when the studies for the cold start in the literature are examined. In the warm-up methods, containers are kept running continuously using automatic triggers, thus eliminating the possibility of a cold start when a new request comes. In function caching, on the other hand, since frequently used functions are kept ready in the cache, it is tried to prevent a cold start originating from these frequently used functions. However, warm containers and function caching-based work can cause unnecessary resource and energy waste, requiring containers to run continuously to prevent a cold start. For this reason, new studies still need to consider resource and energy consumption while reducing cold start latency. Literature reported [4], [12], [13], [14] that the recent Artificial Intelligence (AI) models can be used to solve this problem to large extent. For example, Deep Reinforcement Learning (DRL) can be inspiring for solving complex and non-linear problems such as cold start latency as identified above. In serverless computing, workloads change dynamically and are unpredictable. AI can be used to optimize serverless environment resources and energy consumption using historical workloads and real-time data. Serverless platform providers can make efficient resource allocation decisions by taking AI prediction results into account when planning intensive workloads.

In this paper, we propose a new **AI**-powered sus**T**ainable res**O**urce **M**anagement framework for serverless edge computing environments called **ATOM**, for optimization of cold start latency and energy consumption. It estimates cold start latency, a persistent issue in serverless edge computing, and establishes the groundwork for future efforts in mitigating cold start concerns without necessitating excessive resource consumption. We used the latest DRL models, such as the Deep Deterministic Policy Gradient (DDPG) and Recurrent Deterministic Policy Gradient (RDPG), in this research because these two models have been proven to be successful in solving complex and nonlinear problems using Deep Learning (DL)'s perceptual ability and Reinforcement Learning (RL)'s decision-making ability [15]. In addition, the potential of DRL in determining the cold start problem is undiscovered despite the fact that it is a promising approach that has never been used for this problem to our knowledge. The main contributions of this paper can be listed as follows:

- We propose an AI-based framework called ATOM that predicts cold start latency in serverless edge computing using DRL techniques.
- We create a cold start dataset used in this study by following a serverless system. For this, a heart disease risk scenario integrated into a serverless platform was deployed using Google Cloud functions.
- We evaluate the performance of DRL used in the ATOM framework and compare its performance with two baselines that used DL techniques using RMSE, MAE, and R2 Score metrics to find the best prediction model from two different DRL techniques.
- We compare the performance of the ATOM framework with baselines based on energy cost to find the most energy-efficient model.
- We investigate the potential of DRL algorithms for resource management problems in serverless edge computing.

C. Organization

The rest of this paper is organized as follows. In Section II, literature studies used to solve the cold start problem are examined in detail. In Sections III and IV, the mechanism of the proposed system is explained by giving background information to the reader. Performance evaluation is made by giving the experimental results of the models presented in the study in Section V. In Section VI the paper is concluded by summarizing the conclusion and future work.

II. Related Work

It has been previously explained that the cold start that occurs in serverless edge computing is due to the scaling to zero feature arising from the serverless paradigm. Therefore, studies on the cold start problem in serverless computing can also be used for serverless edge computing. Under this subheading, all cold start solutions for serverless and serverless edge computing are discussed. When the literature is examined, it is seen that the studies on solving the cold start problem in the serverless paradigm are generally grouped under two subheadings. These are the reduction of cold start latency and the reduction of cold start occurrence frequency.

A. Cold Start Latency Reduction (LR)

There are many studies aimed at reducing the latency caused by cold starts through the acceleration of preparation stages for the containers responsible for function execution.

Akkus et al. [16] presented a new method of creating sandboxes with their proposed SAND system. This system uses two main features. These are application-level logical space, which reduces container provisioning time, and the hierarchical bus, which allows functions running on the same source to run faster. The results show that SAND reduces cold start latency 43% more successfully than Apache OpenWhisk. Liu et al. [22] aimed to reduce the cold start latency time by applying application-level optimization in their study, FaaSLight. As a result of their tests, they determined that the application code loading process has the most significant effect on the cold start delay. They suggested separating optional code from FaaS to mitigate this effect by installing only the necessary code. The results show that the cold start latency can be reduced by up to 21.25 times. The serverless paradigm has a dynamic and multi-tenant structure. This structure causes challenges such as resource contention and efficient resource management. Mampa et al. [23] introduced an efficient function scheduling mechanism employing a DRL-based technique. They conducted performance tests in the Kubeless environment and the results showed noticeable improvements in response time and resource usage cost. The other approach is automatic scalability-based approaches. Thanks to automatic scalability, sufficient resources can be allocated to containers to cope with sudden fluctuations in the number of requests and reduce the effect of cold start latency. Agarwal et al. [17] proposed a Long Short-Term Memory-Proximal Policy Optimization (LSTM-PPO) method-based approach and presented an efficient autoscaling mechanism for serverless environments. They deploy the proposed agent to OpenFaaS and conduct experiments using the matrix multiplication function. The results show that the proposed approach increases throughput by up to 18%. Mampage et al. [18] proposed a DRL-based algorithm, providing optimized scaling in serverless computing. To do this, they adapted the Asynchronous Advantage Actor Critic (A3C) model to applications on the platform for horizontal and vertical scaling decisions. The performance of the proposed algorithm is evaluated using a Kubeless-based testing environment. The authors found reductions in latency of up to 34%.

B. Cold Start Occurrence Frequency Reduction (FR)

Generally, there are two ways to decrease the occurrence of cold starts on the serverless platform: container preparation and Keeping the container warm.

1) Container Preparation (FRP): The main purpose of container preparation approaches is to run the container before a function request comes to the serverless. In this way, it is aimed to prevent the formation of cold start.

Vahidinia et al. [24] proposed a new two-layer adaptive approach that prevents cold start. The first layer explores patterns of calling functions and idle containers with an RL-based model. In the second layer, the next run time of the function is determined with the LSTM model, and the number of preheated containers is decided. The results show that the proposed operation reduces the cold start frequency and memory consumption. Kumari et al. [25] proposed a two-stage study using Deep neural network (DNN) and LSTM models. The first step uses DNN for idle container window length estimation. In the second step, LSTM estimates the number of requests. The proposed model is tested on AWS Lambda, Azure, Openfaas, and Openwhisk. The results prove that it reduces the cold start frequency. Agarwal et al. [19] proposed a new approach using an RL-based Q-Learning algorithm that aims to reduce the frequency of cold start occurrence. In the proposed approach, the agent prevents cold start by keeping functions running in advance by monitoring metrics such as CPU usage and function instances. For performance evaluation, the authors deployed the proposed approach on Kubeless and found that throughput could increase by up to 8.81%. Pan et al. [20] introduced a new framework called SSC based on pre-warming to reduce the frequency of cold start occurrence. The proposed framework adapts to complex workflow structures using a gradient-based algorithm and aims to allocate resources efficiently and thus prevent cold start. The framework has been tested in OpenWhisk and a real server environment and has been found to reduce cold start occurrence by 50%.

2) Keeping Container Warm (FRW): Due to the Scale to Zero feature, after the execution of a function is completed, the container is expected to be terminated. In keeping containers warm, containers whose function execution is completed are kept ready for use for a certain period. Thus, it can respond to any function call during this period, and a cold start will be prevented. Google Cloud Functions, AWS Lambda, and Openwhisk platforms use such a mechanism to reduce the occurrence of cold starts [24].

By proposing an Adaptive Warm-Up (AWU) strategy, Xu et al. [26] aimed to estimate function request time and reduce the frequency of cold starts by warming containers. First, non-first functions are estimated using a fine-grained regression model. In the next step, the function launching time is reduced using the Adaptive Container Pool Scaling (ACPS) strategy. Finally, the capacity of the container pool is adjusted to avoid wasting resources as much as possible. Results are evaluated through Kubernetes. Fuerst et al. [27] aimed to reduce the frequency of cold start by applying the Greedy-Dual Caching policy called FaasCache to keep the container warm. The proposed method keeps functions alive according to source and usage patterns. The results show that FaasCache is more than 3x successful in reducing the cold start frequency compared to current studies. Moreno et al. [21] aimed to both reduce the latency in the network and accelerate the run container preparation process (cold start) of functions with the two-layer serverless edge architecture they proposed. For this, they use a double placement policy. According to the first placement policy, the first environment to be preferred to execute a function is the edge. If there are not enough resources in the edge environment, the function is sent to the cloud to be run. In the second placement policy, a warm

TABLE I
COMPARISON OF ATOM WITH EXISTING WORKS

Work	**Technique**	**Objective**	**Platform**	**RA**	**DFC**	**MSP**	**GD**	**EC**
Akkus *et al.* [16]	SAND	LR	Apache OpenWhisk, AWS Greengrass	✓	✗	✗	✗	✗
Agarwal *et al.* [17]	LSTM-PPO	LR	OpenFaaS	✓	✓	✗	✗	✗
Mampage *et al.* [18]	DRL	LR	Kubeless	✓	✓	✓	✓	✓
Agarwal *et al.* [19]	RL	FRP	Kubeless	✗	✓	✗	✗	✗
Pan *et al.* [20]	Gradient-based Algorithm	FRP	OpenWhisk, Real Server	✗	✓	✗	✓	✗
Vozmedia *et al.* [21]	Placement Policies	FRW	FaaSim, AWS	✗	✗	✗	✗	✗
Liu *et al.* [22]	Application-level Optimization	LR	AWS Lambda, Google Cloud Functions	✓	✗	✗	✗	✗
Mampage *et al.* [23]	DRL	FRW	Kubernetes	✗	✓	✓	✗	✗
Vahidinia *et al.* [24]	RL + LSTM	FRP	Openwhisk	✗	✓	✗	✗	✗
Kumari *et al.* [25]	DNN + LSTM	FRP	AWS Lambda, Azure, Openfaas, Openwhisk	✗	✓	✗	✗	✗
Xu *et al.* [26]	AWU + ACPS	FRW	Kubernetes	✗	✓	✗	✗	✗
Fuerst *et al.* [27]	Greedy-Dual Caching	FRW	OpenWhisk	✗	✓	✗	✗	✗
ATOM (this work)	**DRL**	**FRP**	**Google Cloud Functions**	✓	✓	✓	✓	✓

instance is checked in one of the edge or cloud environments and the function is sent to the warm instance environment in order to start faster. In cases where there is no warm instance, the initial placement policy will be followed. The authors use FaaSim, a FaaS platform, and AWS to test their proposed work.

C. Critical Analysis

Comparisons of this paper with current literature studies reviewed are given in Table I. The comparison table consists of a total of eight columns. The first column indicates the study examined, the second column indicates the technique used in the study, and the third column indicates the purpose of the study under review. It is written as LR if the aim of the study is Cold Start Latency Reduction or as FR if Cold Start Occurrence Frequency Reduction. Also, when doing Cold Start Occurrence Frequency Reduction, the preferred process is shown as FRP if Container Preparation or FRW if Keeping the Container Warm.

In this section, we analyze literature studies and this paper in terms of Resource-Aware (RA), Dynamic Function Call (DFC), Multi-step Ahead Prediction (MSP), and Generating Dataset (GD). Now let us explain these concepts:

Resource-Aware (RA): Most of the methods in the literature implement operations such as prewarmed containers, keeping containers warm, and container pools for Cold Start Latency Reduction and Cold Start Occurrence Frequency Reduction. Moreover, these processes will cause additional resource consumption. The "Resource-Aware" feature in the Table represents whether the techniques used in the studies examined require additional resource consumption.

Dynamic Function Call (DFC): This represents whether the reviewed studies consider varying "Function Calls". Because variable function calls and cold start occurrence are correlated in serverless environments, efforts to prevent cold start, such as keeping the container warm, ignore this correlation. There is a need for a model that can model the relationship between requests coming to the server and cold start and adapt to the dynamic nature of the cloud environment.

Multi-Step ahead Prediction (MSP): It is the estimation of multiple time steps into the future by utilizing a historical time series dataset. In serverless environments, a high success rate for the MSP holds significance as it facilitates the expedited transmission of the triggering PING to the server. Sending the PING early can be advantageous for functions that undergo an extended container preparation process.

Generating Dataset (GD): This represents the creation of a cold start dataset through the utilization of a real-world platform in the study.

Energy Cost (EC): It represents whether the energy cost-sensitive model is investigated in the method used.

For these reasons, RL and DRL-based algorithms can be used to reduce the cold start frequency. While RL-based algorithms learn dynamically by trial and error, DRL learns from existing knowledge and applies it to a new dataset [15]. It also requires discretizing continuous values for state representation in RL-based algorithms. This can cause problems such as state action space explosion [28]. Considering the cold start, CPU and RAM usage is continual, so it would be more logical to use DRL instead of RL.

III. PRELIMINARIES

This section explains the time series models and evaluation metrics used to better understand the proposed study.

A. Time Series Models

This section explains the three time-series models, LSTM, DDPG, and RDPG, respectively, used in the study.

Long Short-Term Memory (LSTM): Using a recurrent neural network (RNN) architecture, this model successfully captures long-term dependencies in time series [29]. It is used in time series and various fields, such as machine translation and speech recognition.

Deep Deterministic Policy Gradient (DDPG): DDPG was created by combining DL and Policy Gradient models to develop Actor-Critic-based DRL algorithms [30]. Its applications span across various fields and time series problems. In DDPG, the state is constructed from previous observations within a time series. Conversely, the action space encompasses predictions and forecast results generated by the model. Therefore, the action refers to the model's prediction for the upcoming value. Moreover, an actor-critic-based architecture is employed. The actor network makes predictions for a given series of time steps, while the critic network compares these predictions with the actual values using mean-squared error loss. The gradients from the critic network are utilized to optimize and enhance the predicted rewards within the actor network.

Recurrent Deterministic Policy Gradient (RDPG): It works as an extension of Deterministic Policy Gradient (DPG) and is a robust algorithm in scenarios such as time series forecasting [31]. RDPG and DDPG algorithms differ from each other in handling time series patterns. When extending DPG using DNN in DDPG, RDPG extends using RNNs. RNN allows a model to consider the observation history for future predictions using memory. First, the observation space, action space, and reward function are defined when used for time-series data. Then, the RNN architecture is chosen to capture the repetitive data patterns successfully. Using this architecture, the model interacts with the environment to create observations, actions, and rewards. This created data is kept in the replay buffer for later use. The policy and value networks are updated using this data in buffers, and the algorithm's loss functions are minimized.

B. Evaluation Metrics

In evaluating the performance of the time series models utilized in this paper, several metrics are employed, including root mean square error (RMSE), mean absolute error (MAE), coefficient of determination (R2 Score), and mean absolute percentage error (MAPE). These metrics are utilized for comparing and assessing the models' predictive accuracy and performance in handling time series data.

Root Mean Square Error (RMSE): Used to evaluate the accuracy of predictions in various fields such as statistics and data analysis [32]. The distance between the Predicted and Actual values is calculated using the Euclidean distance.

$$RMSE = \sqrt{\frac{1}{t}\sum\nolimits_{i=1}^{n}(A_t - P_t)^2}, \tag{1}$$

where t represents the number of units in the dataset, A_t is the actual value, and P_t is the predicted value.

Mean Absolute Error (MAE): MAE is widely used in time series analysis [32]. The difference between the prediction errors, i.e., Predicted value and Actual value, is the average of their absolute values.

$$MAE = \frac{\sum_i^t |A_t - P_t|}{n}. \tag{2}$$

Coefficient of Determination (R2 Score): Describes the ratio of dependent variable variance calculated using independent variables in a model [33]. The sum of the squared residuals is obtained by dividing R_{ss} by the sample mean squared deviations T_{ss}.

$$R^2 = 1 - \frac{R_{ss}}{T_{ss}}. \tag{3}$$

Mean Absolute Percentage Error (MAPE): Measures the prediction accuracy of a model [33]. Predicted value and Actual value Calculates the average percentage of the difference between.

$$MSE = \frac{1}{n}\sum_{i=1}^{t}(A_t - P_t). \tag{4}$$

Energy Cost: We use (5) to calculate the energy cost ($\pounds_{cost}$) for AI models.

$$\pounds_{cost} = \mathbb{P}_{cost} \times \top. \tag{5}$$

where $\mathbb{P}_{cost}$ represents the TDP power consumed by the processor. $\top$ is the calculated time to train and test the algorithm. Here, Thermal Design Power (TDP) measures the amount of heat produced by the CPU and can be used for an appropriate power target.[1]

Carbon Emissions: It represents the amount of CO_2 emitted into the atmosphere as a result of processes such as fossil fuels and industrial advancements [34]. Reducing carbon emissions is critical to preventing climate change and global warming. Cloud data centers are facilities that house servers that provide services such as storage and software to users over the Internet. It has basic duties such as providing energy and cooling for the server and other infrastructure systems inside. Since all these processes require electricity consumption, they also have a contribution to carbon emissions. Studies are continuing by researchers to reduce carbon emissions from cloud data centers [35]. Calculating the Carbon Emission amount requires various and difficult processes, but a general result can be obtained by

$$C_\xi = P_c * t * C_{IE}, \tag{6}$$

where C_ξ is carbon emission (kg CO_2), P_c is power consumption (kW), t is time (h), and C_{IE} is the carbon intensity of electricity (kgCO_2/kWh). The amount of C_{iE} varies from region to region and for this paper it is calculated as 182 gCO_2/kWh for the London area.[2]

Cold Start Latency: The following formula is used to calculate cold start latency in serverless computing.

$$C_{cold} = R_{first} - R_{second}, \tag{7}$$

where C_{cold} represents the cold start latency, R_{first} represents the first request sent to the server, and R_{second} represents the second request sent to the server.

IV. Methodology

This section explains the working mechanism of the ATOM framework.

A. ATOM Framework

The ATOM framework is shown in Fig. 1. Serverless edge computing consists of two primary layers, as seen in the figure. The first layer consists of an Edge Network of Edge and IoT devices, and the second layer consists of a serverless platform.

1) Edge Network: It consists of edge nodes with limited processing and storage capabilities and IoT devices such as sensors and smartphones. Edge nodes have a heterogeneous structure as they can consist of devices with different processing and storage capabilities. IoT devices can receive quick responses by sending the data they produce to edge nodes with higher processing

[1] https://www.intel.com/content/www/us/en/support/articles/000055611/processors.html

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

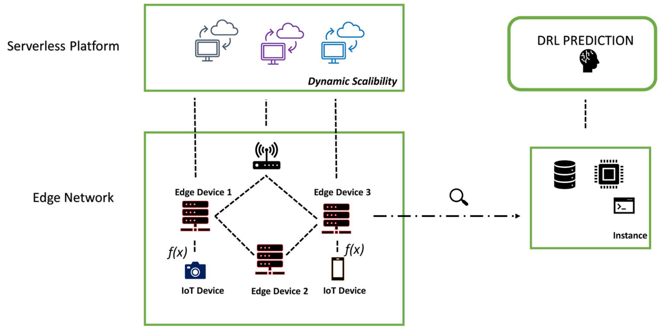


Fig. 1. Proposed ATOM framework.

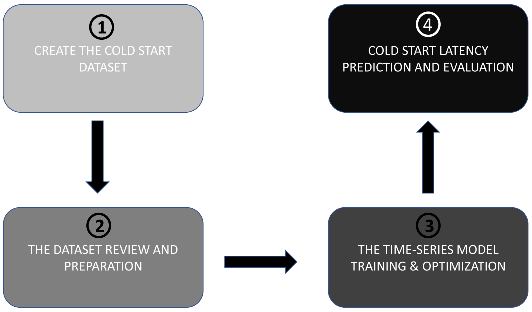

1
CREATE THE COLD START DATASET
2
THE DATASET REVIEW AND PREPARATION
3
THE TIME-SERIES MODEL TRAINING & OPTIMIZATION
4
COLD START LATENCY PREDICTION AND EVALUATION

Fig. 2. Flow chart of the execution of various steps in ATOM.

power. To do this, IoTs send events ($f(x)$) to edge nodes via Hypertext Transfer Protocol (HTTP), Message Queuing Telemetry Transport (MQTT), or Data Distribution Service protocols (DDS), depending on event-driven principles. It should be noted that the same or different infrastructure serverless platform providers (InPs) are deployed on edge nodes [8], [36]. In this way, edge nodes can manage the whole life-cycle of events sent to them or interact with another node to do this. Edge nodes trigger functions on the serverless platform to process $f(x)$ sent to them by IoTs. It sends the response obtained for $f(x)$ back to the IoT or assigns another edge node to do this.

2) Serverless Platform: A Serverless platform becomes necessary when the function $f(x)$ requires too large a processing capacity to be executed on the edge network. However, it's important to consider that the remote placement of cloud centers may lead to increased transmission costs and latencies.

The ATOM framework estimates the cold start time and thus lays the foundation for future work that avoids unnecessary energy and resource consumption by making instances run only when necessary, unlike container pooling [37] and container warming [25] solutions in the literature.

B. Research Organisation

In this section, the flow chart of the execution of various steps in ATOM is given in Fig. 2 so that the reader can better understand the flow of the methodology. The cold start dataset is created in the first step to train the Time-Series models and obtain forecast results. The obtained dataset is examined in the second step, and the preparation process is performed. In the third step, the training and optimizing process of two DRL and DL models is described. In the last step, the computation time, prediction performances, and accumulated reward metrics for these three models are evaluated in Section V.

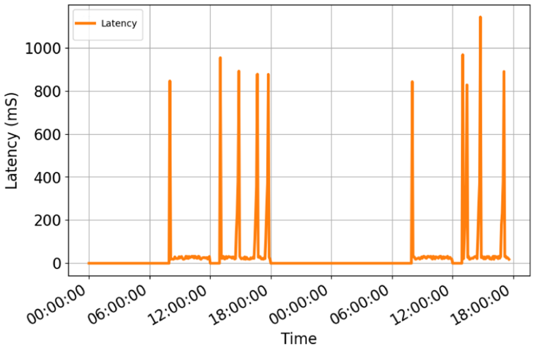


Fig. 3. Cold start latency dataset.

TABLE II
ENVIRONMENT PARAMETERS

Platform	GCP - Cloud Functions
Region	Europe-west2b
Runtime	Python 3.11
Function call format	HTTP
Memory	256 MB

1) Dataset Creation and Preparation: The cold start dataset used in this study is shown in Fig. 3. The cold start dataset was created on March 1-6, 2023, by following a serverless system that works 24 hours a day. For this, a heart disease risk scenario integrated into a serverless platform was deployed to Google Cloud Platform, i.e., Cloud functions [1]. The environment parameters used while creating the dataset are given in Table II. To create workloads and measure latency, between 1 and 250 simultaneous requests were sent to the server at staggered and 5-minute intervals using the Apache JMeter application. It has been choosing these values to simulate a real working environment and has been requested to the server between 08:00 and 18:00 for 6 days. Eq. (7) was used to calculate the latency of the server's responses to these concurrent requests. To calculate the cold start, the latency differences between the first and second requests are calculated [10]. As a result of the observations, it was seen that cold start occurs in two ways: (i) In cases where the number of simultaneous requests is 200 or more (because a new container is started on the Server), and (ii) In cases no request to the server more than 15 minutes (because containers kept warm by the platform are terminated after 15 minutes [14]). In the system, it was observed that the latency time was between 18-32 ms in cases where a cold start did not occur, and the latency times were between 800 - 2000 ms in cases where of cold start occurred. Communication channels use Transport Layer Security (TLS) protocol to address security and privacy concerns of sensitive biometric data in the dataset [38], [39], [40].

The framework architecture used when creating the cold start dataset is given in Fig. 4. In the heart risk scenario, 9 health data received from users are sent to a serverless platform via an Application Programming Interface (API). These data sent to

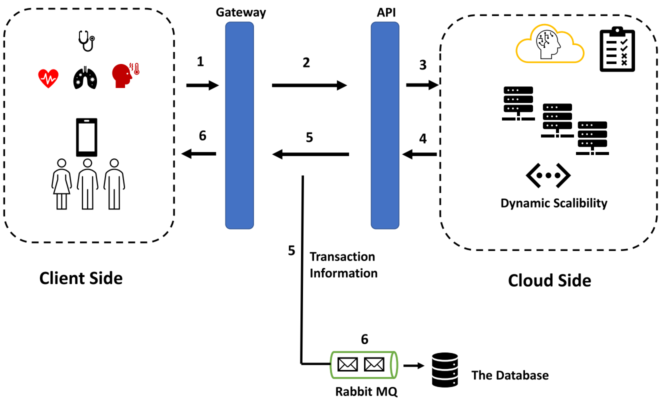


Fig. 4. Dataset creation mechanism.

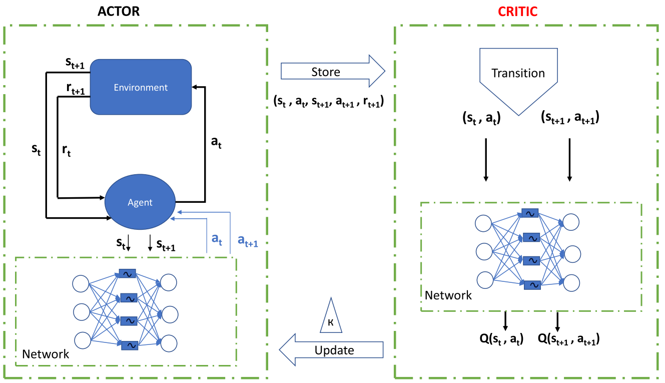


Fig. 5. Schematic diagram of DDPG/RDPG.

Serverless are given to the previously deployed Light Gradient Boosting Machine (LightGBM) model, which detects heart disease risk, via the Serverless API. The prediction results made in the Machine Learning (ML) model are returned to the client side. Transaction Information created during this transaction is saved in the database using Rabbit Message Que (MQ). Transaction Information contains seven variables “DateTime”, “Hour”, “Day”, “Latency”, “Request”, “CPU Usage”, and “Ram Usage”. DateTime shows the time period in “dd/mm/yy hh/mm/ss” format. Hour indicates the time for this period, Day indicates the day of the week for this period. Latency represents the total time spent on each request, and request represents the number of requests sent to the server. CPU Usage and RAM Usage show the amount of CPU and RAM usage used in the instance for requests coming to the server. The cold start dataset consists of this transaction information. We used Rabbit MQ while creating the database because by providing granular scalability, we aimed to prevent any data loss due to the risk of conflicts in the system.

Dataset Preparation: The cold start dataset contains 1,729 observations with variables “DateTime”, “Hour”, “Day”, “Latency”, “Request”, “CPU Usage”, and “Ram Usage”. There are no outliers or meaningless values in these observations. We use the “DateTime”, “Latency” and “Request” variables as inputs to train the models. To train Time series models, the dataset is split in half at a ratio of $3:1$.

2) Development and Optimization of Time Series Models: In this paper, time-series models have been developed for single-step ahead prediction. The models in which the observation in the next time step is predicted by looking at the past time-series values are called single-step ahead prediction. Since the single-step ahead is a simple prediction task, it is sufficient to examine the last 5-time step. Latency values (Lb) of the last 5-time step are given to the models as input, and the next step’s latency value (Lf) is predicted as output. In order to make cold start predictions with DRL models, let us first define state, action, and reward.

- *State:* It is the amount of latency corresponding to each time step. So in our single-step ahead prediction problem, the latency values of the last 5-time steps correspond to $[L_b, L_{b-1}, L_{b-2}, L_{b-3}, L_{b-4}]$. The state space, therefore, encompasses the latency values associated with each individual time step.
- *Action:* Action values consist of latency values ranging between 18 and 2,000. When the agent produces an output, it generates a value within this range. This specific output value stands as the predicted latency value for the given scenario.
- *Reward:* Reward should be created when evaluating the accuracy of the prediction (output) created by the Agent. The following equation is used when creating the Reward:

$$r_{t+1} = -|L_t - a_t|, \tag{8}$$

where L_t represents the actual amount of latency in the tth timestep, and a_t represents the predicted latency value for the tth timestep. The closer the latency value (a_t) predicted by the agent to the actual amount of latency (L_t), the closer the r_{t+1} value is to zero. Otherwise, the value of r_{t+1} will be negative.

All the steps described so far are the same for both DDPG and RDPG models. It has perceptual and decision-making abilities by combining DL and RL in two models. Actor-critic-based architecture and neural networks are used to predict the policy used. This architecture is shown in Fig. 5. Both actor and critic have neural networks. The actor network has the parameter τ, and its task is to find the output of the actions and predict the policy found in the system $(\pi_\tau(s,a))$. The Critic network has the parameter ψ to estimate the action-value function. The action-value function is used to update the policy in the system. The training phases are as follows: First, the agent interacts with the environment according to the policy in the actor network to obtain the transition values $(s_t, a_t, s_{t+1}, a_{t+1}, r_{t+1})$. It then calculates the action-value function for the critic (s_t, a_t) and (s_{t+1}, a_{t+1}) tuples. Finally, the Temporal difference (TD) error (ζ) is calculated. The TD error evaluates the new state after each action selection. The parameters of the actor are updated according to this error value.

The pseudo-code describing the training processes of the DDPG and RDPG models is given in Algorithm 1. Here, $\nabla_\tau \log \pi_\tau(s,a)$ score function, ι and η represent the learning rates for Actor and Critic. The training processes of DDPG and RDPG models are close to each other. The only difference is that RDPG extends DNN using RNNs. When the pseudo-code is examined, it is seen that there are two loops (Repeat-Until)

Algorithm 1: The Pseudo Code of Cold Start Latency Prediction.

```
(τ, ψ) parameters for Actor and Critic networks
(τ′ ← τ, ψ′ ← ψ) targets for Actor and Critic networks
Repeat
  Select random $s_t$ in $S_{space}$
  Repeat
    Select $a_t$ (Actor Network)
    Find $r_t$ using $r_t = -|L_t - a_t|$
    Update $s_t$ using $s_t = s_{t+1}$
    Select $a_{t+1}$ (Actor Network)
    Store $(s_t), a_t, s_{t+1}, a_{t+1}, r_t)$ (Buffer)
    Compute $\tau(s_t, a_t)$ (Critic network)
    Compute $\tau(s_{t+1}, a_{t+1})$ (Critic network)
    $\zeta = r_{t+1} + \kappa\, \tau(s_{t+1}, a_{t+1}) - \tau(s_t, a_t)$
    $d\tau = \iota \nabla_{\tau} log\pi_{\tau}(s, a).\zeta$
    $d\psi = \eta \frac{\partial \Sigma \zeta^2}{\partial \psi}$
    $s_t \leftarrow s_{t+1}$
  Until $S_t$ or Steps done
Until Episodes done
End
```

for loops run until they reach the last element in the dataset. So the time complexity of the algorithm complexity is $\mathcal{O}(n^2)$.

V. Performance Evaluation

In this section, the most successful and least energy-consuming time-series model in cold start prediction is determined to be used in the ATOM framework. To do this, the performances of three time-series models, namely LSTM, DDPG, and RDPG are compared using the cold start dataset. We choose DDPG and RDPG models as DRL models because the literature has shown that these two models are successful in complex and nonlinear problems [15]. We use LSTM as the DL model because two studies in the literature that reduced the cold start showed that this model was successful [24], [25]. The comparison is evaluated in two basic aspects: AI parameters and computing parameters, using ATOM and baselines. The first evaluation part is AI parameters, including i) Comparison of cold start latency and request prediction accuracy performances of DRL and baseline models. ii) Loss per period examination to evaluate the performance of the baseline approaches on the dataset. iii) Episode-reward performance comparison to determine the most effective DRL model. iv) The most successful DRL model's and baseline's approach multi-step ahead prediction performance on the cold start dataset. v) Comparison of calculation times of DRL and baseline approaches. The second evaluation part is computing parameters, including i) Energy cost calculation and comparison of all models. ii) Carbon emission amount and comparison for all models. ii) Observation of the impact of serverless edge computing on network latency.

A. Experimental Setup

This section describes the test environments and model parameters used in the experiments so that future authors can reproduce this work.

TABLE III
System Configuration

CPU	Intel® Core™ i7-10750H
Clock Speed	12M Cache, up to 5.00 GHz
RAM	16 GB
OS	Windows 10 Pro
TDP (W)	45

TABLE IV
Model Hyperparameters

Model	**Hyperparameters**

LSTM	optimizer='Adadelta', loss='mse',
	epoch=50, activation='softmax', input_shape=(10, 1), Dense =1

DDPG	$N_f = 2$, $LR_a = 0.0001$, $LR_c = 0.01$, $N_{ah} = 30$, $N_{ch} = 30$, $MAX_{ep} = 100$
RDPG	$N_f = 6$, $LR_a = 0.001$, $LR_c = 0.003$, $N_{ah} = 30$, $N_{ch} = 30$, $MAX_{ep} = 100$

System Configuration: All performance tests were performed using the same system and are in the system configuration Table III.

Model Parameters: Optimizing model parameters significantly influences the performance of time-series models. Unlike LSTM models, DRL involves a larger number of parameters. Consequently, in determining the parameters for DRL models, the number of Actor and Critic layers was set and computed using a genetic algorithm. In contrast, parameter optimization for the LSTM model was computed with consideration to R^2. Through a series of extensive experiments, hyperparameter optimization was conducted for all models, and the findings are outlined in Table IV.

B. Baseline Models

In serverless computing, we mentioned that one of the reasons for a cold start is that the number of requests coming to the server is greater than the maximum capacity that the container can process. For this reason, it is important to capture the dependencies between the number of requests coming to the server and cold start. Additionally, another important thing is the detection of latency patterns caused by cold start. Latency formation higher than a certain threshold value will provide important clues for a cold start. In this section, we compare the cold start latency prediction and concurrent request prediction performances of the proposed framework (ATOM) with current cold start-based baselines such as Warm-Start Containers (WSA) [25], and Two-layer Adaptive (TLA) [24].

WSA Approach [25]: In the first stage of the proposed adaptive model, the window length is estimated by using the neural network model [22]. The window length represents how long the container will be kept warm after the function has completed execution. In the second stage, the number of concurrent requests is estimated using LSTM. This way, a warm container can be prepared sufficient for the number of requests, and the cold start frequency can be reduced.

TLA Approach [24]: Similar to the previous work, the authors use the actor-critic algorithm to determine the duration of keeping containers warm in the first layer. Containers continue to be

TABLE V
COMPARISON BETWEEN ATOM AND THE BASELINES REGARDING LATENCY AND REQUEST PREDICTION PERFORMANCES USING THE TEST DATASET

Work	Model	RMSE	MAE	R2 Score
ATOM (latency)	DDPG	148.76	51.57	0.071
	RDPG	184.82	85.05	-0.426
TLA [24]	LSTM	162.59	45.59	-0.085
ATOM (request)	DDPG	25.66	13.48	0.762
	RDPG	54.41	46.96	-0.07
WSA [25]	LSTM	64.71	37.33	-0.498

kept warm according to this determined period. Thus, cold start formation caused by requests to the server can be prevented. In the second layer, function invocation times are detected by using LSTM (with time-series problems). In this way, it is aimed to avoid a cold start by preparing prewarmed containers before the request comes to the server.

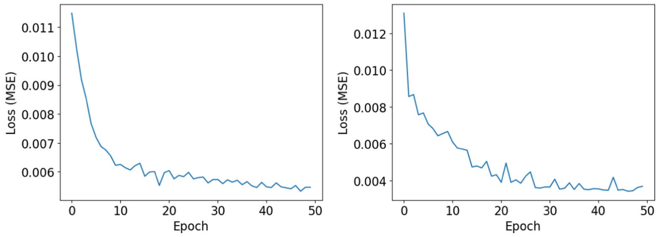
C. Workloads

We provide the basis for energy-aware studies aimed at reducing the frequency of cold starts in serverless environments. The occurrence of cold starts is predicted through the application of DRL techniques, known for their effectiveness in addressing intricate and non-linear challenges. The initial step involves generating a cold start dataset to train all DRL models within a serverless environment. To accomplish this, the HealthFaaS framework [1] was implemented on the Google Cloud Platform (GCP) - Cloud Functions. Workload creation was achieved using Apache JMeter, wherein simultaneous HTTP requests ranging between 0 and 250 were sent to the server, resulting in the acquisition of a dataset containing instances of cold start situations. Subsequently, all DRL and DL models were trained to utilize the previously acquired cold start dataset. Following this training, a comparative analysis was conducted among the models, evaluating their prediction performance, energy consumption, as well as their impact on CO_2 emissions.

D. Results

In this subsection, the initial step involves identifying the most effective DRL model, specifically the ATOM framework, in predicting both cold start latency and the number of requests. Subsequently, a comparison of performance is conducted with the LSTM models [24], [25]. Energy consumption and CO_2 emission amount tests are then performed to find the most efficient model for all AI models.

1) AI Parameters: Latency & Request Prediction: The cold start latency and request prediction performances of these three models are compared with three evaluation indices: "RMSE", "MAE", and "R2 Score". The results are given in Table V. For latency prediction, It shows that among the three time-series models, the DDPG model is the most successful model, with 148.76 RMSE, and the RDPG model is the most unsuccessful model, with 184.82 RMSE. Similar to the result of latency prediction, the request prediction performance of these three models is compared with three evaluation indices: "RMSE", "MAE" and "R2 Score". Among the three time series models, it can be seen that the DDPG model is the most successful model with 25.66 RMSE, and the RDPG model is the least successful model with 64.71 RMSE. Additionally, the results show that the DDPG model has much better performance than the baseline models (WSA [25] and TLA [24]). DDPG models continuously learn in each episode and update policies as a result of these observations. In dynamic time series problems such as cold start, LSTM (Deep Learning - DL) makes predictions with lower accuracy because it does not have the discovery mechanism as in DDPG. As a result, the DDPG model with the highest performance for both latency and request prediction has been applied to the ATOM framework. Thus, for the next experiments, the results obtained for DDPG are also valid for the ATOM framework. Although the RDPG model is expected to be more successful than the DDPG, there may be several reasons why it performs worse. First of all, DDPG allows agents to find new states with its exploration-exploitation trade-off policy. In this way, it can discover more successful policies. However, RDPG has difficulty in discovering new states in time-series-based problems and this hinders it from discovering successful policies as in DDPG. In addition, DDPG can develop policy more quickly with less data compared to RDPG. This may explain the faster convergence and higher success of DDPG on the cold start dataset. Moreover, DDPG uses DNNs to capture complex patterns and correlations in the cold start dataset. Likewise, the RDPG model can capture complex patterns and correlations in the cold start dataset using a neural network, but its repetitive nature may cause limitations in capturing long-term patterns and correlations.



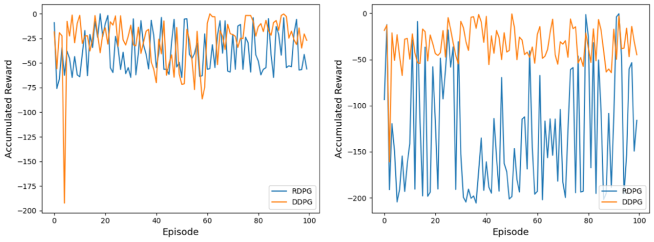
(a) LSTM TLA [24]

(b) LSTM WSA [25]

Fig. 6. Loss per epoch for latency and request prediction training.

Fig. 6 shows the Loss per Epoch graph obtained in the training phase for the LSTM model used in two baseline models [24], [25]. The Loss Function is calculated using the mean squared error (MSE) and shows the observed loss per episode. The loss function tends to decrease and the accuracy of the LSTM model for both latency and request prediction increases in each epoch. This shows that the LSTM model tends to converge.

Fig. 7 shows episode-reward graphs for latency (a) and request (b) prediction of DRL models. These graphs show the rewards that an agent receives cumulatively throughout the episodes, i.e., learning success The chart includes only the first 100 episodes. When the latency prediction graph is examined, it is seen that the reward total of the DDPG model is unstable and decreasing until approximately the 7th episode. But it reaches stability after

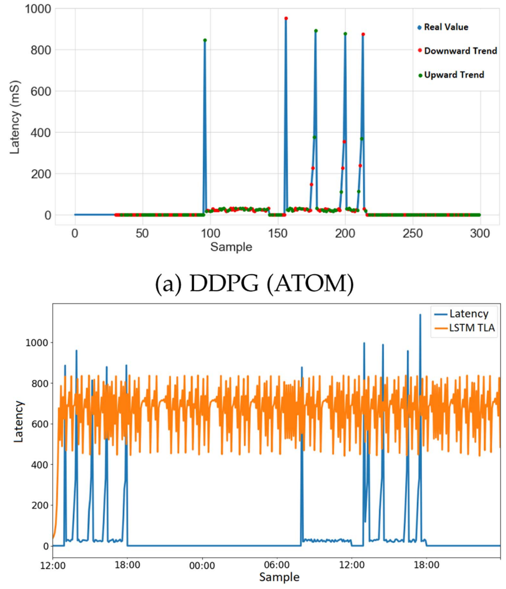


(a) For Latency Prediction (b) For Request Prediction

Fig. 7. Performance comparison of DDPG and RDPG.

TABLE VI
COMPARISON OF ATOM WITH BASELINES IN TERMS OF COLD START LATENCY AND REQUEST PREDICTION COMPUTATION TIME

Work	Model	Training (sec)	Prediction (sec)
ATOM	DDPG	118.76	0.12
(latency)	RDPG	738	0.15
TLA [24]	LSTM	18	0.30
ATOM	DDPG	110.94	0.21
(request)	RDPG	729.78	0.21
WSA [25]	LSTM	6.6	0.08



(a) DDPG (ATOM)

(b) LSTM (TLA)

Fig. 8. Performance comparison of ATOM and TLA in terms of latency.

the 7th episode. RDPG is seen to change relatively constantly between -75 and 0 values. When the request prediction graph is examined, it is seen that the DDPG model reaches stability after approximately the 5th episode, while the RDPG model is constantly unstable and cannot learn well. As a result, the reward of the DDPG model is better than the RDPG in both cases. This shows that DDPG is more stable and converges better.

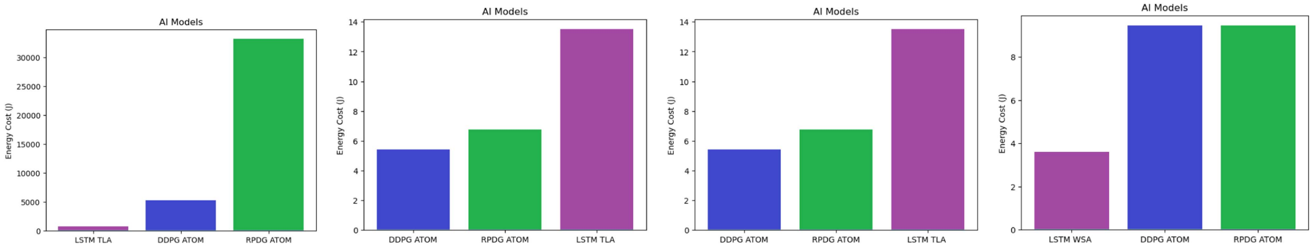
In Fig. 8(a), we conducted a test displaying trend points to demonstrate the multi-step ahead prediction performance of the DDPG model using the cold start dataset. The red dots indicate the “Downward Trend”, signifying a decrease in latency, while the green dots indicate the “Upward Trend”, signifying an increase in latency. The x-axis illustrates the number of time steps observed, while the y-axis represents the latency value. Multi-step ahead prediction was made by selecting the first 30 time steps. Fig. 8(b) shows cold start latency prediction with the LSTM model used in the TLA base study. While the predictive results show promise in successfully identifying cold start situations when a particular threshold is established, they may not yield satisfactory accuracy for general latency prediction purposes.

Time Comparison: The computation times for the prediction of cold start latency and request number for the three time-series models examined in this paper are given in Table VI. The results show that the DRL models require very high training time compared to the LSTM (baseline models). DRL models consist of processing stages where the agent interacts with the environment to collect the most appropriate strategy and highest reward. These stages take time, so DRL models have a higher training time. When DRL models are compared within themselves, it can be seen that RDPG trains around 6 times longer than DDPG to predict latency and request. This is because RDPG contains RNNs. RNNs add increasing complexity to the RDPG model, allowing it to learn better from past situations and actions. This complexity results in longer training time. When we look at the training times, it is seen that LSTM is much faster than other models. The LSTM model can be parallelized on sequential data such as time series. This feature allows multiple computations to be performed simultaneously on the GPU, resulting in a very short processing time. When the table is examined, it is seen that the prediction time for all models is negligible compared to training.

2) Computing Parameters: This section involves a comparison of the Energy Consumption and CO_2 Emission associated with three different algorithms for predicting cold start latency and request number. Additionally, the last subtitle examines the impact of serverless edge computing on network latency.

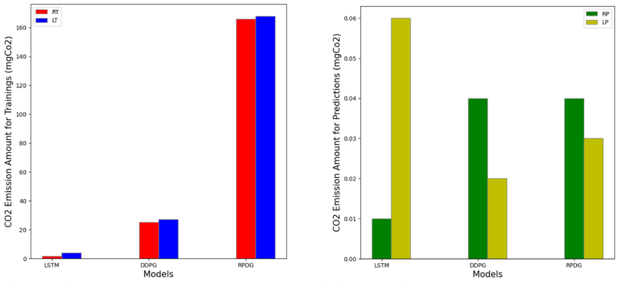
3) Energy Consumption: Fig. 9 shows the amount of energy consumed for the train and prediction phases for cold start latency and the request number of the three different AI models. The results calculated using (6) show that while DRL models consume the most energy for model training, the model that consumes the least energy is LSTM. The biggest reason for this is that DRL models require more computation than DL and ML. Considering the prediction performances of AI models, this situation is slightly different. The amount of energy consumed for almost all models is close to each other. However, the results show that the most energy-consuming model for the prediction process is DDPG, and the least energy-consuming model is LSTM. This may be due to intra-model differences in the LSTM model, such as data Requirements and inference complexity.

4) CO_2 Emission Amount: Increasing energy efficiency in edge and cloud computing also means reducing energy consumption. In the IT sector, this issue is important for reducing the carbon emission rate and therefore reducing its contribution to global environmental problems. To create this awareness, carbon



(a) Training Latency Prediction (b) Latency Prediction (c) Training Request Prediction (d) Request Prediction

Fig. 9. Energy costs comparison of ATOM framework with baselines.



(a) Carbon emission amount for training of models (b) Carbon emission amount for testing of models

Fig. 10. Carbon emission comparison of ATOM framework with baselines.

emission amounts for all AI models examined in this paper were calculated using (6).

Fig. 10 shows the calculated CO_2 emissions for Request Training (RT), Latency Training (LT), Request Prediction (RP), and Latency Prediction (LP). When the figures are examined carefully, it is seen that the DDPG model has the highest CO_2 emission amount for training processes, and the LSTM model has the highest CO_2 emission amount for prediction processes. It is an important research gap for researchers to conduct studies that increase the accuracy rate in the IT sector, while also taking into account energy consumption and CO_2 emissions.

5) Impact of Serverless Edge Computing on Network Latency: In this section, we observe the effect of Serverless Edge Computing, which is based on bringing the serverless platform closer to the source, on network delays. So, we compare the response times of serverless computing and serverless edge computing. For creating a workload, we use a Python script that calculates the Fibonacci series. We calculate the Fibonacci number in increasing numbers as 100, 500, and 1,000 respectively. We distribute this Python script to both a serverless platform and a serverless edge environment. For this, we created two different instances using GCP Cloud Functions. For the serverless scenario, the first instance was deployed to a GCP region far from the location where the experiment was conducted. For the serverless edge scenario, the second instance was deployed to the GCP region in the same city as the location where the investigation was conducted. The environment parameters are shown in Table VII.

TABLE VII
ENVIRONMENT PARAMETERS

Instance	Region	Zone	Machine Configuration	Machine Type
1	south america east1 (Sao Paulo)	south america east1	E2 Series	2vCPU 1 Core 4 GB RAM
2	europe west2 (London)	europe west2c	E2 Series	2vCPU 1 Core 4 GB RAM

TABLE VIII
SERVERLESS AND SERVERLESS EDGE COMPUTING NETWORK LATENCY COMPARISON

Fibonacci Number	Serverless Scenario	Serverless Edge Scenario
100	0.002s	0.001s
500	0.008s	0.005s
1000	0.32s	0.003s

Table VIII shows the results. As can be seen, serverless edge computing provides advantages for time-sensitive scenarios by reducing network delays.

VI. CONCLUSION AND FUTURE WORK

The serverless paradigm is one of the latest cloud delivery models that attract attention with the advantages it offers such as pay-as-you-go and automatic scaling of resources. This paradigm has recently been distributed to edge nodes which has advantages such as low latency and bandwidth, resulting in a new research field, serverless edge computing. However, before it is adopted by the academy, it brings with it problems that still need to be solved, such as cold start. Most of the cold start solutions found in the literature require unnecessary use of resources. This article presents ATOM, an AI-powered Sustainable framework for predicting cold start in serverless edge computing. Latencies due to cold start can cause significant problems in time-sensitive IoT applications such as virtual reality. For this purpose, a cold start dataset was created by applying a study in the literature determining the risk of heart disease to the real-world serverless platform (GCP-Cloud Functions). Two DRL techniques (DDPG and RDPG) are tested on the cold start dataset to observe the cold start latency. It also compares with two different baselines (LSTM).

Experimental results show that the DDPG model outperforms the other two models with an RMSE ratio of 148.76. The performance of two current studies (LSTM) in the literature is evaluated with the Loss per Epoch graph. The results showed that the LSTM model was more unsuccessful than the DRL model. DRL models are compared over episode reward. It was seen that DDPG is more stable and superior to RDPG. The multi-step ahead prediction graph on the cold start dataset of the most successful DRL model (DDPG) has been interpreted. Furthermore, finally, Energy Consumption Comparison and CO_2 Emission Amount were measured for all models. DRL models require more energy consumption and CO_2 emission than other models.

This study shows that DRL models are successful in predicting cold starts and have great potential to reduce the frequency of cold starts. In future work, DRL models can be positively extended to other resource management problems in serverless computing. In future studies, monitoring can be done for cold start by adding a monitoring mechanism. Advanced ML/DL techniques can be applied to improve cold start prediction performance. Additionally, by extending the ATOM framework, work can be done to prevent cold start for serverless environments. Additionally, since the cold start dataset was created using only a single-function service, detailed information about scalability cannot be obtained. Therefore, the dataset can be developed using multiple functions. Finally, by using online ML, a more cost-effective and efficient system can be created in resource-constrained environments.

REFERENCES

[1] 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.

[2] H. Hua, Y. Li, T. Wang, N. Dong, W. Li, and J. Cao, “Edge computing with artificial intelligence: A machine learning perspective,” *ACM Comput. Surv.*, vol. 55, no. 9, pp. 1–35, 2023.

[3] G. Cheng, Z. Wan, W. Ding, and R. Sun, “Memory allocation strategy in edge programmable logic controllers based on dynamic programming and fixed-size allocation,” *Appl. Sci.*, vol. 13, no. 18, 2023, Art. no. 10297.

[4] A. R. Nandhakumar, A. Baranwal, P. Choudhary, M. Golec, and S. S. Gill, “EdgeAISim: A toolkit for simulation and modelling of AI models in edge computing environments,” *Meas.: Sensors*, vol. 31, 2023, Art. no. 100939.

[5] R. Singh and S. S. Gill, “Edge AI: A survey,” *Internet Things Cyber- Phys. Syst.*, vol. 3, pp. 71–92, 2023.

[6] K.-S. Wong, M. Nguyen-Duc, K. Le-Huy, L. Ho-Tuan, C. Do-Danh, and D. Le-Phuoc, “An empirical study of federated learning on IoT-edge devices: Resource allocation and heterogeneity,” 2023, *arXiv:2305.19831*.

[7] H. Ko and S. Pack, “Function-aware resource management framework for serverless edge computing,” *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1310–1319, Jan. 2023.

[8] C. Chen, L. Nagel, L. Cui, and F. P. Tso, “S-cache: Function caching for serverless edge computing,” in *Proc. 6th Int. Workshop Edge Syst. Analytics Netw.*, 2023, pp. 1–6.

[9] N. Eskandani and G. Salvaneschi, “The uphill journey of FaaS in the open-source community,” *J. Syst. Softw.*, vol. 198, 2023, Art. no. 111589.

[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 22nd Int. Symp. Cluster Cloud Internet Comput.*, 2022, pp. 886–892.

[11] M. Golec et al., “BlockFaaS: Blockchain-enabled serverless computing framework for AI-driven IoT healthcare applications,” *J. Grid Comput.*, vol. 21, no. 4, 2023, Art. no. 63.

[12] M. Xu et al., “CoScal: Multifaceted scaling of microservices with reinforcement learning,” *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 3995–4009, Dec. 2022.

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

[14] 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*.

[15] T. Liu, Z. Tan, C. Xu, H. Chen, and Z. Li, “Study on deep reinforcement learning techniques for building energy consumption forecasting,” *Energy Buildings*, vol. 208, 2020, Art. no. 109675.

[16] I. E. Akkus et al., “SAND: Towards high-performance serverless computing,” in *Proc. Usenix Annu. Tech. Conf.*, 2018, pp. 923–935.

[17] S. Agarwal, M. A. Rodriguez, and R. Buyya, “A deep recurrent-reinforcement learning method for intelligent autoscaling of serverless functions,” 2023, *arXiv:2308.05937*.

[18] A. Mampage, S. Karunasekera, and R. Buyya, “A deep reinforcement learning based algorithm for time and cost optimized scaling of serverless applications,” 2023, *arXiv:2308.11209*.

[19] S. Agarwal, M. A. Rodriguez, and R. Buyya, “Reinforcement learning (RL) augmented cold start frequency reduction in serverless computing,” 2023, *arXiv:2308.07541*.

[20] 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.

[21] R. Moreno-Vozmediano, E. Huedo, R. S. Montero, and I. M. Llorente, “Latency and resource consumption analysis for serverless edge analytics,” *J. Cloud Comput.*, vol. 12, no. 1, pp. 1–22, 2023.

[22] 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.

[23] A. Mampage, S. Karunasekera, and R. Buyya, “Deep reinforcement learning for application scheduling in resource-constrained, multi-tenant serverless computing environments,” *Future Gener. Comput. Syst.*, vol. 143, pp. 277–292, 2023.

[24] 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.

[25] 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.

[26] Z. Xu, H. Zhang, X. Geng, Q. Wu, and H. Ma, “Adaptive function launching acceleration in serverless computing platforms,” in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst.*, 2019, pp. 9–16.

[27] A. Fuerst and P. Sharma, “FaasCache: Keeping serverless computing alive with greedy-dual caching,” in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2021, pp. 386–400.

[28] S. Agarwal, M. A. Rodriguez, and R. Buyya, “A reinforcement learning approach to reduce serverless function cold start frequency,” in *Proc. IEEE/ACM 21st Int. Symp. Cluster Cloud Internet Comput.*, 2021, pp. 797–803.

[29] K. Smagulova and A. P. James, “A survey on LSTM memristive neural network architectures and applications,” *Eur. Phys. J. Special Topics*, vol. 228, no. 10, pp. 2313–2324, 2019.

[30] J.-C. Chen, C.-X. Chen, L.-J. Duan, and Z. Cai, “DDPG based on multi-scale strokes for financial time series trading strategy,” in *Proc. 8th Int. Conf. Comput. Technol. Appl.*, 2022, pp. 22–27.

[31] G. Chen et al., “Multiple network embedding for anomaly detection in time series of graphs,” 2020, *arXiv:2008.10055*.

[32] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)?–Arguments against avoiding RMSE in the literature,” *Geoscientific Model Develop.*, vol. 7, no. 3, pp. 1247–1250, 2014.

[33] D. S. K. Karunasingha, “Root mean square error or mean absolute error? Use their ratio as well,” *Inf. Sci.*, vol. 585, pp. 609–629, 2022.

[34] T. Renugadevi, K. Geetha, K. Muthukumar, and Z. W. Geem, “Optimized energy cost and carbon emission-aware virtual machine allocation in sustainable data centers,” *Sustainability*, vol. 12, no. 16, pp. 1–27, 2020.

[35] M. Uddin, Y. Darabidarabkhani, A. Shah, and J. Memon, “Evaluating power efficient algorithms for efficiency and carbon emissions in cloud data centers: A review,” *Renewable Sustain. Energy Rev.*, vol. 51, pp. 1553–1563, 2015.

[36] M. S. Aslanpour et al., “Serverless edge computing: Vision and challenges,” in *Proc. Australas. Comput. Sci. Week Multiconference*, 2021, pp. 1–10.

[37] P.-M. Lin and A. Glikson, “Mitigating cold starts in serverless platforms: A pool-based approach,” 2019, *arXiv:1903.12221*.

[38] 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.

[39] F. Li, J. Zhao, H. Yang, D. Yu, Y. Zhou, and Y. Shen, “VibHead: An authentication scheme for smart headsets through vibration,” *ACM Trans. Sensor Netw.*, 2023.

[40] Y. Tao, S. Chen, F. Li, D. Yu, J. Yu, and H. Sheng, “A distributed privacy-preserving learning dynamics in general social networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 9547–9561, Sep. 2023.

Muhammed Golec received the MSc degree in computer science (Distinction) from QMUL in through the Ministry of Education Scholarship. He is currently working toward the PhD degree in computer science with the Queen Mary University of London (QMUL). He has published articles in prominent journals and conferences such as *IEEE Transactions on Industrial Informatics* and *Elsevier Internet of Things*, *IEEE Consumer Electronics Magazine*, and IEEE CCGRID. His research interests include cloud computing, serverless computing, AI, and security and privacy.

Sukhpal Singh Gill (FHEA) is an assistant professor of cloud computing with the School of Electronic Engineering and Computer Science, Queen Mary University of London, U.K. Prior to his present stint, he has held positions as a research associate with the Lancaster University, U.K. and also as a postdoctoral research fellow with CLOUDS Laboratory, The University of Melbourne, Australia. He is serving as an associate editor in *IEEE Internet of Things Journal*, *Wiley Software: Practice and Experience*, *Elsevier Internet of Things*, *Wiley Transactions on Emerging Telecommunications Technologies* and *IET Networks Journal*. He has co-authored more than 150 peer-reviewed papers (with more than 7000 citations and H-index 41) and has published in prominent international journals and conferences such as *ACM Computing Surveys*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Consumer Electronics*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Sustainable Computing*, *IEEE Transactions on Network and Service Management*, *IEEE Internet of Things Journal*, *Elsevier Journal of Systems and Software/Future Generation Computer Systems*, IEEE/ACM UCC, Springer ICSOC and IEEE CCGRID. His research interests include cloud computing, edge computing, IoT and energy efficiency. For further information, please visit: http://www.ssgill.me.

Felix Cuadrado received the PhD degree in telecommunications engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2009. He is currently a senior distinguished fellow (Beatriz Galindo scheme) with the Universidad Politécnica de Madrid, a visiting reader with the Queen Mary University of London, and a fellow of the Alan Turing Institute. He has numerous publications in top-tier journals and conferences, including *IEEE Transactions on Services Computing*, *IEEE Transactions on Cloud Computing*, *Elsevier Journal of Systems and Software*, *Elsevier Future Generation Computer Systems*, IEEE ICDCS, and WWW. His research explores the challenges arising from large-scale data-intensive applications through a combination of software engineering, distributed systems, and mathematical approaches.

Ajith Kumar Parlikad received the PhD degree from Cambridge University, in August 2006. He is professor of asset management with Cambridge University Engineering Department. He is based with the Institute for Manufacturing, where he is the head of the Asset Management Research Group. He is a fellow and tutor with Hughes Hall. He leads research activities on engineering asset management and maintenance. For his PhD, he developed a methodology for quantifying the benefits of improving product information availability and quality on the effectiveness of product recovery processes.

Minxian Xu (Member, IEEE) received the BSc and MSc degrees both in software engineering from the University of Electronic Science and Technology of China, in 2012 and 2015, respectively, and the PhD degree from the University of Melbourne, in 2019. He is currently an Associate Professor with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences. His research interests include resource scheduling and optimization in cloud computing. He has co-authored more than 40 peer-reviewed papers published in prominent international journals and conferences, such as *ACM Computing Surveys*, *ACM Transactions on Internet Technology*, *IEEE Transactions on Sustainable Computing*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Automation Science and Engineering*, *IEEE Transactions on Green Communications and Networking*, *Journal of Parallel and Distributed Computing*, *Journal of Systems and Software* and ICSOC. His PhD Thesis was awarded the 2019 IEEE TCSC Outstanding PhD Dissertation Award. More information can be found at: minxianxu.info.

Huaming Wu (Senior Member, IEEE) received the BE and MS degrees from the Harbin Institute of Technology, China, in 2009 and 2011, respectively, both in electrical engineering, and the PhD degree with the highest honor in computer science from Freie Universität Berlin, Germany, in 2015. He is currently an associate professor with the Center for Applied Mathematics, Tianjin University, China. His research interests include mobile cloud computing, edge computing, Internet of Things, deep learning, complex networks, and DNA storage.

Steve Uhlig received the PhD degree in applied sciences from the University of Louvain, Belgium, in 2004. Prior to joining Queen Mary, he was a senior research scientist with Technische Universität Berlin/Deutsche Telekom Laboratories, Berlin, Germany. Starting in January 2012, he is the professor of Networks and head of the Networks Research Group, Queen Mary, University of London. Between 2012 and 2016, he was a guest professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. His current research interest includes Internet measurements, software-defined networking, content delivery.