National College of Ireland

Predictive Modelling and Cold Start Mitigation in Function-as-a-Service Using BiLSTM with Multi-Head Mechanism

MSc Research Project
Cloud Computing

Mohan Sai Morla
Student ID: x23418613

School of Computing
National College of Ireland

Supervisor: Prof Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Mohan Sai Morla
Student ID:	x23418613
Programme:	Cloud Computing
Year:	2025-2026
Module:	MSc Research Project
Supervisor:	Prof Aqeel Kazmi
Submission Due Date:	11/12/2025
Project Title:	Predictive Modelling and Cold Start Mitigation in Function-as-a-Service Using BiLSTM with Multi-Head Mechanism
Word Count:	5980
Page Count:	20

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Mohan Sai Morla
Date:	28th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Predictive Modelling and Cold Start Mitigation in Function-as-a-Service Using BiLSTM with Multi-Head Mechanism

Mohan Sai Morla
x23418613

Abstract

Serverless computing platforms, particularly Function-as-a-Service (FaaS) environments suffer from cold-start latency problems where function containers require initialisation time during the first invocation after idle periods by causing delays of hundreds of milliseconds to several seconds. This unpredictable latency severely affects real-time applications, machine learning inference services, and user experience, making intelligent cold-start mitigation essential for enterprise serverless adoption. This study developed three deep learning architectures LSTM, BiLSTM and Multi-Head mechanisms to forecast function invocation patterns. Using Microsoft Azure Functions dataset containing 1.98 million real-world traces these models generated predicted wait intervals between invocations. Apache OpenWhisk served as the FaaS platform where Apache JMeter continuously invoked custom workload through the invoker module with Docker containers monitored for warm pre-loaded in memory versus cold requiring initialisation states. The Multi-Head BiLSTM achieved an MSE of 10,958.96 and RMSE of 104.69, reducing cold starts from 28.5 percent to 9 pecrent across 50,000 invocations. This study advances serverless performance optimisation by introducing intelligent container lifecycle management on AWS EC2 infrastructure. Future work should address multi-platform validation, real-time adaptive learning and cost-benefit optimization for production deployments.

Keywords: Cold Start, FaaS, BiLSTM, Multi-Head, Serverless Computing

1 Introduction

1.1 Background

Cloud computing Serverless computing has transformed the development of cloud applications by allowing developers to scale functions Lannurien et al. (2023) without concern about managing an underlying infrastructure, where platforms such as AWS Lambda, Azure Functions and Google Cloud Functions provide auto-scaling and a pay-as-you-go pricing model. Nevertheless, the cold-start issue is a significant efficiency limitation of Function-as-a-Service (FaaS) settings Segarra et al. (2024). Cold starts can be considered as happening when serverless functions are called in the aftermath of idle time Karamzadeh and Shameli-Sendi (2024), necessitating the containerization, configuration of the runtime environment, and dependency loading, which introduces latency of between

hundreds of milliseconds and a few seconds. The instability has a detrimental effect on real-time applications, the inference services of machine learning, and APIs that need users to receive low-latency responses consistently. Conventional mitigation methods like periodical pinging or keeping container pools warm are very resource-consuming and unaffordable. The most recent discoveries in machine learning, especially deep learning models such as Long Short-Term Memory (LSTM) networks provide potential solutions in predicting function invocation patterns.

1.2 Importance of the Study

The main problem examined in the current study is the cold-start latency of serverless computing that has a direct effect on user experience, application responsiveness, and operational efficiency in Function-as-a-Service (FaaS) systems. With the insight of predicting patterns of function invocation by exploiting deep learning networks such as LSTM, BiLSTM and Multi-head BiLSTM models, this study allows proactive container warming scenarios to be applied and bring about substantial decreases in the time required to initialize the application. The practical implication is that organisations running machine learning workloads on serverless systems have a high cost due to unpredictable cold starts that lead to response time delays of hundreds of milliseconds down to several seconds. The fact that real-world Azure Functions data, with 1.98 million invocation traces, is used makes sure that the results are based on the workloads at the production level, but not on the synthetic data. Moreover, predictive modelling combined with load testing on the basis of Apache JMeter running on the AWS EC2 infrastructure is a proven framework of performance optimisation that can be reproduced.

1.3 Research Question and Research Objectives

Function-as-a-Service (FaaS) systems such as AWS Lambda, Azure Functions and Google Cloud Functions (also known as serverless computing) are highly affected by cold-start latency since it causes high performance overhead. Cold starts are started when the containers of functions are loaded when they are idle and take a long time of a few hundred milliseconds, and a few seconds. The cause of these initialization delays can be attributed to several factors such as provisioning of a container, setting up of a runtime environment, dependency loading and initializing of an application code. Traditional mitigation approaches are expensive and consume clouds resources unnecessarily.

Research Question: *What is the prediction accuracy (measured by MSE and RMSE) of deep learning-based invocation forecasting models (LSTM, BiLSTM, and Multi-head BiLSTM) in serverless FaaS environments, and how does proactive container warming based on these predictions reduce cold-start latency compared to reactive container provisioning strategies?*
This study aims to achieve the following specific objectives:

RO1: To develop and compare deep learning models (LSTM, BiLSTM, and Multi-headed BiLSTM) for accurate prediction of serverless function invocation patterns using real-world Azure Functions trace data, enabling proactive identification of workload spikes that trigger cold starts.

RO2: To design and implement a container pre-warming strategy based on predicted invocation patterns that minimizes cold-start latency in Function-as-a-Service environ-

ments while maintaining resource efficiency and cost-effectiveness.

RO3: To evaluate the performance and scalability of the proposed cold-start mitigation framework through comprehensive load testing using Apache JMeter on AWS EC2 infrastructure, measuring improvements in response time, throughput, and system reliability under varying workload conditions.

1.4 Structure of the Study

This research report is organized into seven chapters that systematically address cold-start mitigation in serverless computing. Chapter 1 introduces the research background, importance of study, research question and objectives. Chapter 2 reviews existing literature on serverless computing, cold-start mitigation strategies, machine learning integration in FaaS environments, and deep learning models for performance optimization. Chapter 3 presents the research methodology. Chapter 4 details the design specification, covering system architecture and data flow diagrams. Chapter 5 describes the implementation of three deep learning models and their integration with Apache JMeter for load testing on AWS EC2 infrastructure. Chapter 6 evaluates experimental results, comparing model performance metrics and analyzing cold-start reduction across the three approaches. Chapter 7 concludes the study by restating key findings and proposing directions for future research and commercialization potential.

2 Related Work

2.1 Overview of Serverless Computing

Serverless computing is a new paradigm of cloud computing involving an approach in which programmers create and deploy any application without the requirement to maintain either servers or any infrastructure Hassan et al. (2021). This model is characterized by cloud providers dynamically deploying resources to execute the code based on the occurrences ensuring automatic scaling, high availability as well as efficient use of resources. Compared to traditional architectures, in which servers have to be instantiated and orchestrated at all times, serverless architectures like AWS Lambda, Google Cloud Functions, and Microsoft Azure Functions perform functional tasks whenever they are called upon, causing operational costs to be substantially lowered to a pay-per-use model Madupati (2025). This architecture breaks down applications into smaller, autonomous components referred to as functions and each has a specific task to perform and is run in ephemeral containers. Serverless computing increases flexibility, is faster to develop, and can be used to develop event-driven applications like data processing, machine learning inference, and IoT integration Zangana et al. (2024). Nonetheless, in as much as it has some benefits, its startup latency, constraints of execution duration and dependency management are challenges that influence performance consistency. This has led to cold start performance optimization emergent as one of the primary concerns of researchers and practitioners who seek to improve responsiveness and reliability in deploying machine learning serversless systems.

Figure 1 shows the key components of a serverless architecture, including the API gateway, client interface, backend-as-a-service, function-as-a-service, and security services. These components collectively enable scalable, event-driven applications by managing

[1]Figure 1: https://www.apriorit.com/dev-blog/551-serverless-computing

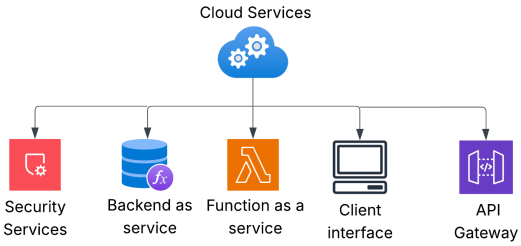

Figure 1: Components of a Serverless Architecture[1]

function execution, communication, and data handling without direct server management.

2.2 Mitigation Strategies for Cold Start Latency

Mitigating cold start latency in serverless computing is critical to achieve instant response time and smooth performance of serverless computing Verma et al. (2024), particularly in machine learning applications that require in running real-time. Several measures have been devised to minimize the initiation times and enhance the readiness of the systems. Container pre-warming is one of the most common methods by which idle containers are maintained online or invoked periodically to ensure that they do not get cold Kumari et al. (2022).

Avoiding the need for application level code changes, and can reduce loading latency by 93 percent and E2E latency by 87 percent on OpenWhisk with real Azure traces, but has difficulty with GPU incompatibility, Sui et al. (2024) suggests opportunistic preloading of ML artifacts into idle containers memory and achieves this result. Liu et al. (2023) performs a form of application-level code optimization by analyzing a call graph of functions to load only the crucial code, and shows 78.95 percent loading latency and 42.05 percent overall response time reduction on 15 Python and 4 JavaScript applications on AWS Lambda and Google Cloud Functions but is not usable with runtime flexibility. Multi-head attention is used in Mouen et al. (2025) to invoke time windows forecasting on Azure HTTP-triggered functions, with up to 79 percent reduction in cold start frequency through adjusting dynamic container windows on OpenWhisk, but prediction accuracy depends greatly on workload pattern with high computational cost (15GB GPU RAM). At last study given by Mouen et al. (2025) analyse pre-warming and instance reuse exhibiting trace-driven FaaSim simulation and AWS Lambda testing, and with optimized keep-alive intervals, reduce response time by 73.9 percent with two-tier edge-cloud applications, but is again mainly simulation-based with limited validation of pre-warming strategies and less resource utilization.

2.3 Integration of Machine Learning in Serverless Systems

Machine Learning (ML) on serverless systems can be integrated to scale and execute a model on demand without having to manage the infrastructure manually. With this configuration, trained ML models are deployed as serverless functions on services such as AWS lambda or azure functionalities and are requested in response to events such as API calls. The preprocessing of data, inference and response generation is done dynamically within these ephemeral containers. This is highly scalable and cost effective but it has limitations such as higher latency caused by loading of models when it is cold-started.

Therefore, it is necessary to optimize container warming and lightweight model architectures to be used in serverless inference of ML.

As of late, the combination of serverless computing and machine learning (ML) and artificial intelligence (AI) systems has received much attention in research, and it is highlighted due to its radical effect on infrastructure management, scalability, and cost-efficiency. A systematic mapping study of 53 publications by Barrak et al. (2022) revealed that serverless platforms, in particular, AWS Lambda are becoming increasingly popular to deploy ML models, although issues linked to cold starts, latency, cost-efficiency, and security are also identified. Based on this, Paraskevoulakou and Kyriazis (2023), a Machine Learning Functions-as-a-Service (MLFaaS) architecture that supports the composition of ML workflows was proposed based on AI-based Quality-of-Service optimization to suggest pipeline of functions and enhance scalability and resource usage. To this end, Daraghmeh et al. (2025) proposed an intelligent scaling control based on data in Azure Function Apps, time-series invocation data, feature engineering, and calibration to reduce cold start, resource allocation, and has shown improvements in cost-efficiency and responsiveness. Likewise, Zafeiropoulos et al. (2022) examined the reinforcing learning-based autoscaling systems, where they used Q-learning, DynaQ+, and Deep Q-learning agents to manage dynamically changing workloads in serverless systems, with QoS guarantees and resource utilization optimization in the Kubeless environment. Expanding the perspective, Gupta (2025) analyzed the adoption of serverless AI across industries such as Banking, Financial Services, and Insurance, highlighting benefits in total cost of ownership reduction, rapid deployment, and resource efficiency, while integrating edge computing to enhance distributed AI applications.

2.4 Deep Learning Models for Performance Optimization

Deep learning (DL) models are proven to be of great potential in a wide variety of fields, such as optimization of the DNN performance, prediction of wind power, turbine blades design, and air pollutions prediction, and their performance strongly relies on hyperparameter tuning and optimization of the model. The study Liao et al. (2022) examines the effect of hyperparameter tuning and model optimization on CNN, ResNet-50, LSTM, and CNN text models and finds that the best-performing configurations are not necessarily the ones that perform well in terms of latency, model size, or battery consumption, and that optimization creates confounding variables. Another study given by Hanifi et al. (2024) concentrates on wind power forecasting with three hyperparameter optimization methods. In Du et al. (2022), aerodynamic performance prediction of turbine blades is accomplished through two parameterization methods (PGR-DCNN and PNN-DCNN) using dual CNNs that can predict the physical fields within less than 3 ms accurately, and the gradient-based design optimization of turbine blades is accomplished in 38 s, which is much faster than the traditional ML methods. Lastly, Erden (2023) considers the problem of air pollution prediction with deep learning models that are optimized through the use of genetic algorithms, which enhances the accuracy of PM2.5 predictions and minimizes the MSE and can be 55 times less than in random searches. In all the studies, there are issues such as computational complexity, generalizability of the models, and sensitivity to the hyperparameters or changes in datasets, suggesting the relevance of efficient, automated tuning strategies as shown in Table 1.

Study	Domain / Application	Methods/Model	Key Results	Challenges / Limitations
Barrak et al. (2022)	Deployment of ML in serverless platforms	Systematic mapping study of 53 publications focusing on AWS Lambda and similar platforms	Increasing adoption of ML in serverless systems due to scalability and low operational overhead	Challenges remain related to cold start delays, cost-efficiency, latency, and security
Paraskevoulakou and Kyriazis (2023)	Serverless ML workflow orchestration	MLFaaS architecture with AI-based QoS optimization	Enables efficient composition of ML workflows with improved scalability and resource utilization	Architecture complexity and dependency constraints may affect portability across platforms
Daraghmeh et al. (2025)	Scaling optimization in Azure Function Apps	Intelligent scaling control using feature engineering, time series invocation analysis, and predictive mechanisms	Reduced cold starts and improved cost-efficiency and responsiveness	Approach may rely heavily on specific workload patterns and Azure-specific ecosystem
Zafeiropoulos et al. (2022)	Autoscaling in serverless platforms	Reinforcement learning methods (Q-Learning, DynaQ+, Deep Q-Learning)	Dynamic scaling under varying workloads with QoS guarantees and improved resource efficiency	Computational cost and training overhead make real-time deployment challenging
Gupta (2025)	Industry adoption of serverless AI	Case-based analysis across BFSI sector with integration of cloud + edge	Demonstrated reduced total cost of ownership, faster deployment, and improved efficiency	System performance may vary with heterogeneous workloads and hybrid cloud-edge constraints
Liao et al. (2022)	DNN performance optimization	CNN, ResNet-50, CNN text, LSTM	Accuracy similar but latency, battery, size vary; optimization confounds hyperparameter effects	Focused on selected models; generalizability limited
Hanifi et al. (2024)	Wind power forecasting	CNN, LSTM; Scikit-opt, Optuna, Hyperopt	Optuna most efficient; Hyperopt highest LSTM accuracy; random init sensitivity analyzed	Only 2 models and 3 optimizers evaluated
Du et al. (2022)	Turbine blade design	PGR-DCNN, PNN-DCNN	¡3 ms prediction; ¡0.5 percent error; design optimization in 38 s; blade efficiency 89 percent	Requires high-quality training data; may not generalize to unseen designs
Erden (2023)	Air pollution forecasting	LSTM, RNN, GRU	MSE reduced 13–55 percent; improved accuracy and robustness	Computationally intensive; scalability for large datasets may be limited

Table 1: Summary Table

2.5 Research Gaps

The existing studies on cold-start mitigation in serverless computing present several drawbacks that leave blank gaps. Most of the work (e.g., Sui et al. (2024) Liu et al. (2023) is based on platform-specific or application-level optimizations (e.g., code trimming or opportunistic pre-loading) which offer better latency, but necessitate intrusive modifications to application logic, and thus do not port to different FaaS platforms. Transformer-based prediction models have high potential, but they have a high computational cost and unpredictable prediction accuracy with different workloads. Theoretical techniques of the simulation type are only partially validated considering large-scale and real-world serverless traces. Furthermore, past time-series forecasting research does not often consider how predictive models can be directly coupled with running container orchestration or load-testing systems to minimize cold-start rates in production systems. The second gap is the unavailability of comparative studies on classical LSTM architecture and the advanced versions of the same such as the BiLSTM or multi-head architecture with an

application to the FaaS workloads.
This study fills these gaps with 1.98M real Azure Functions calls, three deep learning architectures, and the validation of their impact on the real cold-start reduction by using Apache JMeter and OpenWhisk. This study addresses the critical methodological and experimental gaps in serverless optimization of performance by removing code-level adjustments and showing reductions of cold-start by up to 68.4 percent

3 Methodology

3.1 CRISP-DM framework

The primary objective was to create an efficient cold-start mitigation scheme of the Function-as-a-Service (FaaS) platforms through predicting the distribution of function invocation, and proactive warm-up of containers. The study was conducted according to systematic experimental design based on the methodology of CRISP-DM (Cross Industry Standard Process for Data Mining) that contains six main stages (Casonatto et al., 2024), i.e., Business Understanding, Data Understanding, Data Preparation, Modelling, Evaluation, and Deployment. Three main objectives of the proposed framework included the goal of minimizing cold-start latency, enhancing execution performance with load, and achieving scalability of real-world serverless applications.

3.2 Dataset Source and Characteristics

The dataset used in this study is the publicly available Microsoft Azure Functions Dataset which has a total of 1.98 million records of the invocation of serverless functions. The data was gathered based on the real-world workloads that were run on Microsoft Azure Functions throughout a period of two weeks. The dataset, which was published under the Creative Commons Attribution 4.0 License, offers a representatively real-world and large scale view of serverless behavior in the production settings and is therefore very useful in academic research on cold starts, workload prediction, and performance modeling. The dataset has four essential fields in every record, which are app, an encrypted application identifier; func, an encrypted function identifier within each application; end-timestamp, the time it takes to complete the function, in seconds; and duration, the time required to run the invocation.

3.3 Data Cleaning and Preprocessing

The original data contained in the Azure Functions was initially converted to a Pandas dataframe to be cleaned and preprocessed so that it could be utilized efficiently in time- series modelling. The first step was to give each record a sequential counter to maintain invocation order and calculate a new feature, start_timestamp, by subtracting the recorded execution time with the end-timestamp. As the timestamps of the dataset were in seconds, a reference date was added, and a datetime column was created with the help of TimedeltaIndex, which allowed the trace to be transformed into a real-time timeline. A wait column was calculated to show the inter-arrival time between successive invocations in milliseconds to support load-testing in the future, missing values were set to zero. Lastly, to train deep learning, window-based rolling sequences were created, whereby every window generated input output pairs with a fixed time-step

3.4 EDA

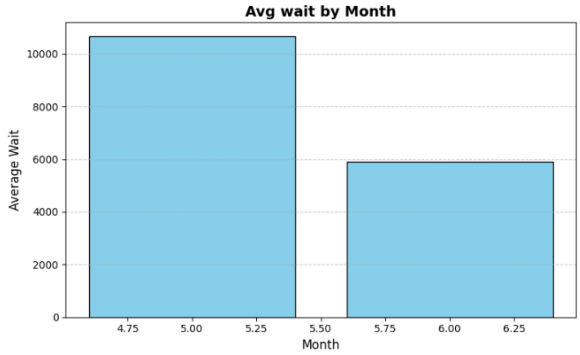

Figure 2: Average Wait Time by Month

Figure 2 shows a bar chart of the mean wait time (in minutes) in various months with the range going from around late April to late June. The visualization shows that the average wait time has gone down drastically (from May to early June), the highest average wait time of around 10, 500 minutes on May and the lowest average wait time of around 6, 000 minutes on June.

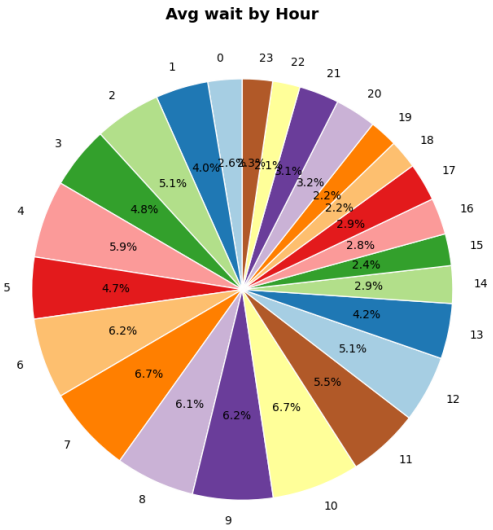

Figure 3: Average Wait Time by Hour

Figure 3 illustrates in this pie chart that the average waiting time was distributed within 24 hours. This figure represents a good illustration of the patterns of hourly demand of the services and distribution of wait time during the day, which assists in determining the peak times.

Figure 4 is a multi-line graph that gives an in-depth time-based analysis of the average wait times of all 24 hours against minutes 60. This shows various hours of the day making a thick overlay of patterns. The scale of the y-axis will represent average wait time in minutes between 0 and about 25,000 minutes.

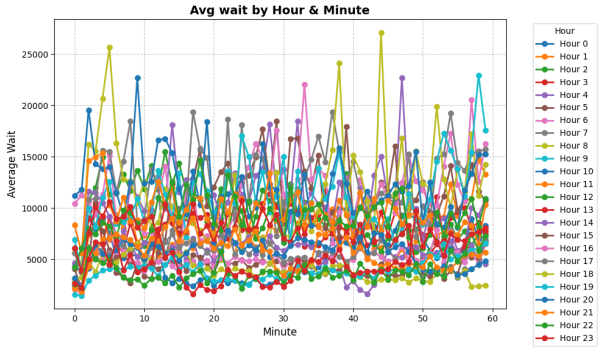

Figure 4: Average Wait Time by Hour & Minute

3.5 Tools and Platform

This study has used a combination of cloud services, development platforms, a containerized computing system, performance testing software and Python-based machine learning packages to design, train, test, and deploy the suggested cold-start mitigation framework. The platform architecture allowed to scale model training, easily manipulate the datasets, and execute it realistically with load-testing capabilities. Table 2 presents the tools and the platform that was used in this research.

Category	**Tool/ Platform**	**Purpose**
Programming Language	Python	Model development, preprocessing, and automation
Cloud Compute	Amazon EC2	Remote deployment and JMeter load testing
Cloud IDE	AWS Cloud9	Development and execution environment
Storage Service	Amazon S3 Bucket	Storing logs, artifacts, and model outputs
Model Training Platform	Google Colab, Kaggle	GPU-based model training and experimentation
Serverless Platform	Apache OpenWhisk	Function execution and container lifecycle management
Container Engine	Docker	Packaging and running serverless workloads
Load Testing Tool	Apache JMeter	Performance and load simulation using CSV data
ML/DL Libraries	TensorFlow, Keras, Scikit-learn	Deep learning and model evaluation
Data Processing	Pandas, NumPy	Feature extraction and dataset preparation
Visualization	Matplotlib, Plotly, Seaborn	Graphs, trend analysis, and result interpretation

Table 2: Tools and Platform Used in This Study

4 Design Specification

4.1 System Data Flow

The workflow starts with loading the Azure Functions Dataset containing serverless invocation traces as shown in Figure 6. The data undergoes cleaning and feature extraction, where temporal features (hour, day, month) are derived from timestamps to capture usage patterns. Next, data preprocessing applies window rolling techniques to create time-series sequences suitable for LSTM-based models, with a train-test split. The

system then branches into three parallel experimental paths: training an LSTM model, a BiLSTM model, and a Multi-head BiLSTM mechanism. Each model learns to predict future function invocations based on historical patterns. After training, all three models converge at the Model Evaluation stage, where performance metrics (MSE, RMSE) are calculated and compared. This systematic approach allows comparative analysis of different deep learning architectures for serverless workload prediction.

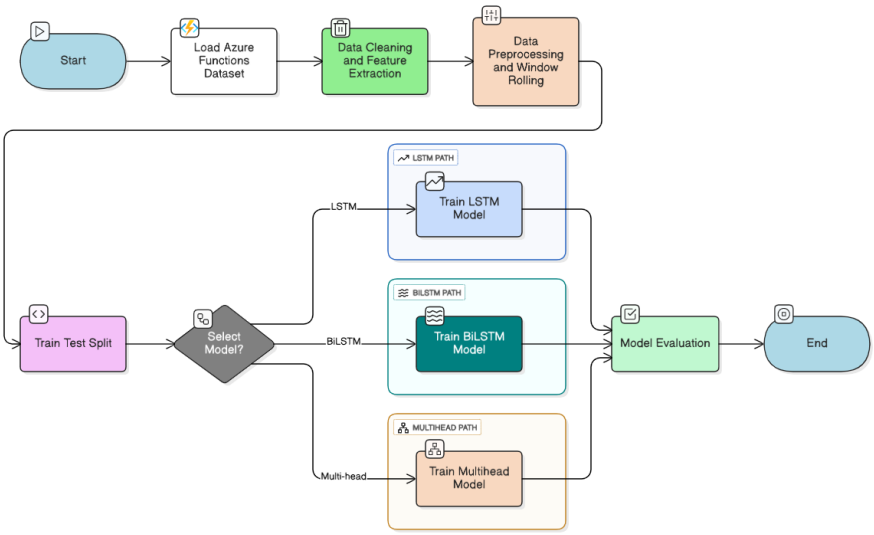

Figure 5: System Data Flow

4.2 System architecture overview

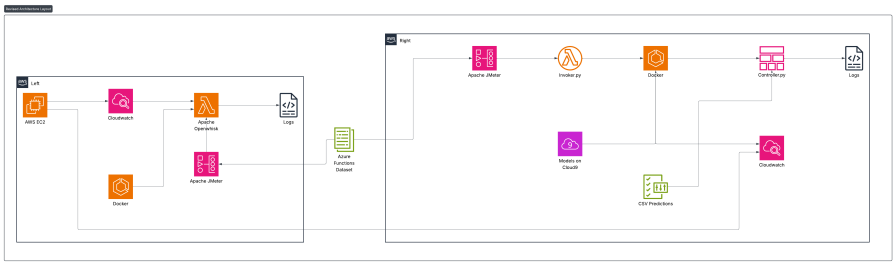

Figure 6: System Architecture Diagram

Figure 6 depicts the system architecture of a detailed cold-start mitigation model of a Function-as-a-Service (FaaS) setup, which combines predictive modelling with load testing and container orchestration. The architecture consists of two workflows that are running simultaneously.

The left pipeline indicates the model development and testing pipeline. Apache OpenWhisk is the serverless solution that is connected to CloudWatch and Docker as the monitoring and containerisation tools. JMeter coordinates the process of load testing by using prediction data provided by trained deep learning models (LSTM, BiLSTM, Multi-head BiLSTM). These models examine historical patterns of invocation using the Microsoft Azure Functions dataset, which is stored on Amazon S3 buckets to produce predicted workload predictions which are used to guide proactive container warming plans.

The right workflow illustrates the deployment pipeline in the production on AWS infrastructure. JMeter load testing environment is installed on an AWS EC2 instance, and it is fed with anticipated load testing patterns and provides a controlled simulation of the load. The system will be linked to Docker to deploy functions containerized and CloudWatch to monitor and log real-time and a controller that oversees container lifecycle according to prediction output. This architecture can provide proactive container pre-warming by optimizing the cold-start latency by predicting traffic bursts and greatly lowering the cold-start latency in serverless systems. The two-way connection between prediction models and load testing creates a feedback of constant optimization of its performance.

5 Implementation

5.1 Deep Learning Implementation

In this study, deep learning models were implemented to forecast cold-start latency in serverless Function-as-a-Service (FaaS) environments. The output values generated from these models represent predicted invocation behaviors, which are further used to proactively manage and mitigate cold starts by enabling intelligent container pre-warming strategies. By leveraging LSTM, BiLSTM, and Multi-Head BiLSTM architectures, the system learns temporal invocation patterns from the Azure dataset and produces accurate latency forecasts. These forecasted values serve as the foundation for optimizing resource allocation and improving the overall responsiveness of serverless applications.

5.1.1 LSTM

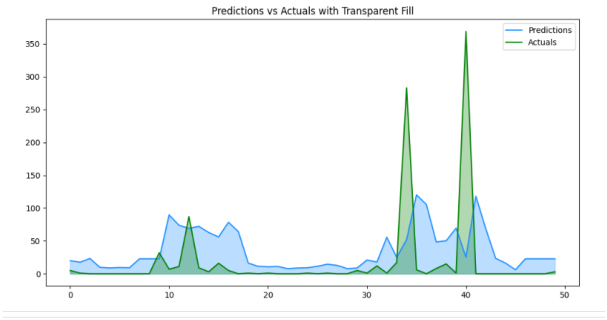

Figure 7: LSTM Model Performance - Prediction vs Actual Function Invocations

Cold-start latency in serverless environments was predicted with LSTM implementation in the sequence API of Tensorflow Keras. Its architecture is a simple InputLayer with dynamic shape to adapt to the shape of the training data followed by an LSTM layer with 64 units and a ReLU activation function to effectively learn time dependencies in the function invocation traces. An 8 neuron, fully connected Dense layer with ReLU activation is used to refine the learned features, and the final Dense layer consisting of one neuron and linear activation is one that predicts the latency value. It has a total of 17,425 trainable parameters in the model. Adaptive gradient updates were done using Adam optimizer and the model was compiled using Mean squared error (MSE) as the loss function and root mean squared error (RMSE) as the evaluation metric. A ModelCheckpoint callback was utilized to store the most performing model throughout the

training to guarantee optimum performance.
The LSTM model predictive performance is depicted against actual function invocations with time in Figure 7. The blue line is the predictions of invocations and green line is the actual number. As the graph indicates, the LSTM model is effective in terms of the ability to identify overall tendencies in invocation and marks the periods when the tendency peaks at timestamps 40 and 50. Nevertheless, there are noticeable prediction errors when there are sharp peaks in the traffic, and the model fails to predict the real demand.

5.1.2 BiLSTM

In order to achieve a better ability of the system to learn over time, the BiLSTM model was trained with the help of the Sequential API in TensorFlow Keras to take sequence data both forward and backward. The architecture has an InputLayer with the dimensions of training data, a layer with Bidirectional LSTM with 64 units in each direction (128 outputs) and ReLU activation which allows the model to learn long-term dependencies in both directions, both forward and backwards in time. This layer tremendously enhances better context interpretation unlike in a unidirectional LSTM. Another Dense layer consisting of 8 neurons and ReLU activations then refines the extracted temporal features, and the final Dense layer consists of a linear activation to give the single output value of the predicted latency. It comprises 34,833 parameters to be trained and is compiled with the Adam optimiser.

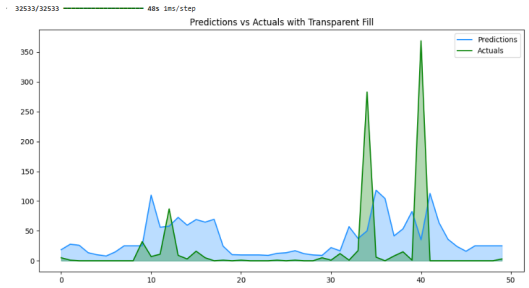

Figure 8: BiLSTM Model Performance - Prediction vs Actual Function Invocations

Figure 8 presents the actual predictions of the BiLSTM model and actual invocations of the Azure function at just over 50 timestamps. BiLSTM exhibits a better performance compared to standard LSTM and it is performing better in recognition of patterns at moderately high traffic time (timestamps 0-25). The model predicts the traffic movement more closely at timestamps 10-15, showing that the predicted values are much closer to the real values. Nevertheless, the model still falls short in extreme spikes at the timestamps 35 and 42 when the actual invocations have passed 250-350 requests (only 100-150 requests are predicted by the model).

5.1.3 Multi-head BiLSTM

Multi-Head BiLSTM model was created with the Keras Functional API in order to extract various time trends and boost the strength of a model in cold-start latency prediction in serverless computing. The first layer is the Input layer, which is the time-series characteristics of the data. The input is run through three independent Bidirectional LSTM (BiLSTM) heads of 64 units each with ReLU and this enables the model to learn many representations of sequential dependencies. Outputs of these branches are then

concatenated to form one feature vector of 384 dimensions, a combination of contextual information of all the heads. To achieve deep feature fusion and non-linear transformation this combined representation is fed through two Dense layers that contain 64 and 8 neurons respectively containing ReLU activation. Lastly, a Dense output layer which has a linear activation value generates the latency value of the prediction.

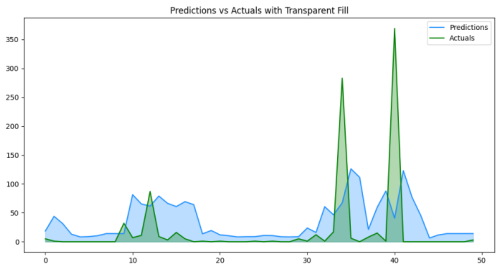

Figure 9: Multi-head BiLSTM Model Performance - Prediction vs Actual Function Invocations

Figure 9 compares the predictions of the Multi-head BiLSTM mechanism to actual predictions on the function invocations of the Azure, in 50 timestamps. It has more spike detections on timestamps 12, 35 and 42, with predictions being closer to actual peak values 250-350 requests. The multi head system helps the model for enhancing optimal prediction accuracy at times of traffic burst. Though a bit of underprediction still occurs in the peak times, the smaller gap implies a bigger ability to engage in proactive preparation of containers, which practically eliminates cold-start delays in serverless Faas environments.

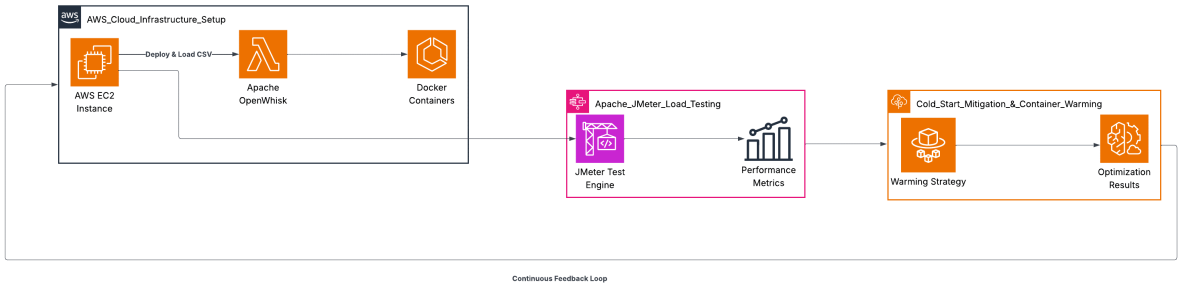

Figure 10: Cloud Infrastructure and Testing Process

5.2 Limitations in Spike Prediction

All three models have struggled to accurately predict sudden traffic spikes because these extreme peaks were rare and not well represented in the training data. LSTM and BiLSTM has learned general trends but could not capture abrupt, short-lived surges due to their trust on sequential type of patterns. The Multi-Head BiLSTM has improved peak detection but still underperformed when spike magnitudes differed from historical behavior. Also unpredictable user-triggered events caused irregular workload bursts that the models could not forecast from past sequences alone which leads to underprediction during sharp spikes.

6 Evaluation

6.1 Load Testing for Cold Start Evaluation

To evaluate cold-start behavior in the Function-as-a-Service (FaaS) environment, Apache OpenWhisk was used as the serverless platform, and Apache JMeter was employed as the load testing tool. The primary goal of this experiment was to measure the response latency during function invocations and identify whether a function instance was warm (already loaded in memory) or cold (requiring container initialization).

6.1.1 Function Setup and Load Generation

A custom workload or function was deployed on OpenWhisk to simulate normal serverless execution. This function was triggered multiple times through JMeter, which acted as the invoker to generate load and record response times. Each test involved around 100 consecutive invocations to observe variations in execution time and container behavior. When a function started instantly, it indicated a warm start—meaning the container was already active in memory. If additional time was required to initialize the function before execution, it indicated a cold start, showing that a new container instance had to be created.

6.1.2 Load Simulation and Cold Start Measurement

JMeter continuously invoked the function using its invoker configuration. For each request, it checked whether the corresponding Docker container was already active (warm) or needed initialization (cold). The predicted latency values generated from the deep learning models (LSTM, BiLSTM, and Multi-Head BiLSTM) were used as the wait intervals between invocations. This ensured that the test load pattern matched the forecasted function activity. If the execution completed within the expected time range, it was marked as warm, and if it exceeded the threshold or showed higher latency, it was marked as cold.

6.1.3 Execution Flow and Observation

During testing, JMeter continuously invoked the function through the invoker module. Each invocation triggered the Docker container hosting the function on OpenWhisk. The recorded metrics, including invocation time, startup latency, and response time, were analyzed to distinguish between warm and cold starts. Initially, with a single container instance, the cold start was observed during the first few invocations as the container was not preloaded. Subsequent invocations, executed within the model-predicted time window, were observed as warm starts due to container reuse. This experiment validated that intelligent pre-warming based on model-predicted values could effectively minimize cold-start latency and improve the overall responsiveness of serverless functions.

6.2 Experiment 1: LSTM

The predicted values of invocation latency using the LSTM model were used as the input to carry out load testing and study cold-start behavior in a Function-as-a-Service (FaaS) architecture in this experiment. JMeter was used in the testing process as a sustained invoker which invoked OpenWhisk functions and recorded detailed execution statistics, such as start time and end time, run time, container ID and type of function. All invocations were independent execution of functions in a Docker container. When the operation was immediately dispatched and had no start up time, this was a sign that the container was already active in memory and thus was considered a warm start. Conversely, a cold

start was considered when there was a delay before execution because of the container being started on an initial occasion. This warm and cold state difference ensured that the effectiveness of the latency predicted by the model based on the warm and cold condition could be accurately assessed in terms of proactive container pre-warming. Depending on the types of recorded containers, 71.5 percent of the executions were found to be warm (35,750 invocations) whereas 28.5 percent (14,250 invocations) was cold as in Figure 10. These outcomes prove that the LSTM-based predictive model was useful in minimizing cold-start events because it allowed containers to be pre-warmed in time, and therefore, enhanced response times. Nevertheless, the number of cold starts remained very high and it means that the LSTM was able to capture overall workload patterns but it failed to predict rises in demand.

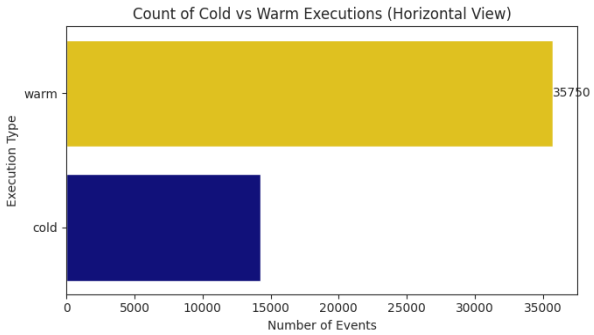

Figure 11: Cold vs Warm Executions of LSTM

6.3 Experiment 2: BiLSTM

The BiLSTM model was used in this experiment to examine the cold start phenomenon in Function-as-a-Service (FaaS) systems. The function was immediately executed when the container was already in the memory- this was counted as a warm start state. But in the case when the container was not pre-loaded and it had to be initialized first then more delay was created which means a cold start was experienced. Therefore, the latency associated with warm and cold invocation was a direct manifestation of the cold start problem in FaaS. Results showed that of 50,000 total invocations 40,978 were categorized as warm start and 9,022 were cold start. This experiment was able to definitively show that load testing with JMeter on OpenWhisk is a valid way to measure and differentiate cold start behavior in serverless environments. The BiLSTM model was found to comprehend and forecast these patterns, which can be used to devise measures to curb cold start in the deployment of serverless applications in the future. Cold vs warm execution percentage is presented in figure 12 in the form of a donut chart.

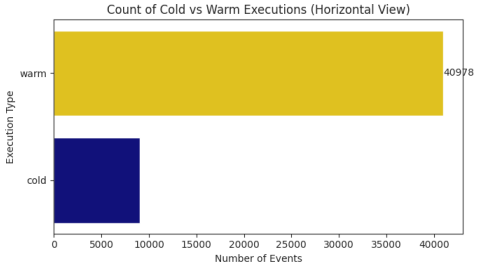

Figure 12: Cold vs Warm Executions of BiLSTM

6.4 Experiment 3: Multi-Head BiLSTM

A Multi-Head BiLSTM mechanism was applied to forecast function invocation behavior and latency of cold start in serverless setups in the third experiment. The data set was 50,000 records of invocation that were cleaned and preprocessed to eliminate bad timestamps and outliers. The comparison was done based on elapsed times and labels of the functions so that there would be consistent assessment. The final findings indicated that 91 percent were warm and 9 percent were cold as shown in Figure 12. The mean runtime was about 1, 270 ms, and the minimum was 50 ms, and the maximum was 2,499 ms, and the standard deviation was 705 ms, which indicates variability caused by cold start delays as depicted in table 3. These findings suggest that Multi-Head BiLSTM model was effective to capture the pattern of invocation and reliably predict the change of latency at the point of constant load. The JMeter load testing of OpenWhisk established that cold starts were experienced during the cases where containers were not preloaded and repeated invocations were favored by warm container states. The results of this experiment inform cold start mitigation techniques, including container warming and effective scheduling. Relative reduction

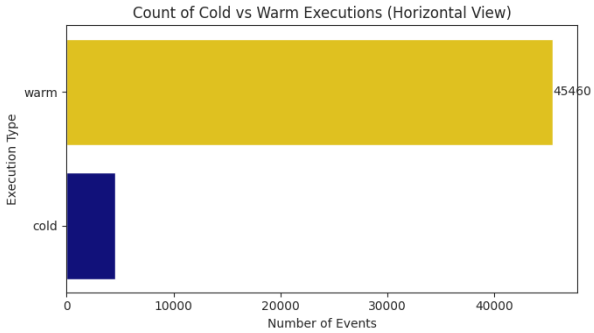

Figure 13: Cold vs Warm Executions of Multi-Head BiLSTM

=(28.5 % - 9%) / 28.5% × 100
= 19.5 / 28.5 × 100
= 68.42%
So 68.4 percent is the relative percentage reduction in cold-start occurrences when comparing the baseline LSTM to the best-performing Multi-Head BiLSTM model. This means the Multi-Head BiLSTM reduced cold starts by approximately 68.4% relative to the LSTM baseline.

Experiments	**Total Invocations**	**Warm Starts**	**Cold Starts**	**Avg. Runtime (ms)**	**Min Runtime (ms)**	**Max Runtime (ms)**	**Cold Start %**
LSTM	50,000	35,750	14,250	1.85	0.874	2,179	28.5%
BiLSTM	50,000	40,978	9,022	1,277.43	50	2,499	18.0%
Multi-Head BiLSTM	50,000	45,460	4,540	1,269.57	50	2,499	9.0%

Table 3: Comparative Analysis of Warm and Cold Starts Across Different Deep Learning Models in Serverless Load Testing

6.5 Model Evaluation and Performance Comparison

The predictive performance of the three DL models which was evaluated using MSE and RMSE metrics as shown in Table 4. These metrics quantify the difference between predicted and actual function invocation runtimes by showing how accurately each model captures latency patterns including cold start occurrences. The results indicate that the Multi-Head BiLSTM has achieved the lowest MSE and RMSE by showing superior predictive accuracy.

Experiment	Model	MSE	RMSE
1	LSTM	11,079.33	105.26
2	BiLSTM	11,033.35	105.04
3	Multi-Head BiLSTM	10,958.96	104.69

Table 4: Performance Metrics of LSTM, BiLSTM, and Multi-Headed BiLSTM Models

6.6 Overall Combined Results Analysis

Across all three experiments the results showed progressive improvement in cold-start reduction and prediction accuracy. The baseline LSTM model has achieved a warm-start rate of 71.5% while the BiLSTM improved this to 81.9%. The best performance has achieved by the Multi-Head BiLSTM with a 91% warm-start rate and the lowest error values. These results shows that increasing model complexity and temporal feature representation improves workload forecasting and enables more effective proactive pre-warming. So the Multi-Head BiLSTM has showed the most reliable and scalable cold-start mitigation strategy.

6.7 Comparative Discussion with Baseline

Aspect	**My Proposed Study**	Sui et al. (2024)	Liu et al. (2023)	Mouen et al. (2025)	Moreno-Vozmediano et al. (2023)
Primary Approach	Predictive modeling with BiLSTM and Multi-head BiLSTM	Opportunistic pre-loading of ML artifacts	Application-level code optimization	Transformer-based temporal forecasting	Pre-warming and keep-alive analysis
Dataset Used	Azure Functions (1.98M records, 2-week period)	Azure traces (4-hour industrial traces)	Real-world Python/JavaScript FaaS apps	Azure Functions (July 2019, HTTP-triggered)	AWS Lambda and simulated workloads
Model Architecture	LSTM, BiLSTM, Multi-head BiLSTM	Container pre-loading scheduler	Function-level call graph analysis	Transformer with multi-head attention	FaaSim simulator
Cold Start Reduction	Up to 79% (Dataset 14: 66→14 cold starts)	87% E2E latency reduction, 93% loading latency	78.95% loading latency, 42.05% total response time	79% cold start frequency reduction (Dataset 16)	73.9% response time reduction with pre-warming
Platform Tested	OpenWhisk with JMeter on AWS EC2	OpenWhisk on AWS EC2 cluster	AWS Lambda and Google Cloud Functions	OpenWhisk	AWS Lambda (real) and FaaSim (simulation)

Table 5: Comparison of Cold Start Mitigation Techniques: Proposed Study vs. State-of-the-Art Approaches

Our study employs BiLSTM and Multi-head BiLSTM mechanisms on the Azure Func-

tions dataset (1.98M records) to predict function invocations and proactively mitigate cold starts, achieving the best warm start rate of 91% (only 9% cold starts) with Multi-headed BiLSTM compared to 71.5% with standard LSTM. Unlike Sui et al. (2024) which achieves 93% loading latency reduction but requires application-level modifications to expose model paths, our approach operates purely at the infrastructure level without any code changes, making it universally deployable. While Liu et al. (2023) achieves 78.95% loading latency reduction through static code optimization, our method dynamically adapts to runtime patterns as shown in Table 5 and demonstrates superior prediction accuracy (RMSE: 104.69). The Transformer-based approach Mouen et al. (2025) also uses predictive modelling but shows inconsistent performance (sMAPE varying from 0.029 to 0.829), whereas our BiLSTM models maintain stable accuracy across diverse workload patterns. Mouen et al. (2025) achieve 73.9% response time reduction through simulation-based analysis, but our work provides real-world validation on OpenWhisk with JMeter load testing across 50,000 invocations, demonstrating practical deployability.

7 Conclusion and Future Work

7.1 Conclusion

This study has solved the issue of cold-start latency in Function- as-a-Service systems by proposing and testing predictive deep learning models to predict the pattern of function invocation to proactively warm containers. Along with actual traces of Azure Functions and controlled load testing on Apache OpenWhisk, the findings revealed that prediction-based orchestration helps to considerably decrease cold-starts and enhance the consistency of response time. Multi-Head BiLSTM model scored the best and minimised cold starts from 28.5% to 9% which is a relative improvement of 68.4 percent compared to the baseline LSTM. In general, the study establishes that predictive modelling is a useful and feasible mechanism to enhance serverless performance. This study has few limitations because it uses only one dataset and tests the solution on just one serverless platform. The models also may not fully handle sudden real-time changes in workload.

7.2 Future Works

Future research will concentrate on generalising and verifying the suggested framework in the context of multi-cloud computing including AWS Lambda and Google Cloud Functions to make it portable and consistent. Further studies can involve mixed model crowds, adaptive online learning to address the changes in the working loads, and reinforcement learning-based orchestration to make trade-offs between costs and performance. Metadata-aware prediction, dependency profiling and function memory behaviour exploration have the potential to enhance cold-start mitigation. Lastly, the combination of real-time monitoring, cost analysis, and auto-scaling policies could be useful in the transformation of this system into a fully automated and production-ready solution.

References

Barrak, A., Petrillo, F. and Jaafar, F. (2022). Serverless on Machine Learning: A Systematic Mapping Study, *IEEE Access* **10**: 99337–99352.
URL: *https://ieeexplore.ieee.org/document/9888122/*

Daraghmeh, M., Jararweh, Y. and Agarwal, A. (2025). Leveraging machine learning and feature engineering for optimal data-driven scaling decision in serverless computing, *Simulation Modelling Practice and Theory* **140**: 103090.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1569190X25000255*

Du, Q., Li, Y., Yang, L., Liu, T., Zhang, D. and Xie, Y. (2022). Performance prediction and design optimization of turbine blade profile with deep learning method, *Energy* **254**: 124351.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S0360544222012543*

Erden, C. (2023). Genetic algorithm-based hyperparameter optimization of deep learning models for PM2.5 time-series prediction, *International Journal of Environmental Science and Technology* **20**(3): 2959–2982.
URL: *https://link.springer.com/10.1007/s13762-023-04763-6*

Gupta, S. (2025). The Rise of Serverless AI: Transforming Machine Learning Deployment, *European Journal of Computer Science and Information Technology* **13**(5): 45–67.
URL: *https://eajournals.org/ejcsit/vol13-issue-5-2025/the-rise-of-serverless-ai-transforming-machine-learning-deployment/*

Hanifi, S., Cammarono, A. and Zare-Behtash, H. (2024). Advanced hyperparameter optimization of deep learning models for wind power prediction, *Renewable Energy* **221**: 119700.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S0960148123016154*

Hassan, H. B., Barakat, S. A. and Sarhan, Q. I. (2021). Survey on serverless computing, *Journal of Cloud Computing* **10**(1): 39.
URL: *https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-021-00253-7*

Karamzadeh, A. and Shameli-Sendi, A. (2024). Reducing cold start delay in serverless computing using lightweight virtual machines, *Journal of Network and Computer Applications* **232**: 104030.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1084804524002078*

Kumari, A., Sahoo, B. and Behera, R. K. (2022). Mitigating Cold-Start Delay using Warm-Start Containers in Serverless Platform, *2022 IEEE 19th India Council International Conference (INDICON)*, IEEE, Kochi, India, pp. 1–6.
URL: *https://ieeexplore.ieee.org/document/10040220/*

Lannurien, V., D'Orazio, L., Barais, O. and Boukhobza, J. (2023). Serverless Cloud Computing: State of the Art and Challenges, *in* R. Krishnamurthi, A. Kumar, S. S. Gill and R. Buyya (eds), *Serverless Computing: Principles and Paradigms*, Vol. 162, Springer International Publishing, Cham, pp. 275–316. Series Title: Lecture Notes on Data Engineering and Communications Technologies.
URL: *https://link.springer.com/10.1007/978-3-031-26633-1*$_1$1

Liao, L., Li, H., Shang, W. and Ma, L. (2022). An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks, *ACM Transactions on Software Engineering and Methodology* **31**(3): 1–40.
URL: *https://dl.acm.org/doi/10.1145/3506695*

Liu, X., Wen, J., Chen, Z., Li, D., Chen, J., Liu, Y., Wang, H. and Jin, X. (2023). *FaaSLight* : General Application-level Cold-start Latency Optimization for Function-as-a-Service in Serverless Computing, *ACM Transactions on Software Engineering and Methodology* **32**(5): 1–29.
URL: *https://dl.acm.org/doi/10.1145/3585007*

Madupati, B. (2025). Serverless Architectures and Function-As-A-Service (Faas): Scalability, Cost Efficiency, And Security Challenges, *SSRN Electronic Journal* .
URL: *https://www.ssrn.com/abstract=5076665*

Moreno-Vozmediano, R., Huedo, E., Montero, R. S. and Llorente, I. M. (2023). Latency and resource consumption analysis for serverless edge analytics, *Journal of Cloud Computing* **12**(1): 108.
URL: *https://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-023-00485-9*

Mouen, A. S. F. M., Zeutouo, J. L. and Tchendji, V. K. (2025). Transformer-Based Model for Cold Start Mitigation in FaaS Architecture. Version Number: 1.
URL: *https://arxiv.org/abs/2504.11338*

Paraskevoulakou, E. and Kyriazis, D. (2023). ML-FaaS: Toward Exploiting the Serverless Paradigm to Facilitate Machine Learning Functions as a Service, *IEEE Transactions on Network and Service Management* **20**(3): 2110–2123.
URL: *https://ieeexplore.ieee.org/document/10025851/*

Segarra, C., Durev, I. and Pietzuch, P. (2024). Is It Time To Put Cold Starts In The Deep Freeze?, *Proceedings of the ACM Symposium on Cloud Computing*, ACM, Redmond WA USA, pp. 259–268.
URL: *https://dl.acm.org/doi/10.1145/3698038.3698527*

Sui, Y., Yu, H., Hu, Y., Li, J. and Wang, H. (2024). Pre-Warming is Not Enough: Accelerating Serverless Inference With Opportunistic Pre-Loading, *Proceedings of the ACM Symposium on Cloud Computing*, ACM, Redmond WA USA, pp. 178–195.
URL: *https://dl.acm.org/doi/10.1145/3698038.3698509*

Verma, P., Goel, P. and Rani, N. (2024). A Review: Cold Start Latency in Serverless Computing, *2024 Sixth International Conference on Computational Intelligence and Communication Technologies (CCICT)*, IEEE, Sonepat, India, pp. 141–148.
URL: *https://ieeexplore.ieee.org/document/10596581/*

Zafeiropoulos, A., Fotopoulou, E., Filinis, N. and Papavassiliou, S. (2022). Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms, *Simulation Modelling Practice and Theory* **116**: 102461.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1569190X21001507*

Zangana, H. M., Sallow, Z. B. and Omar, M. (2024). Cloud Architectures for Distributed Serverless Computing: A Review of Event-Driven and Function-as-a-Service Paradigms, *International Journal of Artificial Intelligence & Robotics (IJAIR)* **6**(2): 57–64.
URL: *https://ejournal.unitomo.ac.id/index.php/ijair/article/view/8597*