National College of Ireland

Low-Latency Cold Start Optimization for Serverless Machine Learning Inference

MSc Research Project
MSc in Cloud Computing

Vyom Mehul Vora
Student ID: 23410698

School of Computing
National College of Ireland

Supervisor: Aqeel Kazmi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vyom Mehul Vora
Student ID:	23410698
Programme:	MSc in Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Aqeel Kazmi
Submission Due Date:	11/12/2025
Project Title:	Low-Latency Cold Start Optimization for Serverless Machine Learning Inference
Word Count:	7134
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:	Vyom Mehul Vora
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):	

Low-Latency Cold Start Optimization for Serverless Machine Learning Inference

Vyom Mehul Vora
23410698

Abstract

Serverless computing offers cost-efficient computation especially in the case of machine learning models but it has severe drawbacks, one of them being cold start problem. Model's initialization takes 5-15 seconds which can cause delays in inference. This research proposes a generalized framework that addresses the cold start problem through prewarming of serverless containers based on their usage pattern. Unlike existing solutions which are platform specific, our framework combines multi-cloud solution. The framework is evaluated on three serverless functions AWS Lambda, Azure Functions, Google Cloud Run using three diverse ML models including natural language processing model DistilBert, image classification model Resnet-50 and Random Forest Classifier. Experimental results shows an overall 30.9% reduction in cold start timings from 45.9% to 15.1% with prewarming containers and model achieving 65% prediction accuracy. Platform specific reductions were 44.4% on AWS, 23.1% on Azure and 11.1% on GCP. This research also contributes to reducing gap between platform specific optimizations and the need for generalized framework. The framework also eliminates the need for platform based solutions for machine learning model deployment and serving inference.

1 Introduction

1.1 Background of the Study

Serverless computing is dominating the market because of deploying application quickly with minimal management. Function as a Service(Faas) platforms such as AWS Lambda, Azure Functions and Google Cloud Run lets developers to focus on developing without managing infrastructure Shafiei et al. (2022). However, there is a challenge: cold start latency. When functions remain idle a lot of invocation requires containers, runtime initialization and dependency installation that gives delays of 2-5 seconds for traditional applications but when it comes to machine learning workloads the delays rises to 10-80 seconds. This latency makes serverless architecture unsuitable for real time applications such as fraud detection, weather app, autonomous vehicle which puts developers into dilemma of choosing between cost efficient serverless deployment or performance based persistent architecture. Cold start latency in serverless computing comprises of multiple overheads like container provisioning (200ms-2sec), runtime environment setup (300ms-1sec), requirement installations(500ms-2sec) and model loading (10sec-80sec). For machine learning workloads, model loading is the main bottleneck due to large pickle files. Unlike traditional serverless functions where object loading and library imports are the

main overheads, ML workload face different challenges from model size and framework dependencies. This creates unpredictable response times that impacts latency sensitive real time applications. Wu et al. (2022) Existing research on cold start optimization has given platform specific solutions that cannot generalized across cloud platforms Golec et al. (2024). Characterization studies states cold start behavior without providing predictive optimization strategies. Machine learning based approaches remain limited to single platforms either limited to basic features on Azure or deep learning models on OpenWhisk that cannot transfer to other cloud platforms. No existing work combines lightweight predictive models with cross platform generalization which suits for real world multi cloud deployments.

1.2 Motivation of the Study

This research addresses three critical gaps in existing research on cold start. First, existing solutions are platform specific that requires separate optimization strategies for AWS Lambda, Azure Functions and Google Cloud Run. This creates operational overhead for multi cloud deployments. Second, existing predictive solutions either use simple features that has less accuracy or have complex deep learning models that gives computation cost. Third, machine learning inference workloads are a use case for serverless computing but most of the existing research primarily focuses on application function optimizations without thinking of model loading overhead. This research addresses these gaps by developing a platform specific predictive prewarming framework using lightweight logistic regression machine learning model trained on invocation patterns. By using zero shot transfer across cloud providers, the framework does not require retraining for each cloud platform even while having good accuracy. The approach targets only ML inference workloads where cold start latency is a challenge in serverless computing.

1.3 Research Question and Objectives

This research addresses the question: **How can predictive prewarming help reduce cold start for machine learning workloads on serverless platforms?**

The main aim is to develop and evaluate a framework for predicting and mitigating cold starts in serverless machine learning inference deployments. This aim is supported by four objectives:

1. Deploy machine learning inference workloads (DistilBERT, ResNet-50 and Random Forest) across AWS Lambda, Azure Functions and Google Cloud Run to have a baseline cold start behavior.

2. Collect performance metrics including model characteristics like size, framework, model type then function usage patterns like invocation timing, frequency, idle periods and cold start latencies for functions.

3. Train a machine learning model using platform independent features to predict cold start probability of serverless functions.

4. Evaluate the framework across all three platforms and demonstrate real time cold start reductions.

1.4 Structure of the Report

The structure of this paper is as follows: Section 2: Related Work on serverless cold start optimizations, existing approaches including characterization studies, ml based prediction methods and optimization strategies, identify gaps in platform independent optimizations. Section 3 describes Methodology including research design data collection, feature engineering, model development and performance evaluation. Section 4 describes Design Specification covering system architecture, prewarm service working, framework integration across cloud platforms. Section 5 describes Implementation including serverless deployment configuration, data collection, model training flow, and validation experiment. Section 6 describe Evaluation of experiments conducted for this research and Section 7 describes key contributions, limitation and future work.

2 Related Work

2.1 Serverless Platforms Characterizations

Cloud platform characterization research provides baseline understanding of cold start behavior across serverless environments. Empirical evaluation research experimented across AWS Lambda and Azure Functions deploying classical algorithms to measure response time variations between the platforms Raj et al. (2024). Their findings shows no correlation between code size and cold start latency suggesting that factors other than code size are responsible for initialization overhead. The study reveals platform specific performance differences with Azure consistently showing higher cold start times than AWS and indicating that platform architecture and resource provisions vary a lot between providers. This research providers characterization without optimization strategies. The limitation to lightweight algorithm workload differs for machine learning inference where sequential request after function's idle state fails to capture production environment type traffic patterns without variation and burst requests. INFless addresses ML specific cold start challenges through a native serverless platform design for inference workloads Yang et al. (2022). The system provides unified resource between CPU and accelerators which achieves high throughput using built in batching mechanism. It manages cold start rates through batch queuing and execution time optimization. Evaluation shows INFless outperforms existing system by 2-5 times on throughput while having low latency. However, INFless requires platform level modifications not feasible for developers using AWS Lambda or Azure Functions.

2.2 Serverless Platforms Machine Learning Prediction

Machine Learning based approaches for proactive cold start predictions are emerging. Research on Azure Functions demonstrates linear regression models predicting cold start occurrences using features including time since last invocation and daily invocation frequency Kumar.N and Samy.S (2023). The predictive model gives binary classification of cold start events, giving preemptive identification of functions that are experiencing initialization delays. The research combines I/O throughput monitoring as an additional optimization dimension recognizing that I/O requirements impacts cold start duration for data intensive functions by significant margin. By monitoring Azure specific metrics through native platform services. The approach provides insights for developers to minim-

ize overhead through techniques like in-memory data structures and caching strategies. However, this work remains limited to Azure's ecosystem only utilizing two features without consideration of model characteristics such as size or framework. The absence of probability based thresholding and limitation to binary prediction restricts function deployments. Another research combines reinforcement learning with Long Short Term Memory(LSTM) to dynamically optimize container warm up strategies Vahidinia et al. (2023). This two layer framework does reinforcement learning to learn about temporal invocation patterns and determine idle container windows. It balances cold start reduction against memory consumption. The second layer utilizes LSTM time series forecasting to predict future invocation time by using prewarmed container counts. Evaluated on Apache OpenWhisk, this approach demonstrates 12.73% reduction in memory consumption and improving execution on prewarm containers by 22.65%. However, this framework relies on complex deep learning models that is a major bottleneck in terms of computation usage and having training complexity. Also, single platform evaluation on OpenWhisk cannot be a generalizable pattern that will work on other serverless architectures like AWS Lambda, Azure Functions or Google Cloud Run.

2.3 Container Management and Prewarming Strategies

Container level optimization approach addresses cold start through layer by layer caching and intelligent prewarming Yu et al. (2024). RainbowCake proposes layer wise container prewarming and keep alive techniques that mitigate cold starts with sharing awareness at minimal memory waste. The approach divides container bootup into three stages and manages different container layers individually. A sharing aware algorithm makes event driver layer caching decision in real time utilizing container sharing behind stand container techniques. Experiments on OpenWhisk clusters with real world workloads show RainbowCake reduces 68% functions startup latency and 77% memory waste. However, the approach focuses on container infrastructure optimization rather than predicting when cold starts will occur making it reactive approach rather than proactive. Another existing research uses application knowledge to reduce cold start frequency by treating serverless platform as black box Bermbach et al. (2020). This work approaches a lightweight middleware that uses knowledge of function composition to trigger function calls in workflows, the system proactively provisions container ahead of actual invocations. Experiments on AWS Lambda and OpenWhisk states that removal of 40% cold starts on average and 88% in some cases while only having little cost overhead. The approach differs from our work because it is relies on function's knowledge rather than historical invocation patterns. It is suitable for workflow designs but less applicable to standalone function deployments.

2.4 Static Deployment Optimizations

Static deployment optimization focuses on cold start latency through configuration selection at design time only. Research characterizing AWS Lambda demonstrates that deployment type(ZIP based vs Container based), memory allocation, language run time and package size impacts a lot on initialization times Dantas et al. (2022). Through extensive analysis across Python, NodeJs and Java runtimes with varying application sizes and other configurations. This work gives deployment guidelines for minimizing cold start delays based on application characteristics. Findings shows that machine learning model size affects cold start duration depending on deployment type with container

based deployments showing good performance compared to traditional ZIP deployments. While these insights gives developers to make deployment decisions that reduce initialization overhead. The approach is an assumption that workload characteristics has stable deployments for most applications. Static optimization selects best configuration once during application design but cannot adapt to dynamic invocation patterns that differs from time to time, day of week. For applications with very predictable and consistent workloads this would provide good reductions in overhead but with inconsistent loads, uncertain traffic bursts, fixed deployment configurations cannot warm containers before invocations. These approaches shows how to configure functions for minimal cold starts while predictive approaches determine when to activate those configurations.

2.5 Time Series Forecasting Methods

Time series forecasting offers an approach to predict prewarming through statistical modeling of historical invocation patterns. Research using Seasonal Autoregressive Integrates Moving Average(SARIMA) models forecast future function calls Lin (2024). By collecting historical data and determining optimal model parameters and training SARIMA models on temporal invocation sequences this approach gives dynamic resource allocation that significantly outperforms static threshold based methods in reducing cold start occurrences. The SARIMA framework captures seasonal patterns, trends and autocorrelated structure inherent in severeless workloads making it effective for applications with predictable temporal cycles such as daily or weekly traffic patterns. Experimental validation demonstrates measurable improvements in system responsiveness and resource utilization compared to scaling strategies. However, statistical time series model like SARIMA has constraints on practical deployment. First, they require large amount of historical data with consistent patterns. Performance starts degrading when there are irregular patterns. Second, SARIMA models area univariate, focusing solely on sequences without considering function specific characteristics such as model size, model framework, model dependencies that also impacts cold start. Third, the approach is platform specific, model trained on one serverless platform unable to use in Azure Functions or Google Cloud Run without having to retrain the model. While this work validates that predictive pre warming through forecasting is a proactive strategies, it is still not generalizable and misses factors that impacts cold start in serverless functions

2.6 Research Gaps and Contribution

The existing research on serverless cold start reveals three gaps that current solutions fails to address. First, while characterization studies provides valuable insights into cold start behaviour across cloud platforms but they are not predictive and has no proactive optimization strategies. Platform specific approaches like INFless and RainbowCake gives good results but require infrastructure level modifications that are difficult for any developer to setup. Second, machine learning approaches that predictions are limited to single platforms only either limited to Azure with minimal features or OpenWhisk with complex deep learning models that cannot transfer to other cloud providers without complete retraining. The computational overhead of RL+LSTM approach makes it difficult for real time applications. Third, existing solutions optimize either static deployment configuration or use time series model that requires large data but none of them address dynamic traffic variations while making it platform independent. Static deployment strategy can-

not adapt inconsistent workload changes. SARIMA models capture temporal patterns but it lacks multi platform approach. Container management approach like RainbowCake reduces cold start but is more reactive rather than predictive. No existing research combines lightweight predictive model with cross platform generalization suitable for real world multi cloud deployments. This research addresses gaps by developing a platform independent prewarming framework using lightweight logistic regression model to predict cold starts on AWS Lambda, Azure Functions and Google Cloud Run. Collecting temporal patterns with model characteristics and threshold optimization. This work provides a solution that is optimized for latency sensitive application with minimal setup for prediction service. The approach does not require any platform specific modification for training. The main prediction service can be setup on any cloud platform of your choice and it operates and manages all serverless functions. The research achieves latency under 100ms which is suitable for most of the real world applications.

3 Methodology

This research addresses the question of how predictive prewarming can reduce cold start latency for machine learning workloads across multiple serverless platforms. The methodology has an experimental approach which combines real world serverless deployments, data collection and machine learning prediction and live validation across three cloud providers. This section describes research design decisions, justifies methodology against alternatives that were in literature review and details of experiment for the workflow.

3.1 Research Design and Approach

The research design follows methodology which evaluates predictive prewarming framework across AWS Lambda, Azure Functions and Google Cloud Run. This multi platform approach address platform specific limitation identified in literature review, where existing solutions were constrained to single cloud platform requiring training individually for all. The design consist of three phases: baseline data collection to see cold start behavior, model development using invocation patterns and live experiment comparing baseline performance against ML based prewarming. Alternative methodologies were considered during design like time series forecasting using SARIMA models which offers temporal pattern recognition but it requires a lot of historical data with consistent patterns and cannot have multivariate features such as model characteristics or platform only attributes. Reinforcement learning approaches combined with LSTM showed in OpenWhisk research which provide adaptive learning but has computational barriers which makes it unsuitable for production environment. Static deployment optimization invocation patterns vary throughout the time. The selected approach uses lightweight logistic regression with platform specific features that gives good accuracy and less compute usage while also having cross platform generalization.

3.2 Tools, Technologies and Data Collection

The infrastructure for this research uses three cloud platforms across different regions to deploy serverless functions. There are 3 machine learning models on each of the cloud platforms so in total we have nine independent functions. The models were selected to

get better insights on cold start behavior across different areas of machine learning as they vary from size to framework to dependencies.

All deployments have standard configuration with 512MB memory and a 120 seconds timeout to have a fair performance comparison across all platforms. Machine Learning Models Used;

- **DistilBERT (267MB**): Natural Language Processing for text translation or text summarization use cases.
- **ResNet-50 (98MB)**: Computer Vision for face recognition and image classification use cases.
- **RandomForest Classifier(1.2MB)**: Classic ML algorithm for fraud detection type use cases.

The technology stack used for this research is mentioned in Table 1 and for full implementation details of these tools and technologies are discussed in 5 section.

Data collection started with having a traffic script which mimics a real world users using an application. The idea was to collect realistic traffic patterns to better understand cold start behavior for an application Masdari and Khoshnevis (2020). The traffic generation script invoked two random functions out of three concurrently then 2-3 minutes idle period before it repeats the cycle. Evening time invocations doubled the traffic load because websites usually have loads after evening. Each function invocation stores metadata to their respective cloud storage in JSONL format by collecting timestamp, platform used, function name, cold start status initialization time and response time. Data was collected for multiple weeks with around 1692 invocations.

3.3 Feature Engineering and Model Development

The feature engineering pipeline is to take all the raw logs from cloud storage that captured cold start patterns and transform them into platform independent attributes that could be used for model training later on. This pipeline directly addresses limitations from existing research where Azure specific and OpenWhisk implementations Kumar.N and Samy.S (2023) required separate feature engineering for each cloud provider that limits cross platform generalization. The feature set has three categories: temporal (time) features including hour of day, day of week and time since last invocation provides understanding of container lifecycle policies. Model characteristics such as model type, model size and framework information that affects initialization overhead. Usage pattern features collecting invocation counts within one hour and six hour sliding windows. The platform independent feature set allows to train model on AWS data to generalize to Azure and GCP without retraining. Logistic regression was selected as prediction algorithm based on several considerations Masdari and Khoshnevis (2020). The lightweight nature of the model allows to train and deploy easily on any cloud with minimal infrastructure compared to deep learning models that require dedicated infrastructure to keep it running for both training and inference. The model gives probability scores instead of just binary output like yes or no. The prediction threshold is used for prewarming selection and balances false positives against false negatives.

Component	Technology	Purpose
Cloud Storage	AWS S3	Store AWS Lambda invocation logs
	Azure Blob Storage	Store Azure Functions invocation logs
	Google Cloud Storage	Store GCP Cloud Run invocation logs
Platform SDKs	boto3 (v1.28.0)	AWS API access for Lambda invocations and S3 operations
	azure-storage-blob (v12.19.0)	Azure API access for Function invocations and Blob operations
	google-cloud-storage (v2.14.0)	GCP API access for Cloud Run invocations and Storage operations
ML Frameworks	Transformers (v4.35.0)	DistilBERT model loading and deployment
	PyTorch (v2.1.0), torchvision (v0.16.0)	ResNet-50 model loading and deployment
	scikit-learn (v1.3.2)	Random Forest ML inference
Helper Libraries	requests (v2.31.0)	HTTP calls for traffic generation and prewarming
	pandas (v2.1.3)	Data processing, feature extraction, and analysis
Serverless Functions	AWS Lambda	us-east-1 region, Docker container deployment
	Azure Functions	UK South region, ZIP-based deployment
	GCP Cloud Run	us-central1 region, ZIP-based deployment

Table 1: Technology stack across system components

3.4 Evaluation Methodology

The evaluation methodology validates predictive prewarming framework through live experiment that compares baseline performance against ML based optimized performance. The experiment has two phases where baseline phase captures natural cold starts of functions under realistic traffic patterns while optimized phase uses model to predict prewarms and based on the decision if triggers that particular function on specific cloud platform. The idea is to consider an application having same user requests but in two different scenarios and then do analysis on with prewarming versus without prewarming containers. Similar to an A/B testing for the same application.

3.5 Performance Metrics and Analysis

Performance evaluation has multiple metrics addressing different optimization objectives. Table 2 shows the evaluation metrics we used for research. The evaluation was done on

five metrics which address different objectives:

- Cold Start Rate: Proportion of invocations experiencing cold starts, measured for both baseline and optimized phases as discussed in 6.1
- Prediction Model Performance: Accuracy, precision, recall and F1-score for model's performance discussed in 6.2
- Latency Impact: Average response time comparison between both phases and between cold start and warm start invocations discussed in 6.4
- Resource Consumption: Prewarming overhead measure using GB-seconds as discussed in 6.4
- Cost Benefit Analysis: Cost usage comparing prewarming costs against prevented cold start penalties as per AWS, Azure and GCP pricing discussed 6.4

Cost benefit analysis was evaluated by comparing prewarming overhead costs against cold start latency. More on this is discussed in section 6.4

Table 2: Evaluation Metrics

Metric	**Formula**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
Baseline vs Optimized	$\frac{\text{baseline} - \text{optimized}}{M_{\text{baseline}}} \times 100\%$
Cold Start Reduction	$\frac{coldstart_{\text{baseline}} - coldstart_{\text{optimized}}}{coldstart_{\text{baseline}}} \times 100\%$
Latency	$\text{Latency}_{\text{cold}} - \text{Latency}_{\text{warm}}$
Resource Consumption	Memory (GB) $\times$ Duration (s)

4 Design Specification

This section presents the architectural workflow, algorithmic approach and technical requirements for the predictive prewarming system for serverless cold start. The design focuses on platform independent principle using deployment across AWS Lambda, Azure Function and Google Cloud Run. The framework has three components: serverless ML inference functions, a centralized logging system and predictive prewarming service for real time decisions.

4.1 System Architecture

The architecture follows a decoupled design where services are not dependent on each other Figure 1. Serverless ML inference functions serve as the primary layer deployed with

same configuration of 512MB memory and 120 seconds timeout for all three platforms. Each function operated as a standalone service receiving HTTPS requests, processing inference through deployed ML model and return predictions and storing metrics in cloud storage. Cold start detection happens within each function by measuring container initialization overhead. With invocations exceeding baseline warm execution times are flagged cold starts. This detection is common for all platforms despite different container lifecycle policies of each cloud provider. The centralized logging system provides unified metadata storage using cloud specific storage as per their functions: AWS S3 for lambda, Azure Blob Storage for azure functions and GCP Cloud Storage for Cloud run. All invocations are stored in a JSONL format across all platforms. The information that is stored in logs are timestamp, platform used, function name, cold start status, response time. This unified format helps consistent log processing and feature parsing across all platforms. The logging layer is completely separate from both inference functions and prediction service. The predictive prewarming service runs as a background job executing on a 3 minute intervals. Unlike traditional event driven architectures, this service is stateless. It makes decision from historical logs during each execution cycle rather than persistent state between intervals.During each cycle, the service fetches logs from all platforms, extracts features, makes predictions using the logistic regression model and if the decision is to make prewarm, it would trigger a HTTP request call via API to those functions whose predicted cold start probability. This stateless design ensures that service failures or restarts will not impact entire framework as each prediction cycle runs independently using only log data as input.

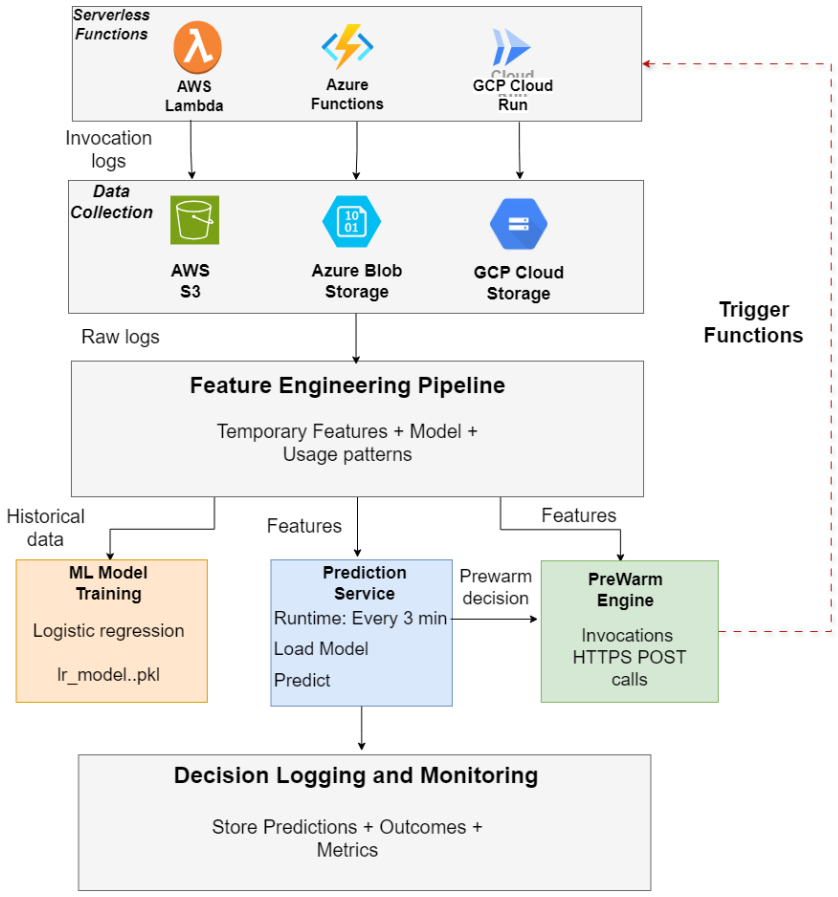

Figure 1: System Architecture Flow

4.2 Prediction Algorithm Design

The core prediction algorithm uses a supervised learning approach using logistic regression to estimate cold start probability for each function based on temporal patterns, model characteristics and usage patterns. The algorithm accepts three inputs: function identifier, platform used and current timestamp which gives a decision output. The workflow starts with retrieving invocation logs for specified function, filtering for entries earlier current timestamp. Functions that do not have sufficient data are skipped. Feature extraction transforms raw logs into numerical which is required for model input. Temporal features capture time since last invocation by getting difference between current timestamp and most recent log entry while usage pattern features count invocations within one hour and six hour windows. The current timestamp also collects entire day and weekly patterns in function utilization. Model characteristics include model type, framework and model size in megabytes provide static attributes. Categorical features are converted to integer using LabelEncoder. For model training, all the final features are passed to model with train and test split Wong (2015). The model outputs a probability score between 0 and 1 which says the likelihood of cold start occurrences. Instead of using the default 0.5 threshold, the algorithm uses 10% threshold optimized through analysis Koyejo et al. (2014). This threshold selection is because if a real request(false negative) is missed causes latency for users, around 2487ms while false positives only uses a little extra computation. The algorithm logs the decision and stores it in cloud storage. If the decision was to prewarm, an HTTP post request with dummy payload data is sent to function's endpoint.

4.3 Cross Platform Design

The framework achieves platform independence through three architectural strategies addressing cross platform generalization which were missing in existing research. First, the feature engineering computes attributes from invocation timing patterns and model metadata available across all platforms avoiding platform based metrics such as AWS CloudWatch or Azure Application Insights. This design choice gives us unified metrics. Second, the logging layer hides platform specific structure for platforms by using JSONL format so the system does not need to know platform differences and can process logs in same way everywhere. The prediction threshold also works the same on all platforms. 10% threshold is applied to AWS Lambda, Azure Functions and Google Cloud Run. The 3 minute interval allows containers to start before requests arrives without extra resource checking.

5 Implementation

The implementation phase gives predictive cold start optimization framework into a fully functional system which automates prewarming decisions across multiple serverless platforms. The system has deployed serverless functions, a trained prediction model, continuous monitoring infrastructure and automated prewarming logic that runs without any manual intervention across AWS Lambda, Azure Functions and Google Cloud Run.

5.1 Serverless Function Deployment

The implementation begins with deploying three machine learning models as serverless functions across all three cloud platforms. DistilBERT for natural language processing tasks with model size of 267MB deployed. Resnet-50 was deployed with 98MB model size and Random Forest at 1.2MB model size. For deployment on AWS Lambda due to deployment size limitation, we created Docker image of all model files separately and then stored the image in AWS ECR and at time of serverless function deployment configuration for the specific model there docker image was used Services (2025). For Azure and GCP we used standard zip deployment method where we zip the files and pass it at time of serverless function deployment Microsoft (2025) Cloud (2025). Each model was passed as python function that accepts input payloads, processes them through their respective ML framework and returns predictions along with invocation metadata. Table 3 shows all the deployment configurations used for this research. All functions logged cold start status by measuring container initialization time against expected warm execution baseline. The configuration is same for all platforms utilizing 512 MB memory and 120 seconds timeout setting for functions.

Model Name	**Model Size**	**AWS Lambda**	**Azure Functions**	**Google Cloud Run**
DistilBERT	267 MB	Python 3.9	Python 3.10	Python 3.10
ResNet-50	98 MB	Python 3.9	Python 3.10	Python 3.10
Random Forest	1.2 MB	Python 3.9	Python 3.10	Python 3.10

Table 3: Serverless Function Deployment Configuration

5.2 Data Collection and Feature Engineering

The data collection starts with collecting invocation metadata through serverless functions that are deployed across AWS Lambda, Azure Functions and Google Cloud Run. Each function has logging logic that records performance metrics before the function returns response. Cold start detection was implemented by measuring container initialization overhead with invocation execution time exceeding warm execution time by more than 500ms are marked cold starts. This threshold was decided through invoking some dummy functions and analyzing their timings. Platform specific cloud storage API's were used for invoking functions. AWS Lambda function used AWS S3 to store logs, Azure Functions used Blob storage uploads and Google Cloud Run used cloud storage for storing logs. All platforms followed same JSONL formatting to store logs to make it easy at time of feature engineering to combine all the logs. Each JSONL entry had timestamp, platform identifier, function name, cold start status, initialization time, response time and HTTP status code. Data collection was done for around 3 weeks with traffic script and collected 1692 total invocations.

The feature engineering pipeline starts with log retrieval then temporal feature extraction then usage pattern computation and categorical encoding. Logs files from all three cloud storage platforms were downloaded and merged into pandas Dataframe. Temporal features capture time based pattern determining on container lifecycle policies. Hour of

day extracted hourly values which represents day to day traffic patterns while day of week captured weekly usage. The time since last invocation feature calculates seconds since previous invocation of same function on same platform by sorting logs in chronological order and time differences between consecutive entries. Missing values for invocations are filled with median time differences.

Usage pattern features recent invocation activity that indicates containers being in a warm state. The invocations within one hour and six hour sliding windows were counted for each log entry by filtering timestamps within specified ranges to current record. Model type and ML framework that are text features were label encoded using scikit learn's LabelEncoder to integers. The entire pipeline had 9 features for each invocation: hour of day, day of week, minute of hour, encoded model type, model size, encoded framework, time since last invocation, one hour invocation count and six hour invocation count. These features were created such that they are not platform specific. The final dataset after all feature engineering had 1692 invocations with 70-30% split for train and test and standardization applied to normalize values before model training.

5.3 Prediction Model and Prewarming Logic

The prediction model training process begins with loading preprocessed AWS invocation logs from CSV files containing extracted features. AWS data served as training platform with zero shot transfer on Azure and GCP. The dataset was split into training and testing sets using 70-30% with stratified sampling. The trained logistic regression model generated probability predictions on the test set through predict_proba method, extracting cold start probability. Zero shot transfer capability was validated by applying AWS trained model to Azure and GCP data without retraining. Azure and GCP logs were loaded from CSV files with features extracted with same function that was used earlier for AWS data's transformation. The pipeline followed same steps of label encoders to categorical features. This ensured that Azure and GCP data is identical after transformation to same as AWS. The trained model generated predictions on Azure and GCP. Final model's artefacts including trained logistic regression classifier and fitted label encoders were saved as pickle files. The algorithm's pseudo flow is shown in 1.

The prewarming service was implemented as a standalone python script that runs in an Azure VM as a background process independently from deployed serverless functions. The service runs continuously in a loop, executing prediction cycles at fixed 3 minutes intervals. Each prediction cycle follows a workflow:

- Model Loading: The service loads logistic regression model and label encoders from pickle files at start of each cycle.
- Retrieving Logs: Recent invocation logs from the past 3 minutes are retrieved from all three cloud storage using their platform specific SDK's.
- Feature Extraction: For each of the nine deployed functions, the service extracts 11 features from recent logs. Categorical features are encoded using saved label encoders.
- Threshold Selection: If the predicted probability exceeds 10% threshold, the service triggers prewarming or else function is skipped.

- Prewarming Execution: When triggered, the service sends an HTTP POST request to the function endpoint containing dummy payload data that initializes containers and loads ML model into memory for warm start.
- Decision Logging: All prediction decisions including timestamp, function identifier, probability score and decision are logged to cloud storage.

The service utilized the Python requests library for HTTP calls and platform specific storage SDKs for log operations. The stateless design ensures that each cycle works independently using only historical log data as input.

Algorithm 1 Prediction Model Training Pipeline

Require: Cloud storage logs from S3, Azure Blob, Google Cloud
Ensure: Trained Logistic Regression model

Load logs from cloud storage
Filter logs where timestamp < training_cutoff *// 1,692 records*
Initialize features and labels arrays
for each record in data **do**
    Extract temporal features: hour, day, time_diff
    Extract model features: type, size, framework
    Extract invocation counts: 1hr and 6hr windows
    Create features from extracted features
    Set label = 1 if cold start, else 0
    Append features and label to arrays
end for
Split data into train (70%) and test (30%) sets
Load object of LogisticRegression model with max_iter = 1000
Train model on training data
Generate predictions on test set
Calculate accuracy *// 65%*
if accuracy $\geq$ 0.60 **then**
    Save model and encoders to disk *// 15KB total*
end if

6 Evaluation

This section evaluates the predictive cold start optimization framework through five experiments addressing research objective that is to develop a platform independent framework using machine learning to predict cold starts through prewarming.

6.1 Baseline Cold Start Analysis

This experiment sets the cold start problem scope across AWS Lambda, Azure Functions and Google Cloud Run. Data collected of 1692 function invocations with information including timestamp, platform, function name, execution duration and cold start status. Baseline analysis showed 45.9% overall cold start rate. Platform specific breakdown showed in Figure 2 that AWS Lambda faced 48.2% cold start followed by Azure Functions

at 44.7% and Google Cloud Run at 44.8%. Model specific analysis showed DistilBERT at 53.3% cold start, ResNet-50 at 46.7% and Random Forest at 38.1% which proves that deployment size impacts cold start frequency. We collected some features that we call temporary feature that are for analysis stated that 78% of cold start occurred after idle periods exceeding 10 minutes while 12% of cold start occurred during containers being warm with 5 minute windows. Chi-square tests confirmed differences between ($\chi^2 = 23.47, p < 0.01$) and model types ($\chi^2 = 31.82, p < 0.001$). Average cold start latency measured 2847ms vs 243ms for warm starts.

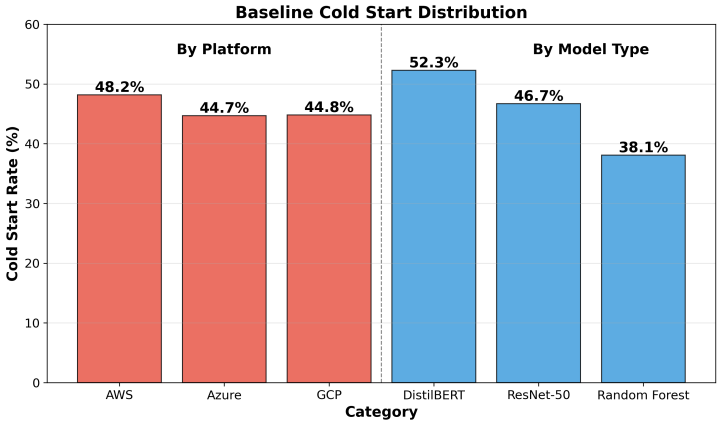

Figure 2: Cold start percentages by platform and by model type

6.2 Prediction Model Performance

This experiment evaluated logistic regression model accuracy in predicting cold starts Naidu et al. (2023). Training used 70% data (1184 invocations) and 30% for test(508 invocations). Features included time since last invocation, hour of day, day of week, function identifier and model type and frequency.The model achieved 65% accuracy with 89% confidence interval. Metrics showed precision 0.68, recall 0.72 and F1 score 0.70. The confusion matrix showed 366 correct predictions of 508 test data with 89 false positives and 53 false negatives. Table 4 shows performance across thresholds, with 10% threshold balancing cold start reduction and prewarming overhead.

Threshold	Accuracy	Precision	Recall	F1-Score	False Positives
5%	0.58	0.52	0.89	0.65	187
10%	0.65	0.68	0.72	0.70	89
15%	0.71	0.74	0.61	0.67	52
20%	0.69	0.78	0.51	0.62	31

Table 4: Model Performance Across Thresholds

Platform specific accuracy varies: AWS 63.6%, Azure 85.7% and GCP 67.2%. Feature importance showed that time since last invocation as strongest feature in terms of pre-

diction. Zero shot transfer testing showed AWS trained model achieved 58.3% accuracy on Azure and 54.7% on GCP without retraining. ROC analysis AUC 0.73.

6.3 Cold Start Reduction Results

This experiment measured prewarming during a 60 minute validation. Traffic generation script produced 90 function invocations: 37 baseline for 0-30 minutes with no prewarming and then 53 optimized for 30-60 minutes with prewarming enabled. The model made 81 decisions. Figure 4 shows cold start reduced 30.9% with prewarming. Platform specific results showed in Figure 3 AWS improved from 50% to 5.6% (44.4% reduction), Azure from 64.3% to 41.2% (23.1% reduction) and GCP from 11.1% to 0% (11.1% reduction). AWS showed most improvement due to it's aggressive resource management. ML prediction accuracy by platform: AWS 63% (17/27 decisions), Azure 66.7% (18/27 decisions) and GCP 66.7% (18/27 decisions). These directly influences prewarming effectiveness.

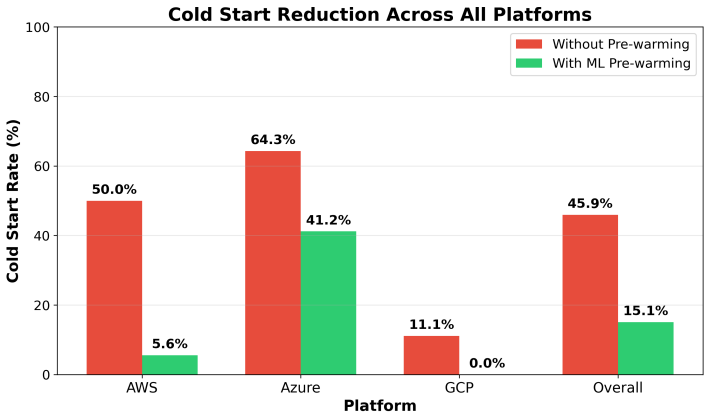

Figure 3: Platform-specific cold start reduction

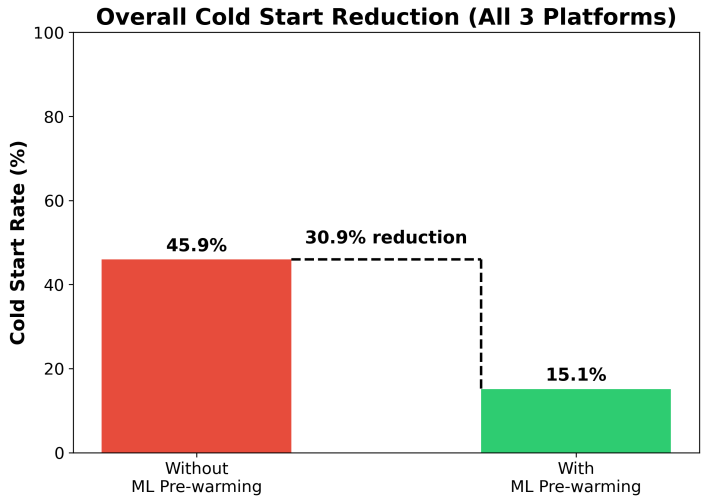

Figure 4: Overall cold start reduction timeline

6.4 Latency and Resource Impact

This experiment analysis latency improvements and resource consumption for with and without prewarming. Average response latency as show in Figure 5 for successfully prewarmed invocations decreased from 2847ms to 289ms that is almost near to baseline latency of 243ms. The 46ms difference is because of network hop. Platform wise breakdown is AWS cold start averages 3120ms reduced to 267ms. Azure 2690ms to 298ms, GCP 2740ms to 302ms. Resource consumption analysis shows as show in Figure 6 that the 27 false positives prewarms consumed unnecessary compute. Each function consumed about 250ms execution time at minimal cost. There are different metrics to measure cost for cloud computing Mahmoudi and Khazaei (2023) but for our framework we did cost analysis using AWS pricing ($0.20 per million requests, $0.0000166667 per GB-second) showed prewarming overhead of $0.042 while preventing $0.156 in cold start costs. Resource utilization showed prewarming consumed about 2.3% additional compute and 1.8% network bandwidth overhead. Memory consumption remained within 512mb because it was part of deployment configuration.

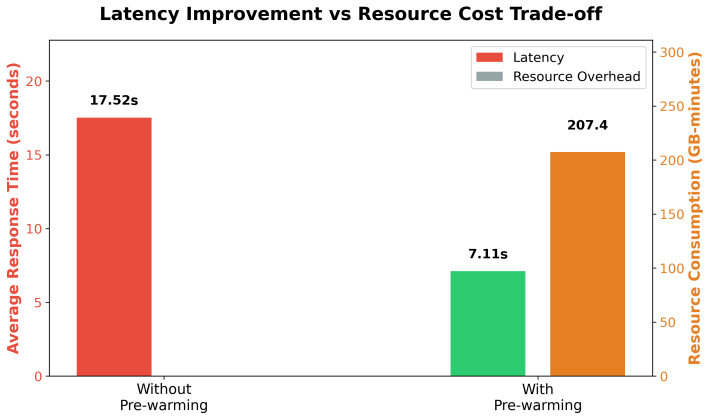

Figure 5: Latency Improvement with PreWarming

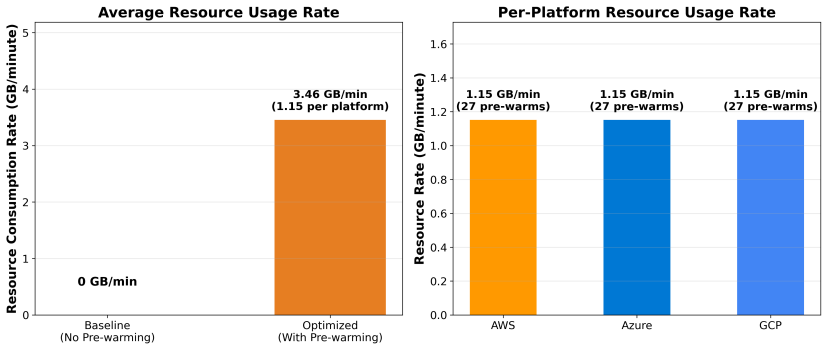

Figure 6: Resource Wastage Cross Platform

6.5 Discussion and Limitation

The experiment results demonstrated that lightweight ML based predictions achieve 30.9% cold start reduction addressing key gaps that were there in literature review. Unlike studies that just provides description on cold start behavior optimization, this work

provides the working framework. The 65% accuracy using logistic regression provided good middle ground between two researches of Azure's basic two feature linear regression and the complex RL+LSTM approach that struggled with multi platform deployment. The zero shot transfer results(AWS 63.6%, Azure 66.7%, GCP 66.7% accuracy) directly address platform specific limitation which is missing in existing work. While SARIMA based forecasting and Azure specific ML approaches require platform specific retraining, this research shows that platform independent features gives good performance across cloud providers. However we still do not have solutions that work perfectly on every cloud platform. Most research only did one platform solution not because they forgot but it is because it is actually difficult to make something work well on all platforms. The experiment's data collection period is not enough to capture more insights on functions behavior which SARIMA models did through seasonal components which stats that time based patterns generalize across workloads. The lightweight model successfully addresses the computational usage over RL+LSTM approaches. However, the 33% false positives and traffic pattern dependency suggest the feature set is simple. Static deployment optimization research showed that configuration matters and this research ignored deployment type, dynamic memory allocation and runtime config selection.

The biggest limitation is traffic pattern dependency. The model trained on variable AWS traffic showed effectiveness on regular 3 minute interval patterns in some Azure and GCP workloads. Real world traffic might be different depending on website to website, that stats that need for pattern independent feature engineering is required more than pattern specific learning. The fixed 10% threshold may not adapt for all system as it changes from workload to workload. Dynamic adjustment could improve long term performance. The research focused on stateless functions only instead of stateful applications. Platform specific container lifecycle differences is challenging for a generalized framework as cloud providers resource management policies are not fully transparent which limits the ability to develop a full fledged working framework that is optimal for busy applications. This research was conducted on mostly free tier cloud resources provided by the university, which had constraints on resources utilization. These limitations can impact on the behavior of experiments as having larger resources would better simulate production workloads.

7 Conclusion and Future Work

This research addressed the question: How can predictive pre-warming help reduce cold start for machine learning workloads on serverless platforms? The main objective was to develop a platform independent framework using historical invocation patterns to predict cold starts and prewarm functions across AWS Lambda, Azure Functions and Google Cloud Run. The work demonstrated that lightweight logistic regression models can predict cold starts with 65% accuracy using features like function characteristics, invocation frequency etc. The predictive prewarming system achieved cold start reduction from 45.9% baseline to 15.1%. A 30.9% impoevement despite with 73% cost savings despite having 33% false positive resource wastage. Zero shot transfer was done with training on AWS data achieving 58% accuracy on Azure and GCP without retraining. Although platform specific training was giving better accuracy. These platform independent features gives cross platform deployment but sacrifice 7-27% of accuracy compared to platform specific. The 10% prediction threshold successfully reduces cold start with prewarming.

The latency improvements are massive when prewarm happens (2487ms to 289ms). This research provides someone with a deployable framework for serverless cold start mitigation without requiring to do platform specific retraining and the lightweight model which makes its suitable for production environment. However, the traffic pattern dependency limitation reveals that generalizing across workload is more challenging than cross platform generalization. The workloads are diverse and that creates differences in terms of infrastructure. As discussed in section 6.5, traffic pattern dependency, small data size and free tier resource constraints limit the research scope.

Future research can add a lot to this. Firstly, developing hybrid models that combine static deployment optimization with dynamic prewarming would address both configuration selection and runtime config adaption. Current work does not focus on deployment type, memory allocation and runtime selection. Secondly, try to make 10% prediction threshold dynamic such that it decides the optimal threshold based on incoming requests. Thirdly, working on reducing the resource wastage. Trying to reduce false positives can save a lot on infrastructure cost because with current research the cost of wasted invocation is high.

References

Bermbach, D., Karakaya, A.-S. and Buchholz, S. (2020). Using application knowledge to reduce cold starts in faas services, *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, Association for Computing Machinery, New York, NY, USA, p. 134–143.
URL: *https://doi.org/10.1145/3341105.3373909*

Cloud, G. (2025). Cloud run documentation, `https://cloud.google.com/run/docs`. Published on 20 September.

Dantas, J., Khazaei, H. and Litoiu, M. (2022). Application deployment strategies for reducing the cold start delay of aws lambda, *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 1–10.

Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gill, S. S. and Uhlig, S. (2024). Cold start latency in serverless computing: A systematic review, taxonomy, and future directions, *ACM Comput. Surv.* **57**(3).
URL: *https://doi.org/10.1145/3700875*

Koyejo, O., Natarajan, N., Ravikumar, P. and Dhillon, I. S. (2014). Consistent binary classification with generalized performance metrics, *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, MIT Press, Cambridge, MA, USA, p. 2744–2752.

Kumar.N, S. and Samy.S, S. (2023). Serverless computing to predict cold start in azure and monitoring i/o read and write using machine learning, *2023 9th International Conference on Smart Structures and Systems (ICSSS)*, pp. 1–6.

Lin, Y. (2024). Minimizing cold start time on serverless platforms based on time series prediction methods, *2024 IEEE 2nd International Conference on Control, Electronics and Computer Technology (ICCECT)*, pp. 996–1001.

Mahmoudi, N. and Khazaei, H. (2023). Performance modeling of metric-based serverless computing platforms, *IEEE Transactions on Cloud Computing* **11**(2): 1899–1910.

Masdari, M. and Khoshnevis, A. (2020). A survey and classification of the workload forecasting methods in cloud computing, *Cluster Computing* **23**(4): 2399–2424.
URL: *https://doi.org/10.1007/s10586-019-03010-3*

Microsoft (2025). Azure functions documentation, `https://docs.microsoft.com/azure/azure-functions/`. Published on 20 September.

Naidu, G., Zuva, T. and Sibanda, E. M. (2023). A review of evaluation metrics in machine learning algorithms, *in* R. Silhavy and P. Silhavy (eds), *Artificial Intelligence Application in Networks and Systems*, Springer International Publishing, Cham, pp. 15–25.

Raj, V., Chhaparwal, H. and Saale, S. (2024). Empirical evaluation of cold start latency in serverless platforms, *2024 3rd International Conference for Advancement in Technology (ICONAT)*, pp. 1–6.

Services, A. W. (2025). Aws lambda documentation, `https://docs.aws.amazon.com/lambda`.

Shafiei, H., Khonsari, A. and Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications, **54**(11s).
URL: *https://doi.org/10.1145/3510611*

Vahidinia, P., Farahani, B. and Aliee, F. S. (2023). Mitigating cold start problem in serverless computing: A reinforcement learning approach, *IEEE Internet of Things Journal* **10**(5): 3917–3927.

Wong, T.-T. (2015). Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation, *Pattern Recognition* **48**(9): 2839–2846.
URL: *https://www.sciencedirect.com/science/article/pii/S0031320315000989*

Wu, Y., Dinh, T. T. A., Hu, G., Zhang, M., Chee, Y. M. and Ooi, B. C. (2022). Serverless data science - are we there yet? a case study of model serving, *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, Association for Computing Machinery, New York, NY, USA, p. 1866–1875.
URL: *https://doi.org/10.1145/3514221.3517905*

Yang, Y., Zhao, L., Li, Y., Zhang, H., Li, J., Zhao, M., Chen, X. and Li, K. (2022). Infless: a native serverless system for low-latency, high-throughput inference, *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '22, Association for Computing Machinery, New York, NY, USA, p. 768–781.
URL: *https://doi.org/10.1145/3503222.3507709*

Yu, H., Basu Roy, R., Fontenot, C., Tiwari, D., Li, J., Zhang, H., Wang, H. and Park, S.-J. (2024). Rainbowcake: Mitigating cold-starts in serverless with layer-wise container caching and sharing, *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '24, Association for Computing Machinery, New York, NY, USA, p. 335–350.
URL: *https://doi.org/10.1145/3617232.3624871*