ISSN: 2093-5374 / E-ISSN: 2093-5382

A Novel Heuristic & Cache-Based Adaptive Prewarming Algorithm for Efficient Cold Start Mitigation in Serverless Computing

Sivaranjani Sundaram[1*], Anandhi Ramani[2], K. Ramya[3], and S. Malathy[4]

[1*]Assistant Professor, PG Department of Information Technology and BCA, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India. sivaranjani.dgvc@gmail.com, https://orcid.org/0009-0005-3874-8879

[2]Assistant Professor, PG & Research Department of Computer Science, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India. sekadhi@gmail.com, https://orcid.org/0000-0002-7247-7567

[3]Assistant Professor, PG Department of Information Technology and BCA, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India. ramya.dgvc2024@gmail.com, https://orcid.org/0009-0006-9546-743X

[4]Research Scholar, PG & Research Department of Computer Science, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India. malathy10112001@gmail.com, https://orcid.org/0009-0009-4452-9348

Received: July 06, 2026; Revised: August 18, 2026; Accepted: September 04, 2026; Published: September 30, 2026

Abstract

Serverless computing offers scalability and cost-efficient execution; however, cold-start latency from runtime setup, dependency loading, and resource allocation remains a critical issue for delay-sensitive applications. This research aims to build an adaptive, lightweight prewarming framework to minimize cold-start overhead while maintaining efficient resource utilization. To develop a Heuristic and Cache-based Adaptive Prewarming (HCAP) framework by combining workload-based function categorization, cold-start probability prediction, adaptive threshold-based prewarming, and LRU cache management. The framework constantly tracks function invocation behavior using a sliding window approach and categorizes functions based on their frequencies and inter-arrival properties into Hot, Warm, and Cold states. Trace-driven simulation is carried out to validate the performance of HCAP compared to the execution and static prewarming schemes based on cold start rate, Latency, resource utilization, and warm pool budget usage metrics. From the experiments, it is evident that HCAP lowers the cold start ratio to 9.7% from 42.8% when using the baseline execution approach and 18.5% when using the static prewarming method. In addition, HCAP achieves an average latency of 198 ms and a P95 latency of 360 ms. In terms of warm-pool budget usage, HCAP reduces it to 55.9% from 65.2% with the static prewarming approach. The component ablation analysis further confirms each component's contribution by eliminating heuristic classification, LRU caching, and adaptive threshold control to achieve latencies of 245 ms, 230 ms, and 220 ms, respectively. In summary, the suggested HCAP model offers an ideal trade-off between minimizing Latency and optimizing resources while ensuring adaptation and decision-making in a serverless environment.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), volume: 17, number: 3 (September-2026), pp. 1935-1954. DOI: *10.58346/JOWUA.2026.I3.112*

*Corresponding author: Assistant Professor, PG Department of Information Technology and BCA, Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India.

Keywords: Serverless Computing, Cold Start Latency, Adaptive Prewarming, Heuristic Algorithms, LRU Caching, Performance Optimization.

1 Introduction

Background

The serverless computing paradigm has gained considerable popularity in cloud computing due to the capability of executing applications without having to manage the underlying infrastructure, using event-driven execution and a pay-per-use model of resource allocation. Serverless computing enables dynamic allocation of computing resources based on workload needs. For these reasons, the serverless computing approach has been adopted for applications with elastic resource provisioning and fast service deployment. Nevertheless, although it has certain operational benefits, serverless computing has yet to overcome the challenge of performance issues, and one of the key problems in this area is the cold-start phenomenon (Baldini et al., 2017). A cold start occurs when a function is invoked after an inactive period and must be initialized before it can process the task. Cold starts add latency and adversely affect the performance of latency-sensitive applications. Research has investigated the workload characteristics of serverless systems and found that the frequency of invocations, execution time, and other variables have an impact on cold starts (Lloyd et al., 2018; Shahrad et al., 2020).

Various strategies have been tried to address cold start Latency in the past, including reusing containers, pre-warming, predictive scheduling, and adaptive resource provisioning. In early strategies, attempts were made to keep a warm execution environment to minimize the delay in initialization; however, the ongoing process of pre-warming results in increased consumption of resources since the instances will have to be maintained despite not being needed at all times. Resource-efficient container management strategies have explored various provisioning and scheduling approaches to improve serverless execution performance and minimize unnecessary resource overhead (Oakes et al., 2018; Suresh et al., 2020). Recent research work in this field has proposed dynamic pre-warming approaches that analyze workload patterns and make functions available based on invocation behavior. However, such methods enhance responsiveness based on varying workloads but are confronted with issues of prediction accuracy and the trade-off between latency reduction and costs incurred (Song et al., 2024; Kumari & Sahoo, 2024).

Machine learning and AI techniques have also been used in cold start prevention through the prediction of function invocation sequences and resource allocation optimizations. Despite being able to predict the workload accurately, they suffer from the drawback of requiring large amounts of training data, constant updates to the model, and extra computing capacity. Recent studies have highlighted the need for cost-aware, adaptive methods that can reduce cold starts efficiently without adding computational complexity to the system (Golec et al., 2024; Huang et al., 2026). Techniques such as cost-aware scheduling and intelligent optimizations have tried to solve this problem by taking into account both the performance improvement and resource costs of the pre-warming process (Nguyen & Chien, 2025; Atwal, 2026).

Existing cold-start mitigation approaches still have drawbacks in dynamic serverless environments where workload behavior varies constantly. While static prewarming approaches can lead to over-provisioning of resources, prediction-based approaches can have problems with dependency on workloads and their generalizability. Modern adaptive approaches provide better cold-start mitigation through workload-awareness and intelligent resource provisioning. Still, more work is needed to come

up with even lighter approaches that could make prewarming decisions in real-time without overloading resources (Aetukuri et al., 2025; Buitrago et al., 2025). As a result, a good cold-start mitigation approach should detect priority functions and estimate cold-start risk for prewarming.

The main goal of this research is to come up with the HCAP approach, which stands for Heuristic and Cache-Based Adaptive Prewarming technique, which helps to mitigate the cold-start latency problem in serverless computing systems. The main goal of the framework under consideration is to create an adaptive technique that includes heuristics-based analysis of workload as well as caching-based function management. In particular, the proposed solution will analyze invocation patterns, inter-arrival behavior, and function state to define their type as Hot, Warm, and Cold. Then, based on the estimated cold-start probability and resource constraints, HCAP will decide whether a function needs prewarming. Moreover, LRU caching will be used for better utilization of warm instances.

The main contributions of this research work are outlined as follows. Firstly, a lightweight Heuristic and Cache-Based Adaptive Prewarming (HCAP) framework is presented for serverless cold-start handling by combining workload-aware classification and cache-based instance management techniques. Second, a heuristic decision technique predicts cold-start likelihood based on invocation frequency, recency, and workload characteristics, enabling selective prewarming instead of constant instance management. Third, an LRU-based cache management approach is employed to improve function reuse efficiency and minimize reinitialization costs. Fourthly, a resource-aware prewarming approach is suggested by taking into account the cost of latency improvement and resource consumption while making decisions. Lastly, the proposed HCAP framework is validated via a trace-based simulation study and compared with other prewarming approaches.

The rest of the paper is structured as follows. Section 2 covers the review of prior work on cold start solutions and their shortcomings. Section 3 introduces the HCAP method in detail, including the system architecture, model formulation, and algorithm design. Section 4 covers the methodology of experiments and implementation. Section 5 includes results and discussions. Section 6 is the conclusion of the paper.

2 Existing Methods and their Limitations

Latency reduction for cold starts has been researched thoroughly in the field of serverless computing. Current approaches mainly focus on keeping the execution environment alive, predicting future use, and efficiently allocating resources. Some of the popular techniques that have been used in order to reduce the initialization time include idle-based pre-warming, machine learning-based prediction, static caching, adaptive provisioning, and cost-aware scheduling. Nevertheless, there is room to improve the balance among latency reduction, resource efficiency, and adaptability (Golec et al., 2024; Ghorbian & Ghobaei-Arani, 2024).

2.1 Limitations and Drawbacks of Existing Approaches

Idle-Based Prewarming

Idle-based prewarming methods minimize cold start delays by keeping the instances active and ready periodically. The approach works well in the case of commonly used functions since the runtime environment is already in place even before the request arrives. But keeping the instances alive in an idle phase causes memory waste along with increased costs. The success rate of the idle-based method

is low when workloads have uncertain invocation patterns since there is no way of estimating the demand (Oakes et al., 2018; Shahrad et al., 2020).

Machine Learning-Based Prewarming

Machine learning approaches have been proposed to forecast future function invocations in the cloud and make prewarming decisions accordingly. These approaches learn from historical workloads and enhance prediction accuracy under stable circumstances. Nevertheless, ML approaches require huge historical data sets, regular training, and increased computation power; their reliance on workload properties limits the scope of applicability to highly dynamic serverless platforms with variable traffic loads (Huang et al., 2026; Aetukuri et al., 2025).

Static Caching-Based Approaches

Static caching techniques address cold start issues by keeping a certain number of containers or function instances that are always ready and hence warm. However, static caching methods cannot effectively manage the dynamic nature of workloads because they allocate fixed resources regardless of changes in demand. This technique causes inefficiencies during peak and off-peak periods due to over-allocation and under-allocation of resources, respectively (Shahidi et al., 2021; Huang et al., 2026).

Adaptive Provisioning and Scheduling Methods

The adaptive resource provisioning technique is designed in such a way that the resources are provisioned according to the behavior and needs of execution. It enhances the scalability process by taking into consideration the invocation behavior and availability of resources. However, there are certain adaptive techniques, which add to complexity because of monitoring, optimization, and predicting workload (Kumari & Sahoo, 2024; Suresh et al., 2020; Biswas & Kumar, 2024)

Cost Inefficiency and Resource Overhead

One of the primary drawbacks of current cold-start mitigation strategies is that they cannot effectively minimize Latency and costs simultaneously. Prewarming results in lower Latency but higher infrastructure usage, while not prewarming results in reduced cost but more frequent cold starts. Cost-aware scheduling algorithms have been introduced recently to solve this problem, although it is still difficult (Nguyen & Chien, 2025; Atwal, 2026).

Limited Adaptability to Real-Time Traffic Variations

Applications in a serverless system suffer from unpredictable workloads. Current strategies can be incapable of reacting swiftly to such unpredictable workloads, in which case the workloads need adaptive and efficient prewarming techniques. While some prewarming methods have taken into consideration monitoring of workload and adaptivity, not all of them consider workload variation, probability of cold start, and infrastructure cost together (Song et al., 2024; Pandey et al., 2025).

2.2 Problem Statement

Cold-start mitigation solutions developed to date have been partially successful in terms of improvement in function availability and resource optimization. But there are limitations associated with their ability to operate in a lightweight and efficient manner. Solutions involving idle-based and static caching have

the advantage of reducing Latency by keeping the instance active continuously, but cause unnecessary use of resources. Machine learning-based solutions, though, offer the ability to predict cold start latency, but require a lot of training data and resource-intensive computations. Adaptive provisioning solutions offer flexibility in operation, but increase complexity in the process (Golec et al., 2024; Buitrago et al., 2025; Ghorbian & Ghobaei-Arani, 2024).

In this context, the necessity of a lightweight adaptive mechanism arises that is able to dynamically detect high-priority functions, evaluate cold-start likelihood, and perform selective prewarming without increasing resource overhead. The needed solution should be able to take into consideration the factors of workload, cache usage, and resource constraints in order to maintain a good trade-off between Latency and cost. In order to overcome these limitations, propose a novel Heuristic and Cache-Based Adaptive Prewarming (HCAP) framework, which involves workload-aware heuristic classification combined with LRU cache management for intelligent prewarming decisions. The suggested strategy seeks to minimize cold start delays, increase resource efficiency, and ensure its applicability in real-world settings (Besozzi et al., 202; Waly et al., 2025; Zhang et al., 2025).

3 Proposed Heuristic and Cache-Based Adaptive Prewarming (HCAP) Framework

The proposed Heuristic and Cache-Based Adaptive Prewarming (HCAP) model attempts to tackle the cold-start problem in serverless computing systems through the detection of critical invocations and the implementation of prewarming on them. Different from the traditional approach that keeps the function instances running all the time, the proposed HCAP conducts the prewarming process on the basis of the estimated cold-start probability exceeding an adaptive threshold. The proposed framework consists of four main modules: (i) sliding window-based invocation monitoring, (ii) heuristic workload classification, (iii) adaptive prewarming control, and (iv) LRU-based cache.

The invocation monitoring component continuously monitors the execution of functions and detects their invocation characteristics, which include the invocation frequency and inter-arrival time. With such information provided, the heuristic classification component is able to classify the functions into three states – Hot, Warm, and Cold states. In addition, the cold-start probability estimation component calculates the possibility of cold-start occurrence, while the adaptive threshold control component decides whether prewarming needs to be done. Lastly, the LRU cache component manages the warm instances that are more useful.

3.1 System Architecture of HCAP Framework

The overall architecture of the proposed HCAP framework is shown in figure 1. The HCAP framework takes up incoming requests from the serverless function requests and works through an analysis and decision process in stages. To start off, function invocation data are first extracted using the monitoring module via the sliding window technique. The extracted invocation data will then be passed to the heuristic classification module for the purpose of classifying the functions according to their behavior in terms of frequency and recency.

Function state classification is then analyzed using the cold-start probability estimation module to determine the likelihood of initialization delay. Depending on the probability of such delay and current utilization of system resources, the adaptive threshold mechanism determines if pre-warming should be applied. In cases where pre-warming is necessary, the concerned function is initialized and then placed

in the warm-instance cache. The LRU cache manager will continually update the function instances in the cache to keep only frequently requested functions and remove least recently used functions.

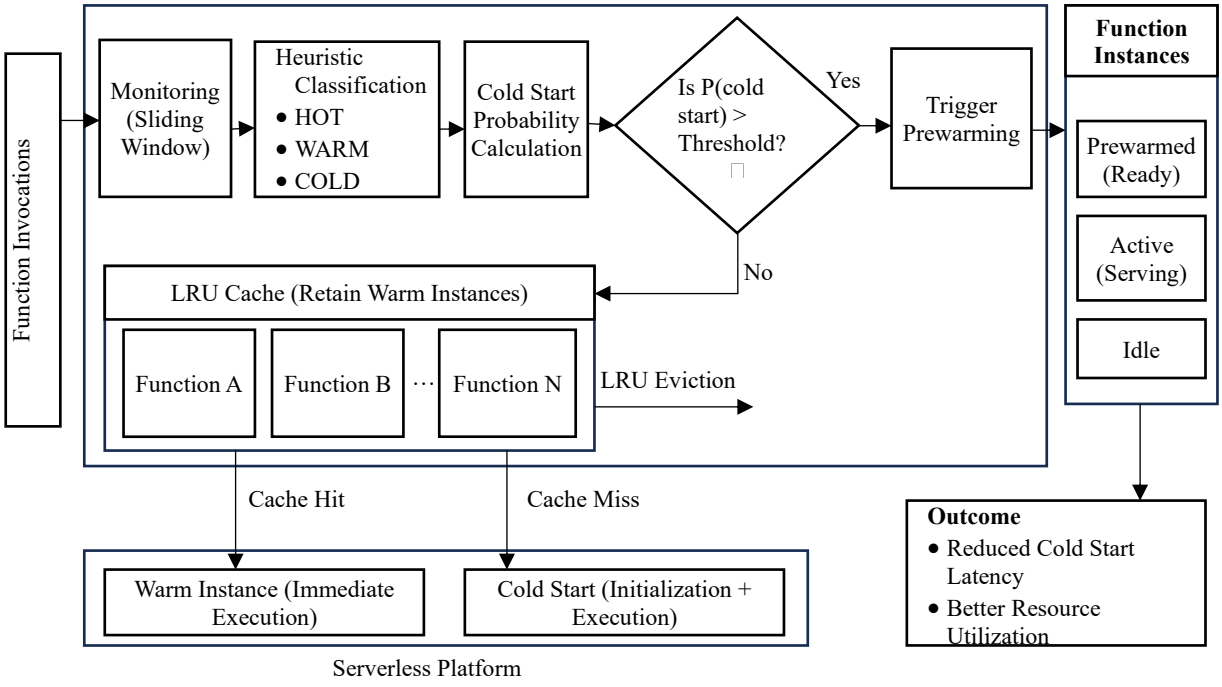


Figure 1: Architecture of the proposed heuristic and cache-based adaptive prewarming (HCAP) framework

Figure 1 demonstrates the architectural design of the suggested HCAP framework that addresses the problem of cold-start Latency in the realm of serverless computing systems. The framework starts with the monitoring process of input functions with a sliding window technique in order to gather their workload characteristics. The obtained function invocation frequency and recency are analyzed by the heuristic classification module to classify the functions into Hot, Warm, and Cold.

Then, the cold-start probability estimation module calculates the probability of the cold-start delay, and the adaptive decision module makes the decision whether to start pre-warming of the function instances or not based on the calculated risk threshold. In case the conditions for pre-warming are met, the corresponding function instance is pre-warmed and remains as a warm instance. Otherwise, the request goes through the regular execution process. The LRU cache mechanism manages warm instances and discards the least recently used instances in order to optimize resource consumption.

3.2 System Model and Assumptions

The HCAP design assumes the existence of a serverless computing model which has a number of function instances. The set of the available functions is represented as follows:

$$F = \{f_1, f_2, .., .. f_n\} \quad (1)$$

Equation (1) where $f_i$represents the i^{th}serverless function. Each function receives a sequence of invocation requests over time, which can be represented as:

$$I_i = \{t_1, t_2, .., .. t_m\} \quad (2)$$

Equation (2) where $t_m$denotes the timestamp of the m^{th}invocation request for function f_i. In operation, each of the function instances retains certain operational states according to their availability and initialization. A function instance can be in the warm, idle, and cold states. The term "warm" stands for an active or pre-warmed function that can be immediately utilized for executing requests, while the idle state of a function instance denotes the inactive state that allows its reuse without full initialization. The term "cold" stands for a function instance which is not initialized yet, so it requires runtime initialization in order to be executed.

3.3 Invocation Monitoring Using Sliding Window

In order to measure the varying load on functions, HCAP uses a monitoring scheme based on sliding windows. The monitoring window facilitates an analysis of the invocation pattern of functions continuously. Let Δt be the time period of the monitoring window. The invocation frequency of function f_i during the monitoring window is given by:

$$\lambda_i(t) = \frac{N_i(t)}{\Delta t} \quad (3)$$

Equation (3) where $N_i(t)$ denotes the number of requests handled by functionf_i in the window period. The Inter-arrival time between two consecutive requests is given by

$$I_k = t_k - t_{k-1} \quad (4)$$

Equation (4), where t_k and t_{k-1} denote the current and previous invocation timestamps, respectively. These workload parameters contain information about temporal locality and behavior of functions, which are later utilized for prediction of cold starts.

3.4 Cold Start Probability Estimation

The cold-start probability estimation component estimates the probability of cold-start delay based on the characteristics of the workloads observed. The probability model introduced takes into account invocation frequency and recency as follows:

$$P_{cold} = e^{-\beta\lambda_i(t)}(1 - R_i) \quad (5)$$

Equation (5), where βrepresents the sensitivity parameter. The probability of cold-start decreases with an increase in the frequency of invocation since popular functions are more likely to stay active.

3.5 Adaptive Threshold-Based Prewarming Decision

The execution process of HCAP depends on cache availability. When the requested function is available in the cache, the warm instance is directly executed, reducing initialization overhead. Otherwise, the request follows the cold execution path, where the required environment is initialized before execution.

The expected Latency is calculated as:

$$P_{cold} \geq \tau \quad (6)$$

Equation (6), where τ denotes the decision threshold value. In order to make the decision based on the conditions of the system, the threshold is dynamically set based on the current resource utilization:

$$\tau(t) = \tau_0 + \gamma \frac{U(t)}{U_{max}} \quad (7)$$

Equation (7), where τ_0 denotes the initial threshold value, $U(t)$ is the current utilization, U_{max} denotes the maximum resource utilization capacity, and γdenotes the scaling factor.

Once the pre-warming criterion is met, HCAP then determines the relationship between latency gain and resource cost. Prewarming cost is determined by the following equation:

3.6 Cost-Aware Prewarming Strategy

After satisfying the prewarming condition, HCAP evaluates the relationship between latency improvement and resource consumption. The prewarming cost is calculated as:

$$C_{prewarm} = C_{compute} + C_{memory} \quad (8)$$

Equation (8), *where* $C_{compute}$and C_{memory}represent computational and memory requirements. The latency improvement achieved by converting a cold instance into a warm instance is represented as:

$$B_i = L_{cold,i} - L_{warm,i} \quad (9)$$

Equation (9), where $L_{cold,i}$and $L_{warm,i}$ are the latencies for the cold and warm executions, respectively.

3.7 LRU-Based Cache Management

To enhance the reuse of warm function instances, HCAP uses the Least Recently Used (LRU) algorithm for managing the cache. The cache maintains the frequently accessed function instances and records their usage details after each execution process. If the cache becomes full, the least recently used function instance is evicted from the cache. Cache performance is measured based on the cache hit ratio:

$$CHR = \frac{N_{hit}}{N_{hit}+N_{miss}} \quad (10)$$

Equation (10), where $N_{hit}$and N_{miss}represent cache hit and cache miss occurrences.

3.8 Execution Model and Optimization Objective

Execution of the HCAP will be dependent on availability in the cache. In cases where the function to be requested is available in the cache, then execution of the warm instance takes place. On the other hand, the request goes through cold execution, which initializes the environment first before execution.

Expected Latency is given by:

$$E(L) = P_h L_{warm} + P_m L_{cold} \quad (11)$$

Equation (11), where $P_h$and $P_m$represent cache hit and cache miss probabilities. The resource utilization is expressed as: Equation (12)

$$RU = \frac{R_{used}}{R_{total}} \quad (12)$$

The joint optimization goal of the HCAP framework is to reduce Latency and cost concurrently:

$$\text{Minimize}(\alpha L + \beta C) \quad (13)$$

Equation (13) where $\alpha$and βrepresent the weighting parameters for latency and cost optimization.

3.9 HCAP Algorithm

The entire operation procedure of the HCAP framework is presented in Algorithm 1. In this algorithm, the first step involves initialization of the cache and the detection of the incoming function calls. For each function call, extraction of invocation features and calculation of heuristic score based on frequency and recency values are performed. Further, a function status check is carried out, and then the cold-start probability is computed, followed by the computation of the adaptive threshold value. On satisfying the pre-warm condition, the framework calculates the latency gain and resource cost before executing the function instance.

The complete workflow of the proposed HCAP framework is presented in Algorithm 1.

Algorithm 1: Heuristic and Cache-Based Adaptive Prewarming (HCAP)

Input:

Function invocation stream (Fi)

Cache capacity (C)

Sliding window size Δt

System utilization (U(t))

Output:

Optimized prewarming decisions and reduced cold-start latency

Procedure:

1. Initialize LRU cache.
2. Monitor incoming function requests.
3. Calculate invocation frequency and inter-arrival time.
4. Extract frequency and recency scores.
5. Compute heuristic score.
6. Classify functions into Hot, Warm, and Cold categories.
7. Estimate cold-start probability.
8. Calculate adaptive threshold.
9. If (P_{cold}):
   - Calculate latency benefit.
   - Calculate prewarming cost.
   - Trigger prewarming if benefit exceeds cost.
10. Update LRU cache.
11. Remove least recently used instances when cache capacity exceeds limit.

The proposed HCAP framework provides adaptive cold-start mitigation by combining workload-aware prediction, selective prewarming, and efficient cache management.

4 System Implementation

The HCAP framework implementation is done via an adaptive and budget-friendly pre-warming approach for the real-time implementation of the HCAP system in the serverless platform. This approach is useful in overcoming the challenges of cold start latency, which is usually experienced in computing platforms. The entire simulation framework operates based on the trace-driven simulation framework, since it facilitates the replaying of real-time function invocations as sequences. The function invocations are periodically replayed to calculate the effectiveness of the pre-warming approach.

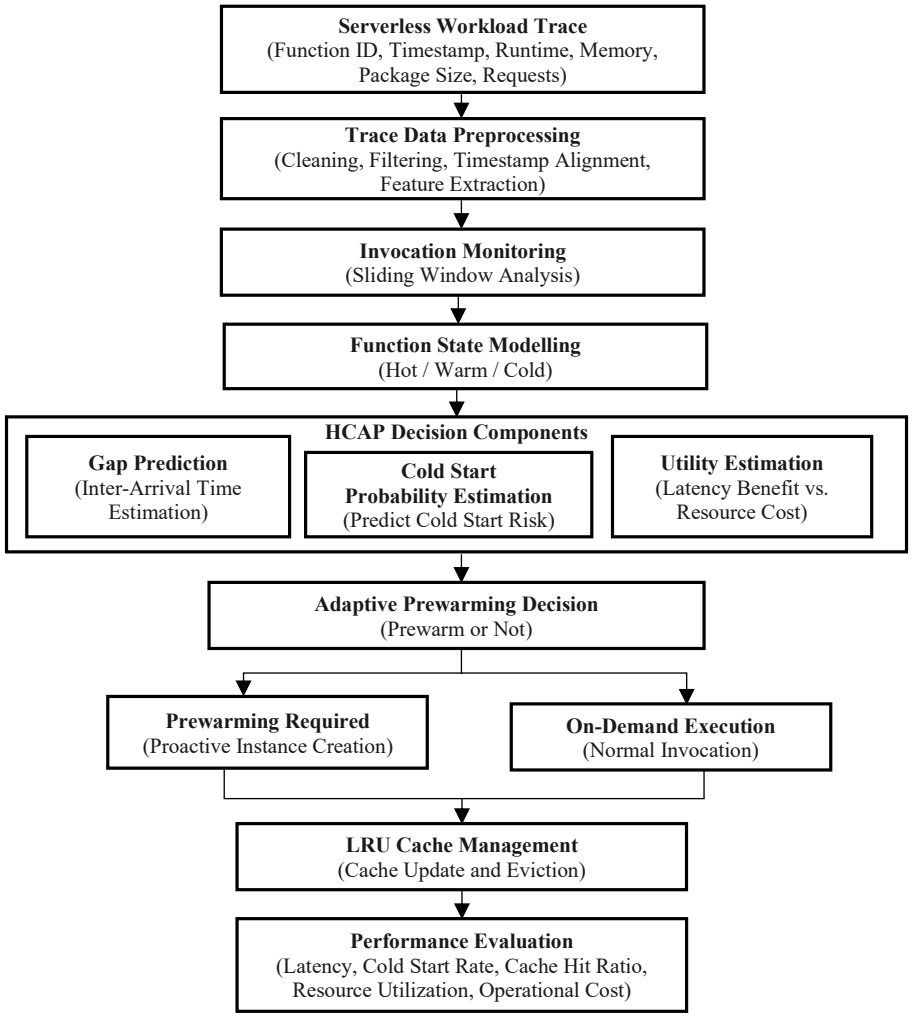


Figure 2: Trace-driven HCAP implementation framework for serverless cold start mitigation

The implementation flow of the HCAP framework using the trace-based simulation environment is shown in figure 2. The invocation monitoring and the functional state model are used to pre-process and analyze the workload traces and classify them into Hot, Warm, and Cold States. The HCAP decision modules are responsible for calculating the gap prediction, cold-start probability, and the utility score to make adaptive prewarming decisions. An LRU cache manager is included in the framework to improve the reuse of functions.

4.1 Trace-Driven Simulation

HCAP framework is tested through a trace-driven simulation technique, where historical serverless workload traces are replayed to study cold starts under dynamically invoked scenarios. Trace-driven simulation helps in simulating the serverless execution pattern realistically by retaining several workload characteristics such as sporadic arrivals, bursting behavior, and dependency among function invocations.

The function-level information includes function ID, invocation time, execution time, memory size requirement, and deployment details of functions. First of all, the raw data collected through traces are preprocessed to resolve issues related to inconsistencies and normalized into structured invocation sequences along with invocation timestamps. Subsequently, the preprocessed information about the workload is analyzed based on the sliding-window technique to derive frequency and inter-arrival information needed for HCAP decision modules.

Trace-driven simulation also facilitates monitoring function status, cache status, and resource usage continuously throughout the simulation process. Trace-driven simulation serves as a controlled environment for comparing the proposed HCAP framework with other cold start prevention mechanisms. However, trace-driven simulation does not help in testing the low-level behavior of clouds such as network delay and container initialization cost.

4.2 Function State Modeling

The HCAP model is used to estimate the execution status and invocation pattern for each of the serverless functions. Function execution state is tracked through three main parameters, namely: invocation frequency, execution status, and inter-arrival time. The use of the above parameters makes it easy for the framework to detect any activity in terms of workloads and the chances of encountering cold starts in the future.

Inter-arrival time between consecutive invocations is computed as follows:

$$I_i = t_i - t_{i-1} \tag{14}$$

Equation (14). Where the terms t_i and t_{i-1} signify the Timestamps recorded at the current and previous invocation. If the value of the inter-arrival gap is smaller, then it indicates the probability for the cold start risk to be lower and invocation continuity to be higher.

Invocation frequency is computed in terms of the monitoring window to detect the most frequently invoked functions. Functions with high invocation frequency have higher chances of being active. Execution status is one of the parameters that determine whether function instances are active or not.

4.3 Inter-Arrival Gap Prediction

Gap prediction plays a significant role in HCAP, since the arrival pattern of the workload directly impacts the cold start phenomenon. The framework involves the use of a lightweight prediction model based on Exponentially Weighted Moving Average (EWMA), median, and invocation behavior. In contrast to sophisticated prediction models based on learning mechanisms, the suggested solution predicts future invocation gaps with lower overhead.

Prediction of the invocation gap is performed as follows: Equation (15)

$$\hat{G}_f(t) = 0.5 \cdot EWMA + 0.3 \cdot Median + 0.2 \cdot Latest \tag{15}$$

Where:

- $EWMA$: This term indicates the Value of the Exponentially Weighted Moving Average between recent Invocation functions.
- $Median$: This Term is the median evaluation between recent arrivals of invocation functions.
- $Latest$: It shows the most recently arrived invocation gap in the system performance

Since the prediction model consists of three different statistical components such as EWMA, median, and latest measure, different wave components are considered in the prediction model to increase the system stability and to achieve adaptive prediction performance. The ewme statistical characteristic captures the workload trends in the system, which enables giving more weight to the newly observed patterns. While, choosing the median statistic improves the robustness in production capability since this characteristic minimizes the spikes and the irregular invocation pattern of the immediate workload behavior. Finally, the latest statistic captures the immediate workload instances.

The suggested prediction algorithm is easy and lightweight, and it supports the HCAP protocol to predict the cold start problem in its yearly period. This will help trigger the adaptive pre-warming technique without any delays. Choosing this prediction algorithm enables improving the system stability and reducing the latency effects. Without having to use deep learning methods and advanced prediction models, the HCAP protocol is able to make predictions with reduced computational overhead.

4.4 Cold-Start Probability Estimation

Upon determining the invocation gap, the HCAP model calculates the probability of cold-starts based on workload attributes. The cold-start probability model takes into account the predicted invocation gaps, rate of invocations, burstiness, and past invocation history to determine the probability of cold-starts.

The cold-start probability is determined by:

$$P_f^{cold}(t) = \sigma\left(w_1 \hat{G}_f(t) + w_2 F_f^{-1} + w_3 B_f + w_4 H_f\right) \quad (16)$$

Equation (16), where $\sigma(w_1)$ represents the sigmoid function, $\hat{G}_f(t)$ denotes the predicted inter-arrival gap, F_f represents invocation frequency, B_f indicates the burstiness factor, and H_f represents the historical behavior score. The sigmoid function converts the estimated risk into a probability value between zero and one, enabling adaptive prewarming decisions.

4.5 Utility-Based Adaptive Prewarming Decision

The HCAP architecture applies a utility-based decision-making process that assesses if the benefit derived from pre-warming outweighs the costs involved. In contrast to pre-warming all functions, the architecture will pre-warm only those functions where there is greater benefit derived.

Utility of function f is given by:

$$U_f = w_r R_f + w_b B_f + w_c C_f + w_a A_f - w_{cost} Cost_f \quad (17)$$

Equation (17), where R_f represents response time improvement, B_f represents the benefit score, C_f denotes confidence level, A_f indicates arrival rate, and $Cost_f$ represents the resource cost.

The final prewarming decision is determined using:

$$Prewarm_f = \begin{cases} 1, & P_f^{cold} > \theta_p \text{ and } U_f > \theta_u \\ 0, & otherwise \end{cases} \quad (18)$$

Equation (18), where θ_p and θ_u represent the cold-start probability and utility thresholds. The dual-condition mechanism ensures that prewarming is activated only when both latency risk and expected benefit are significant.

4.6 Resource and Budget Constraint Model

However, the prewarming procedure needs extra computational and storage resources to keep the functions warm. Hence, the resource model is used in the HCAP design to compute the cost of executing the prewarming activity and minimize any wastage of resources. The cost of each function will be computed as:

$$Cost_f = Memory_f + 20 \times PackageSize_f \quad (19)$$

Equation (19), where $Memory_f$ denotes the memory that is used to allocate space for the function instance, while $PackageSize_f$ denotes the deployment package size of functions. Those functions that have large memory allocation requirements and big package sizes create more prewarming costs; hence, HCAP takes both factors into consideration.

To control resource usage, a budget constraint is applied to limit the total prewarming cost:

$$\sum_{f \in F_p} C\ ost_f \leq B_{eff} \quad (20)$$

Equation (20), where B_{eff} denotes the available budget for prewarming. In the event that the calculated cost is greater than the available budget, functions with low priority are dropped from the consideration set using their utility scores. The budget-based approach ensures that HCAP reduces Latency without compromising efficiency.

5 Results and Discussion

5.1 Experimental Setup and Evaluation Metrics

In order to evaluate the proposed HCAP approach, a serverless trace-driven simulation environment was used to assess the effectiveness of HCAP in minimizing cold-start Latency. The simulator processes function invocation traces and generates the realistic behavior of serverless workloads based on invocation frequency, function execution time, resource requirements, and state changes. In the evaluation of HCAP, the effectiveness of HCAP in identifying cold-start risks and executing selective prewarming is the focus.

In order to evaluate the performance of the proposed approach, its performance was compared against the performance of baseline execution and static prewarming approaches. The following performance metrics were used in the evaluation: cold-start rate, average Latency, P95 latency, resource utilization, cache efficiency, and warm-pool budget utilization.

5.2 Performance Comparison with Existing Prewarming Strategies

Performance comparison of HCAP with Baseline and Static pre-warming is presented in table 1:

Table 1: Performance comparison of HCAP with existing strategies

Strategy	Cold-start rate (%)	Average latency (ms)	P95 latency (ms)	Resource usage
Baseline	42.8	412	890	120
Static Prewarming	18.5	265	540	410
HCAP	9.7	198	360	260

The outcome of this comparison reveals that HCAP has the minimum cold start ratio of 9.7%. This means that the prediction and pre-warming are more efficient than other approaches. The average Latency is also minimized to 198 ms with moderate resource consumption. While static pre-warming improves latency by maintaining warm instances throughout, this increases the cost of resource overhead since unnecessary instances remain alive. However, in HCAP, selective pre-warming of instances is performed according to workload behavior.

5.3 Function Category-Based Cold-Start Behavior Analysis

HCAP categorizes functions based on workload behavior into three categories, namely Hot, Warm, and Cold functions. The figure below gives a categorization of function behavior.

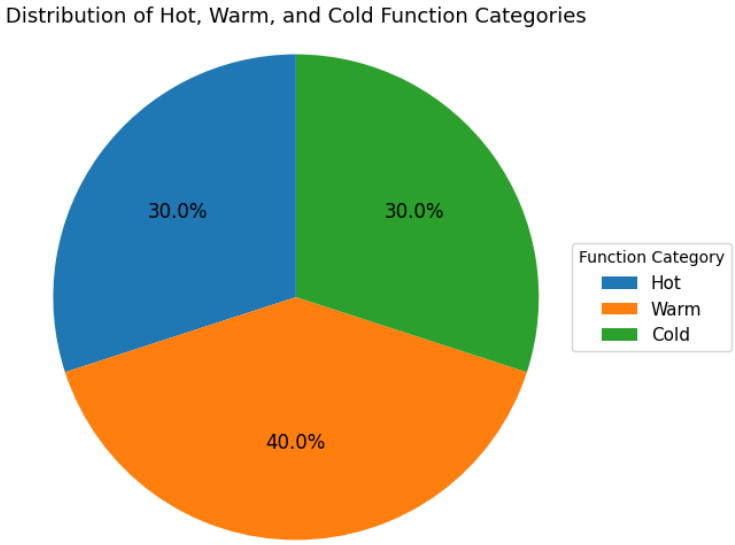


Figure 3: Distribution of hot, warm, and cold function categories

Figure 3 shows the frequency of different categories of functions obtained through the HCAP heuristic classification algorithm. The classification allows workload-based prewarming through identification of often-used functions and prevents resource allocation to unused functions.

5.4 Cold-Start Probability Estimation Analysis

The cold-start probability distribution of different function categories generated via the HCAP heuristic classification method is shown in figure 4. The x-axis denotes the different function categories, while the Y-axis denotes the cold-start probability of those function categories (%). The hot functions have a low range of cold-start probability, which is about 45-63%, with a median of about 54%, because of repeated function invocation. The cold-start probability of warm functions is moderate, which falls into

a range of 55-78% with a median of 66% due to their repeated behavior. The cold functions have a higher range of cold-start probability, which is 59-89%, with a median of 78%.

The increasing likelihood pattern from Hot to Cold functions proves the efficiency of HCAP in categorizing HCAP workload functions. Function behavior patterns can be identified by HCAP, and only risky functions are prewarmed, thereby saving unnecessary resource utilization and avoiding cold-start delay.

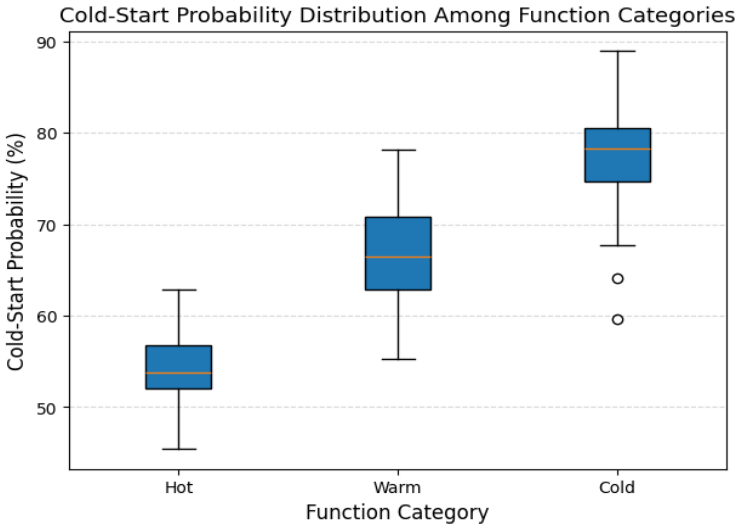


Figure 4: Cold-start probability distribution among function categories

5.4 Adaptive Cold-Start Probability Estimation Analysis

The cold-start probability estimation ability of HCAP can be determined through an analysis of various function classes. In table 2, the average estimate of cold-start probability is shown.

Table 2: Cold-start probability estimation

Function category	Number of functions	Average cold-start probability (%)
Hot	30	65.9
Warm	40	67.1
Cold	30	66.5

The findings of probability estimation show that HCAP constantly assesses the nature of workload instead of depending only on static features of the function. The probability values enable adaptation in decision-making.

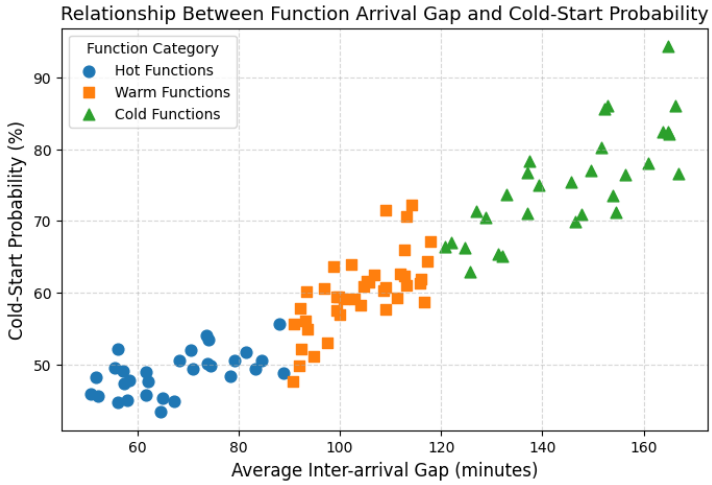


Figure 5: Relationship between function arrival gap and cold-start probability

Figure 5 depicts the relation between the average inter-arrival gap and the estimated cold-start probability of various function categories. The horizontal axis refers to the average invocation gap ranging from 50 to 170 minutes, whereas the vertical axis reflects the cold-start probability from 40% to 85%. The hot functions with small inter-arrival gaps (50-90 minutes) have a low cold-start probability of about 45-60%, as they have higher invocations and utilize their warm instances. The warm functions with medium inter-arrival gaps (90-120 minutes) display a medium cold-start probability of approximately 60-75%. On the other hand, the cold functions with large inactivity times (120-170 minutes) possess high cold-start probabilities of about 75-85%.

This is due to the higher likelihood of instance expiration in these functions. The discovered trend proves that long invocation gaps increase the probability of occurrence of cold starts. This proves the workload-aware prediction technique of HCAP, wherein inter-arrival behavior plays an important role.

5.5 Latency Reduction and Resource Efficiency Analysis

Latency reduction efficiency of HCAP is considered along with the use of warm pools to study the trade-off between performance enhancement and resource consumption. Static prewarming keeps the function instances always active, whereas HCAP uses prewarming depending on workloads and resource limitations.

Table 3: End-to-end latency and warm-pool budget utilization

Method	Average End-to-End Latency (ms)	Warm-Pool Budget Utilization (%)
Baseline	588.9	0.0
Static Prewarming	496.2	65.2
Adaptive Prewarming (HCAP)	517.1	55.9

Table 3 shows that static pre-warming attains the smallest Latency due to its ability to keep more warm instances, but it needs a higher budget allocation for warm-pooling at 65.2%; on the other hand, the new HCAP model is able to achieve a balance between Latency and budget allocation by having a latency of 517.1 ms using only 55.9% of the warm-pool budget.

5.6 Resource Cost and Initialization Overhead Analysis

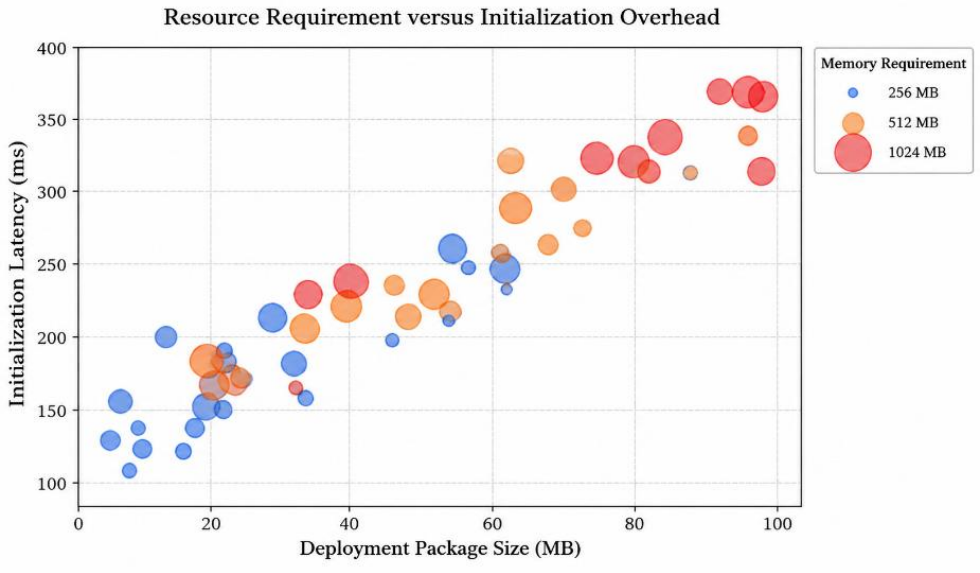


Figure 6: Resource requirement versus initialization overhead

Figure 6 depicts the relation between deployment package size, memory requirement, and function initialization latency. Deployment package size is presented on the x-axis ranging from 5 MB to 100 MB, and initialization latency is on the y-axis from 100 ms to 370 ms. Bubble size reflects memory requirement, which varies from 256 MB to 1024 MB.

The findings suggest that functions with small package sizes (around 5–30 MB) require less initialization time (100–200 ms) due to the absence of high overhead for dependency loading. Meanwhile, functions with large deployment package sizes (70–100 MB) require much more initialization time (300–370 ms) due to the necessity of resource preparation. Large bubbles in the area of large package sizes reflect greater memory usage. Therefore, it is evident that there is a clear connection between resource usage and initialization overhead. Hence, the chosen resource-aware prewarming approach used by HCAP can be explained based on those observations.

5.7 Ablation Analysis of HCAP Components

An ablation study involves the removal of key HCAP elements one at a time in order to assess the effect of each element on HCAP.

Table 4: Ablation analysis of HCAP framework

Model variant	Cold-start rate (%)	Average latency (ms)
Without Heuristic Classification	16.8	245
Without LRU Cache	14.5	230
Without Adaptive Threshold	12.9	220
Complete HCAP	9.7	198

Ablation results from table 4 demonstrate the role played by each of the HCAP components. Excluding heuristic classification leads to poor workload prediction ability, whereas excluding cache management leads to high initialization overhead. The best performance is achieved by the HCAP architecture because of the synergy between these components.

5.8 Discussion Summary

The HCACB-based HCAP framework effectively minimizes cold start latency in serverless architectures through workload-aware heuristic classification, adaptive prewarming, and LRU-based caching. According to experimental outcomes, HCAP was able to minimize the cold start rate to 9.7%, whereas the corresponding cold start rates in baseline execution and static prewarming were 42.8% and 18.5%, respectively. HCAP also achieved average and P95 latency of 198 ms and 360 ms, respectively. Efficient resource utilization was also obtained by the framework through minimal use of the warm pool budget, at 55.9% in comparison to 65.2% in static prewarming. In addition, the ablation study showed the importance of individual components, where omission of heuristic classification increased Latency to 245 ms, omission of LRU caching increased Latency to 230 ms, and omission of adaptive threshold control increased Latency to 220 ms.

The above observations suggest that a selective pre-warming approach through invocation behavior and cache awareness represents a more suitable trade-off between performance and efficiency compared to instance maintenance throughout the entire time frame. The above observations stress the importance of light-weight and adaptive approaches in dynamic serverless applications were workload behavior changes frequently.

However, it is important to note that the above observations are made by means of trace-driven simulations, which do not cover all cloud deployment issues, including network latencies and heterogeneous infrastructure. Further work is needed in order to test HCAP in realistic cloud settings and evaluate its scalability.

6 Conclusion and Future Work

Serverless computing helps deploy applications effectively and economically; nevertheless, cold start overhead is one of the biggest issues because of the high overhead of initializing inactive functions. Although methods such as static prewarming and continuously maintaining warm instances help to reduce the response time, they consume too many resources, while other methods like prediction-based approaches require higher complexity. In order to solve the issue above, the authors have developed the Heuristic and Cache-Based Adaptive Prewarming (HCAP) scheme, which utilizes workload-aware function classification, threshold-based adaptive prewarming, and LRU cache management. Experimental results show that HCAP has greatly enhanced the performance of the serverless platform, reducing the cold start rate from 42.8% (baseline execution) and 18.5% (static prewarming) to 9.7%. The authors' method has reduced average Latency to 198 ms, whereas the amount of consumed resources is well-balanced at 55.9%, lower than 65.2% for static prewarming. Furthermore, the ablation study proves that eliminating heuristic function classification, LRU caching, and threshold-based adaptive prewarming leads to an increase in latency up to 245 ms, 230 ms, and 220 ms, respectively. The main contribution of this study is that adaptive and selective pre-warming can be used to mitigate the cold-start problem efficiently without continuous resource reservation. HCAP provides a low-cost, practical approach to serverless environments with an optimized trade-off between latency reduction, caching effectiveness, and cost-efficiency. The potential future research directions include validation of the framework using real cloud deployment, multi-cloud workload analysis, and adaptive learning approaches.

References

[1] Aetukuri, S. A., Agarwal, R., Goyal, K. K., Nutalapati, P., Sadat, Q. T., & Lakhina, U. (2025, August). Optimal Holistic Cold-Start Management in Serverless Cloud Computing Using Artificial Intelligent. In *2025 International Conference on Intelligent and Secure Engineering Solutions (CISES)* (pp. 1095-1099). IEEE.

[2] Atwal, S. (2026, April). Optimizing Serverless Computing Performance: A Cost-Aware Strategy for Cold-Start Reduction and Workload Scheduling. In *2026 4th International Conference on Knowledge Engineering and Communication Systems (ICKECS)* (pp. 1-5). IEEE.

[3] Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., ... & Suter, P. (2017). Serverless computing: Current trends and open problems. In *Research advances in cloud computing* (pp. 1-20). Singapore: Springer Singapore.

[4] Besozzi, V., Della Bartola, M., Dazzi, P., & Danelutto, M. (2025). High-performance serverless computing: A systematic literature review on serverless for HPC, AI, and big data. *IEEE Access*, *13*, 195611-195656.

[5] Biswas, T., & Kumar, P. (2024, June). Optimizing resource management in serverless computing: A dynamic adaptive scaling approach. In *2024 15th International Conference on Computing Communication and Networking Technologies (ICCCNT)* (pp. 1-7). IEEE.

[6] Buitrago, N., Camacho, H., Jimeno, M., Viloria-Nuñez, C., Cardona, J. A., & Salazar, A. (2025). PARSEC: An Adaptive and Efficient Platform for Reducing Cold Start in Serverless Computing. *IEEE Transactions on Services Computing*, *18*(6), 4082-4095.

[7] Ghorbian, M., & Ghobaei-Arani, M. (2024). A survey on the cold start latency approaches in serverless computing: an optimization-based perspective. *Computing*, *106*(11), 3755-3809.

[8] Golec, M., Walia, G. K., Kumar, M., Cuadrado, F., Gill, S. S., & Uhlig, S. (2024). Cold start latency in serverless computing: A systematic review, taxonomy, and future directions. *ACM Computing Surveys*, *57*(3), 1-36.

[9] Huang, C., Huang, Y., Feng, J., Liang, S., Yan, M., & Wu, J. (2026). LACE: Mitigating cold-starts in serverless with a multi-task mixture-of-experts caching. *Journal of King Saud University Computer and Information Sciences*, *38*(3), 93.

[10] Kumari, A., & Sahoo, B. (2024). ACPM: adaptive container provisioning model to mitigate serverless cold-start. *Cluster Computing*, *27*(2), 1333-1360.

[11] Lloyd, W., Ramesh, S., Chinthalapati, S., Ly, L., & Pallickara, S. (2018, April). Serverless computing: An investigation of factors influencing microservice performance. In *2018 IEEE international conference on cloud engineering (IC2E)* (pp. 159-169). IEEE.

[12] Nguyen, H. D., & Chien, A. A. (2025, December). Efficient performance guarantees for function-as-a-service with cloud allocators. In *Proceedings of the 26th International Middleware Conference* (pp. 99-113).

[13] Oakes, E., Yang, L., Zhou, D., Houck, K., Harter, T., Arpaci-Dusseau, A., & Arpaci-Dusseau, R. (2018). {SOCK}: Rapid task provisioning with {Serverless-Optimized} containers. In *2018 USENIX annual technical conference (USENIX ATC 18)* (pp. 57-70).

[14] Pandey, S. R., Martinez, M. M., & Herbke, P. (2025, March). Coldstart reduction in serverless using function discovery and distribution. In *2025 28th Conference on Innovation in Clouds, Internet and Networks (ICIN)* (pp. 125-129). IEEE.

[15] Shahidi, N., Gunasekaran, J. R., & Kandemir, M. T. (2021, November). Cross-platform performance evaluation of stateful serverless workflows. In *2021 IEEE International Symposium on Workload Characterization (IISWC)* (pp. 63-73). IEEE.

[16] Shahrad, M., Fonseca, R., Goiri, I., Chaudhry, G., Batum, P., Cooke, J., ... & Bianchini, R. (2020). Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *2020 USENIX annual technical conference (USENIX ATC 20)* (pp. 205-218).

[17] Song, Y., Ye, B., Zeng, Y., & Wang, X. (2024, December). Dynamic Pre-Warm Strategies for Reducing Cold-Start Latency in Serverless Edge Computing. In *2024 20th International Conference on Mobility, Sensing and Networking (MSN)* (pp. 228-235). IEEE.
[18] Suresh, A., Somashekar, G., Varadarajan, A., Kakarla, V. R., Upadhyay, H., & Gandhi, A. (2020, August). Ensure: Efficient scheduling and autonomous resource management in serverless environments. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)* (pp. 1-10). IEEE.
[19] Waly, N., El-Ramly, M., & El-Kady, M. (2025, November). SPARQ: Sidecar Pooling for Serverless Cold Start Mitigation. In *2025 Twelfth International Conference on Intelligent Computing and Information Systems (ICICIS)* (pp. 1-9). IEEE.
[20] Zhang, H., Mu, S., Angel, S., & Liu, V. (2025). Causalmesh: A causal cache for stateful serverless computing. *ACM SIGMOD Record*, *54*(1), 48-57.

Authors Biography



Sivaranjani Sundaram serves as an Assistant Professor at Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India, bringing over 16 years of teaching experience. Her academic expertise spans Data Mining, Programming Languages, Database Management Systems (DBMS), and Operating Systems. She has presented 10 research papers at national and international conferences. She has consistently been involved in shaping the curriculum and supporting departmental undertakings. Her teaching style highlights the integration of practical experiences with academic theories. She demonstrates a commitment to lifelong learning through regular participation in various faculty development activities and scholarly events.



Anandhi Ramani is an Assistant Professor at Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, with over 19 years of teaching experience. Her areas of expertise include Database Management Systems (DBMS), Operating Systems, Programming Languages, Blockchain Technology, Cloud Computing, and Android Development. She has published more than 20 research papers in reputed journals and international conferences. Currently, she is guiding three research scholars under the University of Madras. She has been actively involved in curriculum development and departmental activities. Her teaching approach integrates practical applications with theoretical concepts. She regularly participates in faculty development programs and academic workshops to stay current in her field.



K. Ramya is a faculty member in the field of computer science with a strong focus on software engineering and system modeling. Her research interests include Unified Modeling Language, software design methodologies, and the integration of emerging technologies in education and organizational systems. She actively engages in scholarly research, academic writing, and conference-oriented studies to enhance both theoretical and practical knowledge. She participates in faculty development and induction programs to strengthen her research and pedagogical competencies. Her academic goal is to contribute high-quality research that supports innovation, effective teaching, and knowledge dissemination.



S. Malathy is a full-time Ph.D. research Scholar in the PG and Research Department of Computer Science at Dwaraka Doss Goverdhan Doss Vaishnav College, Chennai, Tamil Nadu, India. She holds a master's degree in Computer Applications and has qualified the UGC-NET examination in Computer Science and Applications. Her research interests include cybersecurity, stock market manipulation detection, explainable artificial intelligence, intrusion detection systems, and financial data analytics.