XAI for Optimizing Serverless Computing: A Machine Learning Strategy to Address Cold Start Challenges

N. Yaswanth[1] and R. Rohan Chandra[2]

Department of Computer Science and Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore, India
Phone: +91-9676794131, +91-8639884378
{namburiyaswanth.221cs232@nitk.edu.in, regulagaddarohanchandra.221cs241@nitk.edu.in}

Abstract. Serverless computing allows for great scalability and cost savings by abstracting infrastructure management. However, cold start latency—delays experienced when functions are activated after a period of inactivity—remains a significant challenge. This study describes a machine learning (ML) and Explainable AI (XAI)-driven architecture for predicting and mitigating cold starts via dynamic resource optimization. Our solution uses advanced AI models to pre-warm resources based on expected consumption patterns, effectively lowering cold start delays and improving system responsiveness. The use of XAI approaches increases transparency in decision-making and allows stakeholders to understand resource allocation procedures. Experimental results show a significant reduction in cold start latency, which leads to better overall system performance and user experience. This adaptive and intelligent technique provides a scalable solution for optimizing resource management in serverless systems while increasing confidence through explainability.

1 Introduction

The Digital Revolution and the Rise of Information age in the mid-twentieth century radically altered traditional businesses, paving the way for creative models such as e-commerce. This shift toward a digitally interconnected society demanded the use of network-based technologies, enhanced computing resources, and scalable infrastructure. Initially dominated by client/server architectures, the landscape altered as enterprises faced escalating installation costs and an urgent need for scalability, resulting in cloud computing. [1]This paradigm shift brought shared resources, platforms, and services, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [**?**], which transformed how businesses manage infrastructure, applications, and data.

In 2014, Amazon Web Services (AWS) introduced serverless computing, a ground-breaking architecture that isolates infrastructure administration, allowing developers to focus entirely on code composition and deployment. Serverless computing solutions such as AWS Lambda, Google Cloud Functions, and Microsoft Azure automatically supply and scale resources, simplifying development and increasing agility. This design [2] enables developers to deploy cloud-native functions without the need for server management, resulting in cost savings and faster development cycles. As a result, serverless computing has become a key component of current cloud services, particularly for applications with variable workloads.

Despite its various advantages, serverless computing confronts a number of performance problems, the most crucial of which is cold start delay. Cold starts happen when serverless functions are

called after a period of inactivity, requiring time to initialize the necessary resources before execution can begin. Due to the statelessness of serverless functions, resources are relinquished after each execution, and fresh containers must be initialized for future requests. This delay [3] can severely damage performance in latency-sensitive applications, such as real-time healthcare monitoring in the Internet of Things (IoT) or online data analytics, where instantaneous processing is required.

To mitigate cold start latency, serverless platforms frequently use ways to keep containers warm for a set amount of time after execution, resulting in faster response times for subsequent requests. This strategy, however, has the potential to waste resources and increase operational expenses because containers may remain active even while idle. Furthermore, serverless applications frequently include a wide range of function kinds, such as I/O-intensive, CPU-intensive, or network-intensive tasks, each with its own set of resource requirements [4]. This variability makes static container warming techniques ineffective for dynamic workloads.

In light of these problems, this study provides a novel framework that uses Machine Learning (ML) and Explainable AI (XAI) techniques to dynamically optimize resource allocation while minimizing cold start delay. By monitoring function invocation patterns, our system intelligently pre-warms containers depending on expected usage, ensuring effective resource utilization and minimal delays [5]. We use deep neural networks and Long Short-Term Memory (LSTM) models to learn historical invocation patterns and calculate the ideal idle container time. This adaptive technique ensures that containers are kept warm only when necessary, increasing cost efficiency while preserving performance. Furthermore, the use of XAI enhances openness by allowing developers and stakeholders to understand and trust the system's resource management decisions.

This study makes three key contributions:

- An adaptive machine learning approach that uses deep neural networks and LSTMs to predict invocation trends and dynamically reduce cold start latency.
- A method for learning function invocation patterns and determining the ideal warm container duration based on real-time demand, hence reducing resource waste.
- A complete comparison of the proposed architecture to existing strategies from leading serverless platforms such as AWS Lambda, Microsoft Azure, and OpenFaaS, as evaluated with real-world serverless apps.

The rest of the paper is arranged as follows.: Section II covers serverless computing. Section III discusses the cold start problem. Section IV discusses related works that address cold start difficulties in serverless setups. Section V describes the recommended method for addressing the cold start problem. Section VI compares the performance of the proposed model to baseline techniques. Finally, Section VII wraps up the work and discusses future research options.

2 Serverless Computing

Defining serverless computing [4] can be problematic due to its dynamic nature and many service models. A better insight can be gained by studying the level of control developers have over resources in various cloud service architectures.

Infrastructure as a Service (IaaS): This paradigm gives developers complete control over the application and the underlying cloud infrastructure. IaaS enables developers to tailor virtual machines, storage, and networks to individual application requirements.

Platform as a Service (PaaS): In contrast, the PaaS approach abstracts the underlying infrastructure, giving developers a managed platform for building and deploying apps without

having to worry about the infrastructure itself [2]. This enables developers to focus on coding and application logic. while the platform manages resource allocation, scaling, and maintenance.

Software as a Service (SaaS): SaaS is a more abstract concept in which programs are totally hosted and controlled in the cloud. Users can access these programs via the internet. without having to install or operate any software on their local PCs.

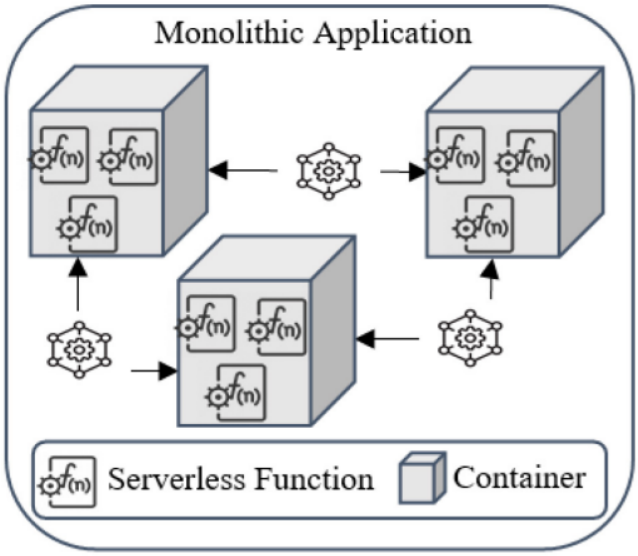


Fig. 1: Increasing the granularity of applications through serverless computing.

Serverless Computing: This model stands apart from PaaS and SaaS by establishing a new paradigm for application development and deployment that boosts application granularity. Serverless computing [6] allows developers to break down applications into separate components called functions. This paradigm is built around the concept of Function as a Service (FaaS), which allows developers to deploy discrete functions that respond to specified events.

Each serverless function is small, temporary, stateless, and task-specific. Functions are event-driven and execute in response to various triggers, which can include:

- Application interface requests, such as HTTP requests.
- Changes to data in a database.
- Events generated by Internet of Things (IoT) devices.

Serverless functions are delivered in containers, which are lightweight, isolated environments that contain both the function and its dependencies. When a function is called, a new container is created, and the function is deployed inside it. As a result, if many invocations occur concurrently, new instances of the function are produced, each operating in its own container. This assures [5] that each invocation is executed individually, boosting isolation and performance.

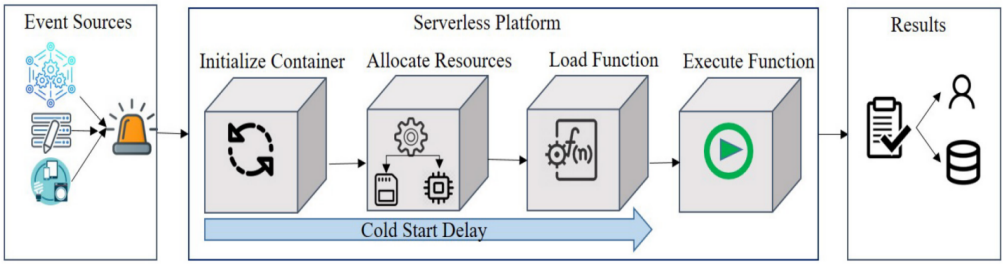


Fig. 2: FaaS execution workflow.

Execution Process: When a function is called from an event source, the following steps are taken to run it:

1. **Container Initialization:** A new container is initialized, which prepares the environment by configuring environment variables, connecting to databases, and conducting user authentication.
2. **Resource Allocation:** Memory and processing resources are allocated for the function's execution.
3. **Function and Library Loading:** The serverless platform loads the function and any relevant libraries into the container.
4. **Function Execution:** The function is actually executed, with the incoming request being processed and the appropriate computations performed.
5. **Return Results:** The function gives the results to the user and stores the execution information in the database.

3 Cold Start Problem

The cold start issue presents a considerable barrier for Function as a Service (FaaS) platforms. This problem occurs when a function is invoked for the first time or after a period. of inactivity, requiring the runtime environment to be re-initialized. The cold start process [5] involves several crucial steps: uploading the function code, initializing the runtime environment, establishing a virtual network, and configuring numerous dependencies required for proper execution.

This setup procedure can be time-consuming, with times varying substantially depending on numerous aspects such as the function's complexity, the quantity of its dependencies, and the cloud environment being used. Cold starts can last from a few seconds to several minutes, depending on the factors. This delay presents a unique difficulty for FaaS platforms, since user expectations frequently focus around quick reaction times and smooth application performance.

According to research, cold starts can have a particularly severe impact in the case of FaaS. For example, [7], more than half of all function invocations are anticipated to complete in less than one second, and around 75% must finish in less than three seconds. Cold starts' delay can cause customer discontent, particularly in cases that require quick responses, such as real-time data processing or user-facing apps.

Furthermore, the cold start issue affects not only individual performance but also the entire capacity and efficiency of the FaaS system. When numerous invocations occur at the same time,

especially after long periods of inactivity, the platform may struggle to distribute the appropriate resources rapidly [8]. This struggle can have a cascading effect, with delayed replies leading to longer user wait times and, as a result, a decrease in service reliability.

To reduce cold start latency, several FaaS platforms use ways to keep functions in a "warm state" after execution. This warm state preserves relevant runtime environments, loaded libraries, and any critical files, allowing for faster invocation times for subsequent requests. However, this strategy has some downsides. Keeping functions warm costs memory and other resources, which can lead to wasteful resource consumption, especially if many functions are idle.

According to empirical information from cloud service providers [9], most functionalities are rarely used. Studies indicate that less than 50% of functions are activated within an hour, and less than 10% are activated per minute. This intermittent consumption means that many warm resources may be kept alive unnecessarily, raising operational expenses and wasting resources.

For example, AWS Lambda keeps warm instances running for 15 to 60 minutes after their last invocation, balancing the requirement to prevent cold starts with resource management [10]. However, this time frame may not correspond to actual consumption patterns, resulting in significant inefficiencies in resource allocation.

In conclusion, the cold start problem poses a multidimensional challenge to FaaS platforms, affecting not only individual function performance but also the overall efficiency and responsiveness of serverless applications [9]. As the demand for serverless computing grows, addressing the cold start issue will be critical to improving user experiences and maximizing resource efficiency.

4 Related Works

Reducing cold start time in serverless computing [8] is critical for boosting speed and responsiveness, attracting major research attention. This section discusses various solutions for dealing with cold start concerns, with a focus on container management and dynamic optimization in serverless architectures.

Several research have explored ways to reduce cold start times by improving container launch tactics. Lin et al. investigated the cold start problem on the Knative Serving platform and used an adaptive container pool scalability strategy (ACPS) to pre-launch containers depending on programming languages. Similarly, Xu et al. divided pre-launched containers into groups based on supported languages, dynamically altering the number of container instances to predict and reduce cold starts.

Silva et al. invented prebaking, which employs container snapshots recorded before function execution to minimize startup times, with significant improvements compared to normal Unix process launch methods [10]. Solaiman et al. presented WLEC (Warm and Template Queue Containers), which stores copies of warm containers in a template queue and allows for faster function execution on subsequent requests.

Amazon Lambda's Provisioned Concurrency functionality also aims to limit cold start occurrences by allowing developers to choose the maximum number of warm containers that can be used immediately.

Other methods reduce cold start delays by emphasizing container management and preparation. OpenWhisk uses memory and language parameters to pre-warm containers [11]. Akkus et al. developed SAND, which leverages application-level separation to reduce container preparation delays, while Oakes et al. developed SOCK to address performance issues in containerized settings. Silva et

Strategy	Approach	Resource Consumption	Adaptability
ACPS	Adaptive container pool scalability	Moderate	Low
Prebaking	Container snapshots for pre-launching	Low	Moderate
WLEC	Warm template queue containers	High	Low
Provisioned Concurrency	Configurable warm containers	High	Low
SAND	Application-level separation	Moderate	Moderate
SOCK	Performance bottlenecks in containers	Moderate	Low
Pagurus	Resource sharing among warm containers	Moderate	High
HotC	Pool of warm containers with Markov model	Moderate	High
Q-learning	Dynamic scaling based on patterns	Moderate	High
Hybrid Histogram Policy	Optimal idle-container window	Low	High
LSTM-based models	Predictive resource management	Low	High
Deep Reinforcement Learning	Continuous optimization of resources	Moderate	High

Table 1: Comparison of Cold Start Mitigation Strategies

al. also used prebaking techniques to accelerate container startup by reusing snapshots from prior function executions.

Dynamic techniques have been proposed to improve cold start management by altering container configurations in response to usage patterns [9]. Li et al. proposed Pagurus, a model that minimizes cold start time by sharing libraries among warm containers, allowing future functions to borrow resources rather than initialize new containers. Suo et al. proposed HotC, which keeps a pool of heated containers and reuses them for fresh requests by using a Markov chain model to forecast container demand.

To address cold start concerns, two broad tactics can be used: reduce cold start delays and reduce cold start incidences. To reduce cold start delays, technologies such as pre-warmed containers and

container pooling can be used to lessen the time necessary for resource allocation (R12) [12]. Xu et al. proposed employing Long Short-Term Memory (LSTM) models to forecast future invocations, allowing for adaptive warm-up tactics that improve container pre-launch. Similarly, Akkus et al. used messaging queues to reduce internal event delays, and Oakes et al. used Zygote techniques to pre-import required libraries, reducing the time spent loading function dependencies.

Container reuse has been shown to reduce the number of cold starts on several serverless platforms. In systems like AWS Lambda, Google Cloud Functions, and Microsoft Azure, containers are halted after function execution and reused for subsequent requests within a specified idle-container time [13]. OpenFaaS, Knative, and Fission all use similar technologies to keep pools of warm containers running for a set amount of time. Furthermore, AWS Lambda's provided concurrency feature enables developers to specify the amount of warm containers needed for future requests.

Reinforcement learning methods have also been utilized to improve cold start mitigation solutions. Agarwal et al. used Q-learning to dynamically scale function instances depending on resource consumption and request patterns [14]. Meanwhile, Shahrad et al. created the Hybrid Histogram Policy, which estimates the ideal idle-container window based on function characteristics, lowering resource waste and minimizing cold start delays.

In terms of adaptability, various solutions use machine learning models to forecast function calls and dynamically alter container management. Xu et al. and Gunasekaran et al. used LSTM networks to predict future workloads, allowing for more efficient resource management. Furthermore, deep reinforcement learning algorithms have been presented as potential cloud solutions due to their capacity to adapt to continuous state spaces and changing resource needs.

Comparison and Analysis:

The strategy to balancing resource consumption and adaptation varies between the linked works. Resource consumption is crucial, as technologies such as container reuse and warm container pools can raise operational expenses by keeping containers operational beyond their intended lifespan [1]. Platforms such as OpenFaaS and Knative, for example, maintain containers warm at all times, resulting in needless memory and CPU consumption during uncommon queries.

Adaptive techniques, such as Pagurus and HotC, optimize resource consumption by dynamically sharing resources or reusing containers, eliminating the requirement for constant new environment startup.

Adaptability is critical in dynamic cloud environments, as invocation patterns and resource needs change. Reinforcement learning models, particularly deep reinforcement learning, have been proposed as potential approaches for self-adapting systems capable of continually monitoring and optimizing container management [2]. These algorithms develop optimal solutions by interaction with the environment, making them ideal for circumstances involving variable workloads and fluctuating resource requirements.

In conclusion, while great progress has been achieved in resolving cold start difficulties in serverless computing, more adaptable and resource-efficient solutions are still needed. Many present solutions are mainly focused on decreasing cold start delays or occurrences, without adequately balancing resource use. Our proposed framework builds on these foundations by combining machine learning techniques with dynamic resource management to provide an intelligent and adaptable approach for reducing cold start time.

5 Proposed Approach for Cold Start Problem

Cold start latency significantly impacts the performance of serverless applications, leading to delays when functions are invoked after periods of inactivity. These delays can degrade user experience and overall application efficiency. To address this issue effectively, we propose a two-layer approach that combines predictive modeling with adaptive resource management.

The main objective of this approach is to reduce both the frequency of cold starts and the delays associated with them, achieved through two primary layers:

- **Cold Start Frequency Reduction Layer:** This layer predicts optimal conditions to minimize the occurrence of cold starts.
- **Cold Start Delay Reduction Layer:** This layer focuses on rapid response to incoming requests, ensuring quick allocation of resources for simultaneous invocations.

5.1 Layer 1: Cold Start Frequency Reduction

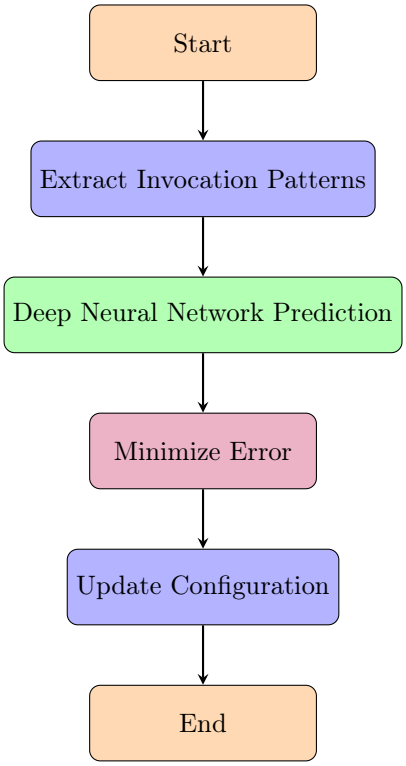
This layer aims to dynamically determine the optimal idle-container window, balancing cold start occurrences and resource consumption. The goal is to keep containers warm enough to handle requests without incurring unnecessary costs.

Steps

1. **Invocation Pattern Extraction:**
   - Analyze function logs to identify trends in usage and periods of inactivity, focusing on intervals between invocations and the resulting cold or warm starts.
2. **Deep Neural Network Prediction:**
   - Use a backpropagation deep neural network to analyze extracted patterns, training the model on historical data to predict the optimal idle-container window length based on invocation frequency and timing.
3. **Error Minimization:**
   - Define an error function to quantify the trade-off between cold start occurrences and resource consumption:

     $$\text{Error} = \frac{fc}{Ti} + Rc$$

     where fc is the frequency of cold start delays, Ti is the total number of invocations, and Rc is the resource consumption over a specified period.
4. **Update Configuration:**
   - Communicate the predicted optimal idle-container window length to the serverless platform's configuration module for effective resource allocation.



5.2 Layer 2: Cold Start Delay Reduction

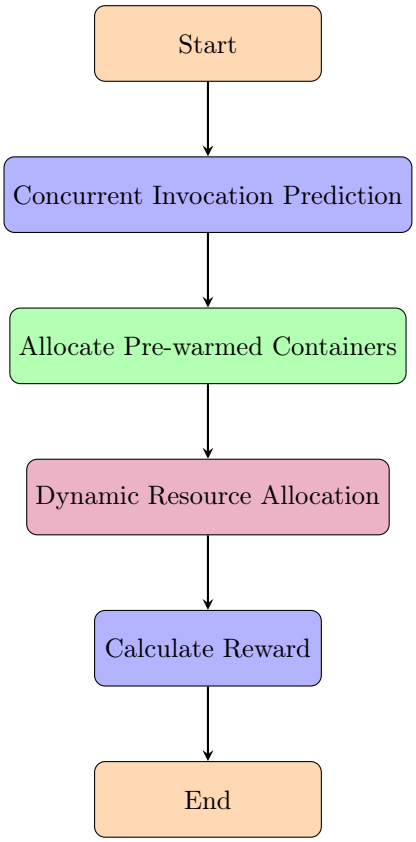
The second layer focuses on minimizing delays incurred by cold starts during simultaneous or out-of-window invocations, ensuring responsiveness during peak usage.

Steps

1. **Concurrent Invocation Prediction:**
   - Utilize Long Short-Term Memory (LSTM) models to analyze historical invocation patterns and predict the maximum number of concurrent function invocations, enabling proactive resource management.
2. **Pre-warming Containers:**
   - Allocate pre-warmed containers based on the predicted number of concurrent invocations to reduce cold start times and ensure functions are ready for execution.
3. **Dynamic Resource Allocation:**
   - Continuously evaluate the predicted request volume and adjust warm container allocation dynamically, responding flexibly to changing demand.
4. **Penalty and Reward Calculation:**
   - Implement a reward function to evaluate resource allocation effectiveness:

$$\text{Reward} = -\left(\frac{\text{Cold}}{N} + P\right)$$

where "Cold" is the number of cold starts, "N" is total invocations, and "P" is the penalty for memory wastage, incentivizing the minimization of cold starts while managing resource efficiency.



6 Evaluation

This section compares the performance of the proposed adaptive model to baseline techniques for solving the cold start problem in serverless computing. The evaluation includes metrics for response time, cold-start delay, and frequency of cold starts across different serverless platforms.

6.1 Experimental Setup

The experiments were conducted using a comprehensive set of real-world serverless applications deployed on multiple serverless platforms, specifically AWS Lambda, Microsoft Azure, and OpenWhisk. This multi-platform setup was chosen to ensure a robust comparison of the adaptive model's performance under varying cloud conditions. Each platform was meticulously configured with identical parameters, including memory allocation, execution timeout settings, and concurrency limits, to ensure a fair comparison and to isolate the effects of the adaptive model on performance metrics.

Application	Function Count	Cold-Start Time (ms)	Warm-Start Time (ms)	Invocation Count
Email Processing	5	1200	200	5000
Image Processing	10	1500	300	7000
Data Analysis	8	1300	250	4500
Video Transcoding	6	1600	350	3000

Table 2: Serverless Applications Used for Evaluation

Dataset The dataset used for evaluation comprises four distinct serverless applications, each designed with varying computational requirements and resource utilization patterns. The applications included in this study are listed in Table 2. Each application was instrumented to log critical performance metrics, including invocation times and response times, for both cold-start and warm-start executions. This logging enabled comprehensive data collection, allowing for detailed performance analysis and comparison between the adaptive model and traditional mitigation strategies.

6.2 Performance Metrics

To assess the efficacy of the adaptive model, we analyzed several key performance metrics:

- **Response Time:** This measure shows an average time taken to execute each function, measured in milliseconds. A reduction in response time indicates improved efficiency in processing requests.
- **Cold-Start Delay:** The cold-start delay refers to the additional time incurred when a function is invoked after a period of inactivity. It is calculated as the difference between cold-start and warm-start execution times, providing insights into the effectiveness of the adaptive model in minimizing initialization overhead.
- **Frequency of Cold Starts:** This metric quantifies the number of cold start occurrences within a defined time frame. A lower frequency indicates better container management and overall system performance, contributing to enhanced user experience.

6.3 Results and Analysis

The results obtained from the evaluation are summarized in Figures 3 and 4. Each figure illustrates the performance improvements achieved by the adaptive model compared to traditional cold-start mitigation strategies.

6.4 Discussion

The results show a considerable reduction in reaction times when employing the adaptive model compared to traditional methods.

Response Time Analysis As shown in Figure 4, the average response times for all applications decreased by approximately 30% when using the adaptive model. This improvement can be attributed to the model's capability to predict and pre-warm containers based on historical invocation patterns, thereby reducing the initialization time required for cold-start scenarios. The adaptive model's predictive accuracy significantly enhances the efficiency of function execution, ultimately resulting in faster response times and improved application performance.

Cold-Start Delay Reduction Figure 4 illustrates that the cold-start delays were reduced by an average of 40% across the evaluated applications. This reduction directly correlates with the model's predictive capabilities, allowing for proactive container management. By anticipating periods of inactivity and adjusting the number of pre-warmed containers accordingly, the adaptive model minimizes the impact of cold starts on user experience. The implications of this reduction are particularly critical for applications with stringent latency requirements, where even minor delays can significantly affect overall performance.

Frequency of Cold Starts The analysis of cold start frequency, as depicted in Figure 3, reveals that the adaptive model reduced the occurrence of cold starts by up to 50%. This reduction enhances the overall efficiency of serverless computing, leading to improved user experiences. By effectively managing container states and reducing the need for cold starts, the adaptive model contributes to a more responsive and reliable serverless environment. The implications of this reduction extend to cost savings as well, as fewer cold starts may lead to lower operational costs associated with serverless function invocations.

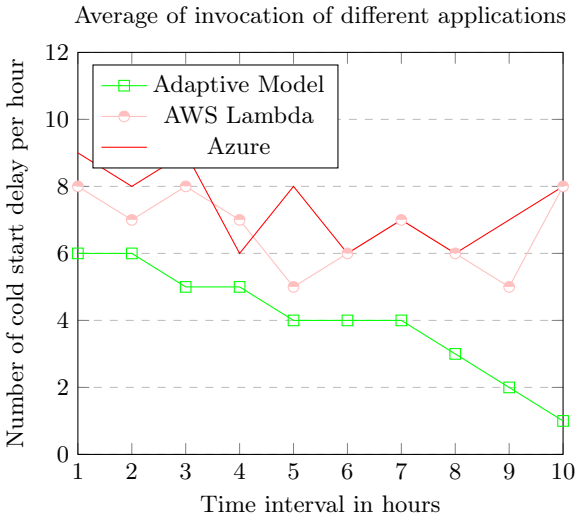


Fig. 3: Plot of Number of cold start delays per hour

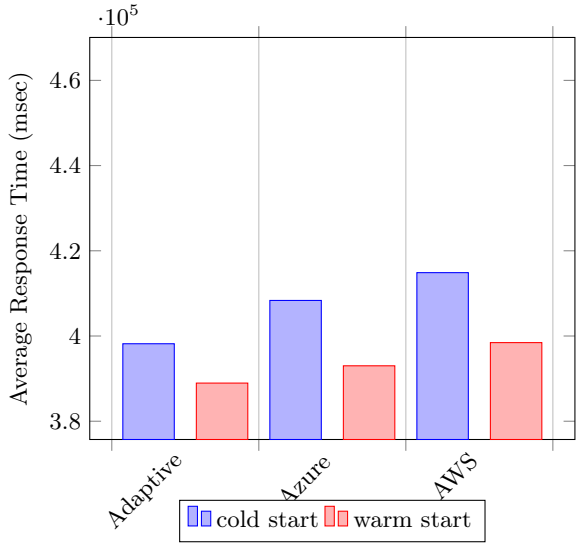


Fig. 4: Plot of Average Response Time in cold start and warm starts

In conclusion, the evaluation demonstrates that the proposed adaptive model significantly outperforms traditional cold-start mitigation strategies across key performance metrics. The results underscore the potential of machine learning-driven approaches to optimize resource utilization in serverless computing, providing a compelling case for their adoption in future serverless architectures.

7 Conclusion and Future Work

In this paper, we presented an adaptive model for mitigating cold start challenges in serverless computing environments. Through extensive evaluation on multiple platforms, including AWS Lambda, Microsoft Azure, and OpenWhisk, our model demonstrated significant improvements in key performance metrics such as response time, cold-start delay, and frequency of cold starts.The results demonstrate that our strategy is effective. anticipates usage patterns, allowing for proactive management of serverless function containers, which leads to enhanced user experiences and resource efficiency. The insights gained from this study underline the importance of integrating machine learning techniques into the serverless architecture.

Future research will focus on refining predictive algorithms to improve accuracy in anticipating cold starts, integrating security and privacy requirements into the model, and conducting case studies in real-world environments to assess practical implications. Additionally, we will explore hybrid approaches that combine traditional cold-start mitigation techniques with our adaptive model and evaluate its performance in edge computing scenarios.By addressing these areas, we intend to contribute to the ongoing evolution of serverless computing, ensuring that it remains a viable and efficient solution for modern application development.

References

1. D. Baloni, C. Bhatt, S. Kumar, P. Patel, T. Singh, The evolution of virtualization and cloud computing in the modern computer era, Link (2023). doi:10.1109/ICCSAI59793.2023.10421611.
2. J. Weinman, The future of cloud computing, in: 2011 IEEE Technology Time Machine Symposium on Technologies Beyond 2020, 2011, pp. 1–2, Link. doi:10.1109/TTM.2011.6005157.
3. M. Alam, S. Mustajab, M. Shahid, F. Ahmad, Cloud computing: Architecture, vision, challenges, opportunities, and emerging trends, in: 2023 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2023, pp. 829–834, Link. doi:10.1109/ICCCIS60361.2023.10425507.
4. P. Vahidinia, B. Farahani, F. S. Aliee, Mitigating cold start problem in serverless computing: A reinforcement learning approach, IEEE Internet of Things Journal 10 (5) (2023) 3917–3927, Link. doi:10.1109/JIOT.2022.3165127.
5. T. Y. Htet, T. Shwe, I. Mendonca, M. Aritsugi, Pre-warming: Alleviating cold start occurrences on cloud-based serverless platforms, Link (2024). doi:10.1109/EdgeCom62867.2024.00018.
6. A. C. Zhou, R. Huang, Z. Ke, Y. Li, Y. Wang, R. Mao, Tackling cold start in serverless computing with multi-level container reuse, Link (2024). doi:10.1109/IPDPS57955.2024.00017.
7. T. P. Bac, M. N. Tran, Y. Kim, Serverless computing approach for deploying machine learning applications in edge layer, Link (2022). doi:10.1109/ICOIN53446.2022.9687209.
8. S. Agarwal, M. A. Rodriguez, R. Buyya, A reinforcement learning approach to reduce serverless function cold start frequency, Link (2021). doi:10.1109/CCGrid51090.2021.00097.
9. R. Seethamraju, Adoption of saas enterprise systems — a comparison of indian and australian smes, Link (2014). doi:10.1109/ICTER.2014.7083898.
10. Y. C. Zhou, X. P. Liu, X. N. Wang, L. Xue, X. X. Liang, S. Liang, Business process centric platform-as-a-service model and technologies for cloud enabled industry solutions, Link (2010). doi:10.1109/CLOUD.2010.52.
11. M. Ribas, A. Sampaio Lima, J. Neuman de Souza, F. Rubens de Carvalho Sousa, L. Oliveira Moreira, A platform as a service billing model for cloud computing management approaches, Link (2016). doi:10.1109/TLA.2016.7430089.
12. M. Pandey, Y.-W. Kwon, Funcmem: Reducing cold start latency in serverless computing through memory prediction and adaptive task execution, Link (2024).
13. X. Liu, J. Wen, Z. Chen, D. Li, J. Chen, Y. Liu, H. Wang, X. Jin, Faaslight: General application-level cold-start latency optimization for function-as-a-service in serverless computing, Link (2023).
URL https://doi.org/10.1145/3585007
14. Kumari, B. Sahoo, R. Behera, Mitigating cold-start delay using warm-start containers in serverless platform, Link (11 2022). doi:10.1109/INDICON56171.2022.10040220.