PULSE: Using Mixed-Quality Models for Reducing Serverless Keep-Alive Cost

Kausalya Sankaranarayanan
Department of ECE
Northeastern University
Boston, USA
sankaranarayanan.k@northeastern.edu

Rohan Basu Roy
Department of ECE
Northeastern University
Boston, USA
basuroy.r@northeastern.edu

Devesh Tiwari
Department of ECE
Northeastern University
Boston, USA
d.tiwari@northeastern.edu

***Abstract*—This paper addresses a key challenge with using serverless computing for machine learning (ML) inference which is cold starts that occur during initial invocations and container inactivity. Fixed keep-alive policies, like the commonly adopted 10-minute strategy, have been implemented by cloud providers to alleviate cold start issues. However, the substantial size of ML models poses a significant hurdle, leading to elevated keep-alive costs and potential strain on system resources. In response to these challenges, we introduce PULSE, a dynamic 10-minute keep-alive mechanism that employs ML model variants to optimize the balance between keep-alive costs, accuracy, and service time while avoiding peaks in keep-alive memory consumption. Our evaluation, using real-world serverless workloads and commonly used machine learning models, demonstrates reduced keep-alive costs compared to the fixed policy. Additionally, we observe that integrating PULSE improves the performance of existing state-of-the-art serverless function warm-up strategies.**

I. Introduction

Serverless computing has emerged as a transformative paradigm in cloud computing, offering a dynamic and cost-effective approach to application development and deployment.

The convergence of serverless computing and Machine Learning (ML) has opened up a new era of possibilities, enabling the deployment and management of ML models in a highly scalable manner.

The challenge of cold starts in serverless computing. Cold starts in serverless computing occur during the initial invocation or following periods of inactivity, where the creation of the container and the loading of the initial code contribute to increased service time.

Cloud provider's approach to cold starts. To address the cold start challenge, cloud providers use a fixed keep-alive policy, where a container is kept alive for a certain period, typically 10 minutes, after its last invocation.

Challenges of a fixed 10-minute keep-alive policy for machine learning models in serverless computing. The substantial size of machine learning models, poses a challenge when employing a fixed 10-minute keep-alive policy. Large models incur substantial keep-alive costs, consuming valuable memory resources without guaranteeing actual usage. This issue is exacerbated during periods of high invocation demand, as the system must keep-alive containers for an extended 10-minute period after the peak has subsided. This keep-alive memory consumption leads to unnecessarily high keep-alive costs and can potentially strain the system's memory resources.

Key idea. To address the inherent limitations of fixed keep-alive policies, we propose PULSE, a dynamic 10-minute keep-alive mechanism that leverages machine learning model variants to achieve a balance between keep-alive cost, accuracy, and service time.

Overall, PULSE makes the following contributions:

- PULSE is a novel machine learning model keep-alive scheme for serverless functions, optimizing accuracy, service time, and keep-alive costs. It utilizes predictive mechanisms based on past invocations and a greedy optimization technique to determine model variant selection and duration within the 10-minute keep-alive period.
- Additionally, PULSE employs a utility value-based strategy for downgrading to lower accuracy model variants during peak keep-alive memory usage. This strategy factors in arrival probability, accuracy benefits, and prior downgrade frequency for decision-making, achieving resource efficiency while maintaining accuracy.
- Evaluation demonstrates a 39.5% reduction in keep-alive costs and an 8.8% improvement in service time in comparison to the OpenWhisk fixed 10-minute keep-alive policy. Furthermore, PULSE enhances the performance of existing serverless techniques when integrated.

II. Motivation

In this section, we commence by showcasing the benefits of introducing a mix of diverse quality models into the serverless execution environment. Subsequently, we examine the complex challenges presented by user invocation patterns observed in serverless workloads that hinder the full realization of this model blending. As a solution to these challenges, we introduce PULSE, which offers comprehensive strategies.

TABLE I: Comparative analysis of model variants: service time, keep-alive cost, and accuracy.

Model	Service Time (with Warmup) (sec)	Keep Alive Cost (cents/hour)	Accuracy (Percent)
GPT-Small	12.90	11.7	87.65
GPT-Medium	22.50	22.57	92.35
GPT-Large	23.66	41.71	93.45
BERT-Small	1.09	4.392	79.6
BERT-Large	2.21	6.12	82.1
DenseNet-121	1.09	3.46	74.98
DenseNet-169	1.38	3.53	76.2
DenseNet-201	1.65	4.07	77.42

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00021

Machine learning models frequently come in various variants, each presenting unique design trade-offs, particularly pertinent in serverless computing. As illustrated in Table I, the highest quality variant excels in accuracy but at the expense of longer service times and higher keep-alive costs – characteristics less desirable in the serverless landscape. In contrast, the lowest quality variant maintains lower accuracy but offers reduced service times and keep-alive costs. Keeping-alive this high-quality variant continuously to prevent cold starts translates to substantial keep-alive costs. Hence, the judicious warming of high-quality models solely during invocation periods, when their usage likelihood is notably high, becomes an essential strategy for cost-effective serverless operation. Simultaneously, the utilization of lower-quality models when there's even a slight chance of invocation prevents these models from incurring high service times due to cold starts, albeit at the trade-off of slightly lower accuracy compared to their high-quality counterparts. By judiciously combining models, particularly incorporating lower quality models that consume less keep-alive memory i.e the memory consumed in keeping a container alive, under scenarios where numerous models have a high likelihood of invocation, providers can leverage available memory resources to accommodate all the models, optimizing both resource allocation and the ability to deliver warm starts to models with the highest invocation probability. The result is a balance between quality and resource efficiency, reshaping the serverless ecosystem and enhancing the experience of both users and providers.

Next, we substantiate our arguments with empirical evidence sourced from real-world machine learning models listed in Table IV [5], [10], [11], [26], [27] and a production-level serverless workload trace [31]. Our objective is to highlight the critical challenges that need to be tackled to fully leverage the benefits of reduced service time and keep-alive costs, achieved through the integration of diverse quality models, as opposed to relying solely on high-quality variants.

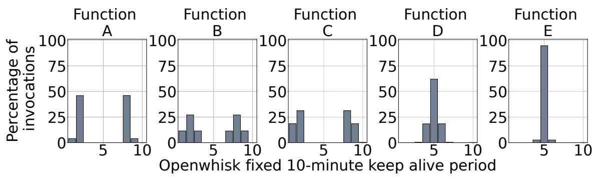

Fig. 1: Diverse inter-arrival patterns are observed among various functions within the 10-minute timeframe following invocation (represented on the x-axis) .

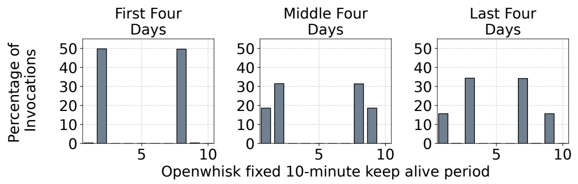

Fig. 2: Different inter-arrival time patterns are observed across different periods for the same function within the 10-minute timeframe following invocation (represented on the x-axis) .

Observation 1. Dynamic inter-arrival patterns call for adaptive keep-alive policies in serverless environments.

TABLE II: Peak I evaluation

	Service Time (sec)	Keep-alive Cost (USD)	Accuracy (Percent)
All High Quality	1799.49	0.86	77.81
All Low Quality	902.38	0.39	71.41
Random High Quality Low Quality	1246.05	0.61	76.13
Intelligent Solution	1661.80	0.78	76.85

TABLE III: Peak II evaluation

	Service Time (sec)	Keep-alive Cost (USD)	Accuracy (Percent)
All High Quality	1771.12	0.9	78.01
All Low Quality	912.94	0.40	71.62
Random High Quality Low Quality	1246.92	0.62	76.26
Intelligent Solution	1648.79	0.78	77.02

Figure 1 showcases the diverse inter-arrival time patterns exhibited by different functions sharing a common serverless environment. This visualization underscores the need for a keep-alive policy to accommodate to the unique invocation patterns of each function, dispelling the notion of a one-size-fits-all strategy. Additionally, Figure 2 reinforces the same notion, here a dynamic inter-arrival pattern occurs across varying periods, for the same function. This indicates that rigid policies, such as the standard 10-minute fixed keep-alive policy commonly used by major serverless service providers (including OpenWhisk, Azure Functions, AWS, and Google Functions) where the functions are kept-alive for 10 minutes after an invocation, are suboptimal in the presence of such dynamic patterns. The decision of when to keep a model alive and for how long it should remain alive should adapt to these evolving invocation patterns.

This discussion has predominantly addressed inter-arrival patterns, but it has not delved into the significant implications for critical factors such as keep-alive cost, service time, and accuracy. In the following discussion, our objective is to underscore the need to optimize all functions collectively. This entails identifying the functions that need to be kept alive and specifying the variants of these functions that should be kept alive, based on the potential performance gains in terms of accuracy, reduction in keep-alive costs, and service time these keep-alive decisions could yield.

Observation 2: The existence of peak invocation periods necessitates the collective consideration of all invoked functions for efficient optimization in serverless computing.

In the production-level serverless workload trace, we identified numerous peaks in invocations (cumulative for all concurrent functions in the serverless environment) . For analysis, we designate two prominent peaks , characterized by the highest volume of invocations. We comprehend the performance implications of the 10-minute fixed keep-alive policy for these peak periods. Following the peak, the functions are kept alive for 10 minutes regardless of their likelihood of invocation during this period, incurring substantial keep-alive memory and cost requirements. We assessed the performance of four distinct approaches during this 10-minute keep-alive periods following these two peak periods when functions run machine learning models in Table II and Table III respectively, evaluating their effectiveness in terms of keep-alive cost and accuracy.In terms of service time, the objective is to achieve higher warm starts.

In this scenario, all four approaches exhibit an equal number of warm starts.

The first approach involved keeping alive high-quality models for the entire 10 minutes following an invocation, resulting in the highest service time,keep-alive cost, and the best accuracy. In contrast, the second approach entailed keeping-alive low-quality models throughout the full 10-minute period for all functions with an invocation, leading to significantly lower accuracy, coupled with the lowest service time and keep-alive cost.

The third approach introduced the concept of blending models of varying qualities, employing random decisions to determine which functions would have high-quality models kept-alive and which would have low-quality models. While these decisions were randomized, we ensured that the number of functions with high-quality and low-quality models kept-alive was balanced. This approach achieved a middle ground, with keep-alive cost falling between those of the first and second approaches, and accuracy close to using only high-quality models.

In the fourth approach, we implemented an intelligent solution wherein functions with a higher number of actual invocations during the 10 minutes had high-quality models kept alive, while others utilized low-quality models. This approach achieved accuracy levels even closer than the third approach to those of high-quality models while maintaining relatively lower keep-alive costs compared to scenarios where only high-quality models were kept alive.

Our findings reveal that real-world scenarios involve sudden spikes in invocations, resulting in corresponding peaks in keep-alive memory and costs. In such contexts, our approach, integrated into PULSE, considers all functions collectively and allocates model variants judiciously based on the performance gains in terms of keep-alive cost, accuracy, and service time achieved by the assignment. This approach ensures the optimized management of keep-alive memory and cost during unpredictable peaks, which would otherwise go unnoticed when focusing exclusively on individual functions.

III. PULSE:Key Ideas and Design

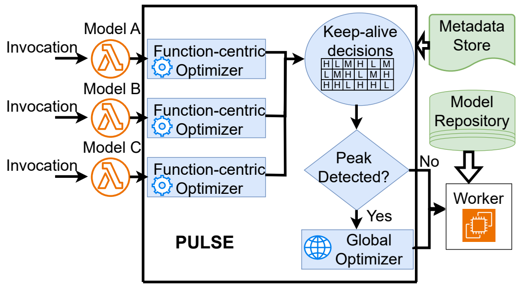

Fig. 3: Overview of the design of PULSE.

We begin by providing an overview of PULSE (Figure 3) .

(1) Individual function optimization. The PULSE system utilizes historical data for which inter-arrival times are calculated, and associated probabilities for each inter-arrival time are computed. Probability thresholds are determined based on the number of model variants. Employing a greedy optimization technique utilizing the computed probability and probability thresholds, the system selects the model variant to be kept alive and specifies the duration for the keep-alive of each variant within the 10 minutes following an invocation. While this approach effectively reduces keep-alive costs, challenges associated with keep-alive memory peaks persist, necessitating unbiased function downgrades.

(2) Cross-function optimization. During a peak, PULSE employs cross-function optimization to determine which function to downgrade. This decision is based on a utility value calculated from the function's accuracy, the frequency of downgrades to lower accuracy variants in the past, and the probability of invocation during the peak.

Next, we delve into these two components in detail.

A. PULSE:Individual Function Optimisation

Why is an individual function optimization needed?

In the 10-minute fixed keep-alive policy followed by serverless providers, if an invocation occurs in the 2nd minute, the container remains active for a constant 10-minute duration, extending until the 12th minute. Serverless providers face abrupt peaks in invocations (refer Section II) . During these peaks, if invoked functions are associated with machine learning models, the subsequent 10-minute period experiences significantly high keep-alive memory usage due to the substantial memory consumption of machine learning models, ranging between 300 and 3500 MB. To alleviate this keep-alive memory, we introduce various quality variants of the machine learning models within these 10 minutes. Individual function optimization is employed to determine which model variant to keep-alive and for how long.

How is individual function optimization implemented?

To achieve this, we utilize historical data from two time periods: the immediate past, referred to as the local window, and the duration until the most recent invocation. We employ two time periods because function inter-arrival times can vary (refer Figure 2) over time. Therefore, it is necessary to examine both the immediate period and the entire duration the system has been operational. For each of these periods, we calculate the inter-arrival time of invocations, and the time resolution used for inter-arrival time is in minutes. We calculate the probabilities associated with the inter-arrival times during the keep-alive period. For example, when the inter-arrival time of 2 appears 10 times, we compute the probability of 2 as 10 divided by the total number of inter-arrival times. Subsequently, we calculate the average of the probabilities obtained for both periods.

We employ a greedy optimization technique to determine which variant to keep-alive, guided by the computed probabilities. For determining the probability threshold we divide the probability space, ranging between 0 and 1, into the number of variants. For N variants, we require N-1 thresholds (computed as 1/N,2/N…N-1/N) , dividing the probability space into N areas. The lowest accuracy variant is assigned to the area with the lowest probabilities and so on.

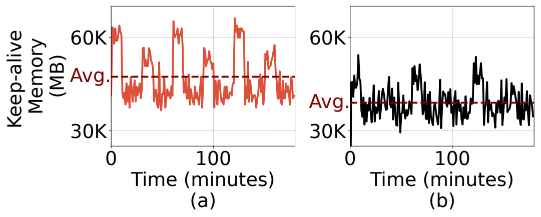

Fig. 4: (a) OpenWhisk's fixed keep-alive policy exhibits high and sudden peaks in keep-alive memory. (b) Individual function optimization reduces keep-alive memory but does not eliminate peaks. Therefore, there is a need for a cross-function optimization technique.

This method facilitates the selection of model variants based on calculated probabilities while minimizing overhead, by employing a greedy optimization approach. The greedy optimization can be tuned by the provider based on available resources and specific needs. As demonstrated in Section V , PULSE remains effective as long as the general principle of keeping alive the variant with the highest accuracy at higher invocation probabilities is followed.

The memory, a finite resource for serverless providers, is shared between actual invocations and keep-alive. Individual function optimization, as depicted in Figure 4, reduces keep-alive memory; however, peaks persist. During peak memory consumption when total memory consumption exceeds available resources, random functions/models are downgraded, which may result in models with higher-chance of invocation being downgraded while lower-chance models are kept alive. We address this issue by implementing unbiased downgrades to flatten peaks, as discussed in the following section.

B. PULSE:Cross-function Optimisation

How is the concept of a "peak" defined in our approach?

We define a peak at a specific minute if its keep-alive memory exceeds the prior keep-alive memory by a threshold, calculated as the product of the keep-alive memory threshold and the prior keep-alive memory. This keep-alive memory threshold serves as a tunable parameter depending on the system's capacity to withstand memory stress. Functions exhibit variability in their activity patterns. Some experience periods of inactivity between two time periods, while others may be nocturnal or diurnal. When invocations occur for such functions, setting the prior keep-alive memory based on the immediate past period's keep-alive memory value (equal to 0 in circumstances after inactivity) would result in a high number of cold starts. Conversely, some functions have a consistent pattern of invocations. The design of the prior keep-alive memory takes into account these diverse conditions. These challenges are particularly relevant during the first minute of the keep-alive period. For subsequent minutes, the approach is straightforward: the prior keep-alive memory is always the keep-alive memory consumed in the previous minute.

Algorithm 1 Peak Determination

```
Require: Current keep-alive memory (C_KaM) , sliding local window duration (l_window) and keep-alive memory threshold (KM_T)
 1: Define: P_KaM : Prior keep-alive memory
 2: function IsPeak(C_KaM, P_KaM)
 3:     if C_KaM $>$ P_KaM $+$ KM_T $\times$ P_KaM then
 4:         return True
 5:     else
 6:         return False
 7:     end if
 8: end function
 9: for $t$ = 1 to len (keep-alive period) do
10:     if $t$ equal to 1 then
11:         $\overline{KaM}$ $\leftarrow$ Avg. Keep-alive memory for l_window
12:         if $T \geq 2 \times$ l_window and $\overline{KaM}>0$ then
13:             P_KaM= $\overline{KaM}$
14:         else
15:             for $q$ in 1 to $T$ do
16:                 Q_KaM $\leftarrow$ Keep-alive memory at q
17:                 ( P_KaM = Q_KaM $>0$ ? Q_KaM : $\infty$
18:             end for
19:         end if
20:     else
21:         P_KaM = Keep-alive memory of $t$-1
22:     end if
23:     IsPeak(C_KaM, P_KaM)
24: end for
```

Algorithm 1 illustrates the methodology used in this paper to determine the prior keep-alive memory after an invocation at period T. The methodology is designed to tackle the scenarios mentioned earlier. When the PULSE system has been operational for at least local window minutes but has an immediately previous period of inactivity, the prior keep-alive memory is set to the last non-zero keep-alive memory value. Conversely, if there has been continuous activity, the prior keep-alive memory is calculated as the average keep-alive memory over the last local window minutes.

How do we determine which models and associated variants should be kept alive during a "peak"?

When we identify a period as a peak, we have access to specific information. This includes the model variants selected for keep-alive during the peak, performance meta-data for all model variants, and the computed invocation probabilities from our individual function optimization. Next, we will delve into the various components utilized by PULSE to compute the utility value of a specific keep-alive decision, based on the available data.

Accuracy improvement. For each model selected to be kept alive during a peak, we need to consider two variants: the chosen keep-alive variant and one with slightly lower accuracy. We compute the accuracy improvement provided by the chosen keep-alive variant in comparison to the other variant. When the chosen variant is the lowest accuracy option,

Algorithm 2 Utility Value Computation for Model Keep-Alive Decisions

```
1: Initialize the priority structure as an array with zeros for all models. This initialization occurs immediately after the system has started.
2: for every time period $t$ classified as peak do
3:     while Peak is not flattened do
4:         Normalise the priority structure
5:         for every model that is kept-alive in $t$ do
6:             Compute $A_i$ and $P_r$
7:             $U_v \leftarrow A_i + P_r + I_p$
8:         end for
9:         Downgrade the model $m$ with lowest $U_v$
10:        Update Priority Structure with +1 for $m$
11:    end while
12: end for
```

as there is no further lower variant to keep the model alive, the accuracy improvement is equivalent to the accuracy of this lowest quality variant in decimal form. Accuracy improvement values range from 0 to 1. However, a challenge arises when using accuracy improvement as the sole criterion. For example, if YOLO's lowest accuracy variant has an accuracy of 56.8%, and GPT's lowest accuracy variant has an accuracy of 87.65%, relying solely on accuracy improvement to determine which model to downgrade would introduce bias and consistently prioritize GPT over YOLO when both are competing to be kept alive during a peak. To mitigate this bias, the next component has been introduced.

Priority. We maintain a count of how many times a model has been downgraded within a structure known as the Priority structure. When a peak occurs, we employ the normalization formula presented in Equation 1 to standardize the values within the structure. This gives a priority value for each model, ranging between 0 and 1. The model that has experienced the most downgrades will be assigned the highest priority value. To minimize memory overhead, the priority structure is implemented as an array.

$$X_{\text{normalized}} = \begin{cases} \dfrac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}, & \text{if } X_{\text{max}} \neq X_{\text{min}} \\ X - X_{\text{min}}, & \text{if } X_{\text{max}} = X_{\text{min}} \end{cases} \quad (1)$$

Where:
$X_{\text{normalized}}$: Normalized Value
X: Original Value
X_{max}: Maximum Value
X_{min}: Minimum Value

Probability of invocation. This value is derived in the individual function optimization. It plays a crucial role in computing the impact of a model keep-alive decision, offering insights into the probability of the function being invoked.

Using the above three components (accuracy improvement (A_i) , normalized priority (P_r) , and probability of invocation (I_p)) the utility value (U_v) is calculated during a peak for every model chosen to be kept-alive as follows:

$$U_v = A_i + P_r + I_p \quad (2)$$

Each of these components ranges from 0 to 1. To ensure a balanced assessment and prevent bias, the three components are equally weighted, holding equal significance in determining the utility value of a model keep-alive decision.

As shown in Algorithm 2, during a peak, the utility value for all the models selected to be kept alive is computed. The model with the lowest utility value is downgraded by one variant and receives an additional point in the priority structure. This process continues until the peak is flattened. In this way, when there is a sudden spike in keep-alive memory, instead of randomly selecting models for cold starts, models with the lowest utility value are chosen to have warm starts with models having lower accuracy, or even cold starts. This process is unbiased, ensuring that one model doesn't consistently bear the burden of downgrading itself due to the existence of the priority structure.

TABLE IV: Model Families and their variants used.

Model Family	Model Variants	Dataset
BERT (sentiment analysis) [5]	BERT-base,BERT-large	sst2
YOLO (object detection) [27]	s,l,x	COCO
GPT (text generation) [26]	base,medium,large	wikitext
ResNet (image classification) [10]	50,101,152	CIFAR-10
DenseNet (image classification) [11]	121,169,201	CIFAR-10

IV. Experimental Methodology

Experimental platform. We conducted a simulation of the setup. To achieve this, we constructed models as Docker images, which were subsequently stored in an AWS Elastic Container Registry (ECR) . These Docker images were then leveraged within AWS Lambda functions and invoked using Python 3.10 scripts that incorporated the Boto3 library. To ensure efficient operation and handle a substantial volume of invocations, we configured the AWS Lambda memory size to be twice the size of the ECR image.

Workloads. The evaluation of PULSE is driven by machine learning models and real-world workload traces [31]. To ensure the diversity and realism of our experiments, we employed a varied set of machine learning models, each designed to perform distinct tasks mirroring the diversity of real-world workloads. The models selected for our experiments encompass some of the most commonly used machine learning models, including BERT, YOLO, GPT, ResNet, and DenseNet. Table IV provides details about the machine learning models employed, the dataset used for inputs, and their respective variants. In a demonstration of versatility, we also explored the presence of multiple variants. For instance, BERT featured two variants, while ResNeT exhibited three variants. This approach highlights PULSE's capability to adapt to various model configurations. To ensure optimal performance and resource efficiency, we opted for the ONNX version of these models, as it required fewer dependencies that consumed less memory compared to the PyTorch version.

In terms of dataset inputs, we made use of widely recognized datasets within the machine learning domain. These datasets provided a rich and diverse range of inputs, allowing us to effectively capture the intricacies of machine learning models in the serverless context, including service time, which comprises both cold start time and execution time.

To mirror a production-like workload, we utilized the Microsoft Azure Function trace, which contains two weeks' worth of serverless invocation data from Microsoft's production systems. This trace offered valuable insights into the inter-arrival times of functions, the memory allocations for each function, and their corresponding execution times. It's worth noting that PULSE, in its experimentation, utilized the inter-arrival of 12 functions observed in the Azure trace,previously employed by Wild and IceBreaker to illustrate PULSE's standalone efficacy and integration potential with state-of-the-art methodologies. These functions represent different invocation patterns encountered in serverless environments, ensuring our approach's alignment with real-world usage patterns.

State-of-the-art serverless function warm-up strategies. The cutting-edge strategies for serverless function warm-up are:

(a) *Serverless in the Wild (Wild)* [31]: This approach employs a hybrid histogram-based technique to predict inter-arrival times of various serverless functions, incorporating the ARIMA model for functions with a heavy-tailed histogram.

(b) *IceBreaker* [28]: IceBreaker utilizes a fast Fourier-based method to forecast inter-arrival times of diverse serverless functions. It also employs a utility function for making selections among heterogeneous nodes.

It is noteworthy that these techniques were not designed to accommodate various machine learning model variants to reduce keep-alive costs while ensuring optimal performance. This feature is unique to PULSE. Our experimental findings demonstrate that integrating PULSE to these techniques helps them achieve better performance. Once techniques like Wild and IceBreaker forecast the inter-arrival times of functions, PULSE takes the lead in determining which model variant should be kept active and for how long. For this experiment, we used only one type of node for both IceBreaker and Wild, thereby eliminating the need for utility function computation in IceBreaker.This approach proves superior to the conventional practice of invoking high-quality models indiscriminately.

Competing keep-alive strategy. The performance of PULSE is compared with OpenWhisk's policy, which keeps the function alive for 10 minutes after invocation. However, this policy cannot predict the timing of subsequent invocations. Furthermore, it is not tailored to handle different machine learning model variants for minimizing keep-alive costs while ensuring optimal performance. OpenWhisk strategy aligns with those of other major commercial serverless providers like AWS, Google, and Azure Functions.

Metrics. Our experimental analysis focuses on three key performance metrics: service time, keep-alive cost, and accuracy.
Service Time: This metric encompasses the combined time for a function, including cold-start time and execution time. When an invoked function experiences a warm start, it incurs zero cold-start time.
Keep-Alive Cost: Keep-alive cost refers to the total monetary expenses incurred by the service provider in maintaining a function in memory. This cost reflects the resources allocated to keep functions readily available.
Accuracy: The accuracy of the system is calculated as the sum of the accuracy delivered by each model during invocations, divided by the total number of invocations. This metric provides an overview of the system's overall precision in delivering results.

Simulation. To investigate the performance of PULSE over many runs (each run with different model-to-function assignments) we established a simulation framework. This setup commenced by characterizing the behavior of ML models within a serverless environment. We initiated this process by executing the Lambda function for each model and its associated variants using 1000 distinct inputs drawn from the datasets.

To measure the service times during a cold start, we used a technique that revolved around manipulating the memory size of the Lambda function. By modifying the memory size, a new container is generated, leading to a cold start for the next invocation after this adjustment. Our procedure began with the initial execution of a function to determine its cold start time. Following this, we adjusted the memory size of the function to an arbitrary value, conducted a dummy invocation, and subsequently reverted the memory size to its original setting. Running the function again after these alterations allowed us to precisely gauge the service time during a cold start.

To assess service times during warm start, we performed a dummy run followed by 1000 consecutive runs. These consecutive runs ensured that the containers remained active, leading to warm starts for each of these 1000 invocations.

The cost associated with keeping functions alive was determined based on Amazon Web Services pricing , which amounts to $16.67 for every KB-second. For accuracy assessments, we relied on values provided by the authors of relevant papers.

Using the gathered data, we conducted 1000 simulation runs, each presenting a unique combination of model-to-function assignments. Within each simulation run, we navigated through the entire trace period for the 12 most commonly used functions from the trace. For each invocation, we employed PULSE to decide which variant should be warmed up and for what duration in the subsequent 10-minute window following the invocation. These decisions informed the computation of the total service time, the cumulative keep-alive cost, and the average accuracy for each run, guided by predetermined values.

V. Evaluation and Analysis

This section presents an analysis of PULSE and evaluates its performance relative to OpenWhisk. Furthermore, we assess the performance of state-of-the-art techniques when integrated with PULSE.

In Figure 5, the 10-minute keep-alive periods, when using only the low-quality model, exhibit low accuracy and keep-alive costs. Conversely, employing solely the high-quality

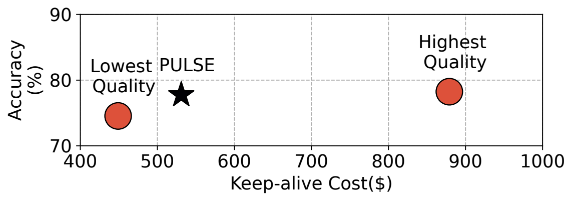

Fig. 5: Analyzing the trade-off between accuracy and keep-alive cost in scenarios where only high-quality models, only low-quality models, and PULSE, a hybrid technique combining different quality models is employed during 10-minute keep-alive periods.

model yields high accuracy along with increased keep-alive costs. In contrast, PULSE attains a keep-alive cost similar to the lowest quality while preserving accuracy levels closely aligned with the highest quality.

In Figure 6, the performance of PULSE is depicted in comparison to the OpenWhisk fixed keep-alive policy.

Keep-Alive cost. *PULSE reduces the overall keep-alive cost for the service provider by 39.5% compared to the OpenWhisk 10-minute fixed keep-alive policy.*

In Figure 6 (a) , the overall keep-alive cost, representing the total expenditure for keep-alive of containers hosting the models and their dependencies, is observed to be lower for PULSE compared to the OpenWhisk policy. This difference arises because, during the 10-minute keep-alive period after an invocation, OpenWhisk consistently keeps the containers with high-quality model alive, regardless of the likelihood of invocation. In contrast, PULSE dynamically decides which containers to keep-alive, guided by the likelihood of future invocations. When the likelihood of invocation is high, PULSE keeps the containers with the high accuracy model alive, while during periods of low invocation likelihood, it keeps the containers with low accuracy model alive. This adaptive approach allows PULSE to minimize the keep-alive cost while maximizing the number of warm starts.

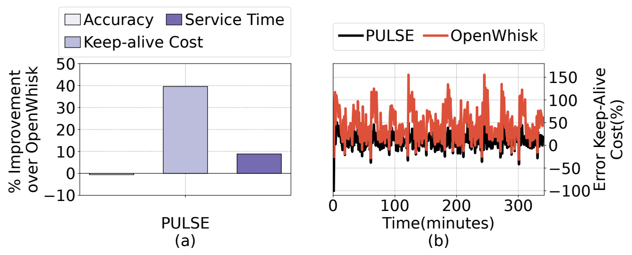

Fig. 6: PULSE improves the overall keep-alive cost and service time when compared to OpenWhisk's fixed 10-minute keep-alive policy despite a slight reduction in accuracy.

Service time. *PULSE reduces the total service time by 8.8% compared to the fixed keep-alive policy.*

Figure 6 (a) illustrates the total service time, representing the cumulative service times of all functions throughout the entire duration. It is presented as a percentage improvement relative to the OpenWhisk 10-minute fixed keep-alive policy. The primary objective of PULSE is to reduce the keep-alive cost while maintaining an equivalent number of warm starts as OpenWhisk. Even in scenarios with hard-to-predict patterns encountered by the function-centric optimization, PULSE ensures that at least the container with low-quality model is kept alive every 10 minutes after an invocation, preventing cold starts during that timeframe. Additionally, in response to keep-alive memory usage spikes, PULSE dynamically selects which containers to keep alive based on their utility values and may downgrade to containers with lower-quality variants. This strategic decision potentially results in instances where containers with lower quality models are kept alive instead of allowing invocations to randomly undergo cold starts, contributing to a reduction in service times.

Accuracy. The accuracy obtained by using PULSE closely approaches (with a 0.6% decrease) the best accuracy while maintaining a lower keep-alive cost and service time.

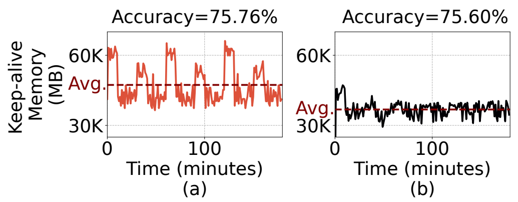

Fig. 7: OpenWhisk's fixed 10-minute keep-alive policy (a) creates memory peaks, while PULSE (b) reduces keep-alive memory and smooths memory peaks.

Keep-alive memory. *PULSE reduces overall keep-alive memory consumption compared to the OpenWhisk policy, while simultaneously eliminating sudden peaks in keep-alive memory usage.*

Figure 7 (a) illustrates the keep-alive memory consumption when employing the 10-minute fixed keep-alive policy. The high keep-alive memory consumption is likely attributed to the 10-minute fixed keep-alive policy maintaining all invoked functions containers in an active state for the entire 10-minute duration, regardless of their utilization. This leads to a keep-alive memory wastage. Additionally, there are abrupt spikes in keep-alive memory usage due to sudden peaks in invocation (refer to Section II) . The average accuracy achieved by the machine learning models assigned to the functions during this run is 75.76%. Figure 7 (b) presents the results achieved by PULSE. Here, the keep-alive memory consumption has been reduced, with no abrupt spikes, and with a minimal accuracy drop of 0.16%. PULSE achieves this through function-centric optimization, which aims to keep containers with lower-accuracy models alive when the invocation probability is low within the 10 minutes following an invocation, and global optimization, which reduces keep-alive memory consumption spikes using a utility value-based methodology to downgrade models to lower accuracy variants in an unbiased manner.

Why does PULSE outperform OpenWhisk?

Figure 6 (b) displays the deviation in keep-alive costs for each minute, comparing both PULSE and OpenWhisk to the ideal value of keep-alive cost, where the model is only kept alive during the time it is invoked. The time resolution in this context is in minutes.

Figure 6 (b) highlights that the keep-alive cost associated with the OpenWhisk 10-minute policy is often higher compared to the ideal scenario. In contrast, PULSE maintains a keep-alive cost comparatively closer to the ideal value. The improved performance of PULSE compared to OpenWhisk in terms of keep-alive cost is primarily attributed to the strategic mixing of different quality models within the 10-minute keep-alive period using function-centric optimization. Specifically, the inter-arrival time-based probability of invocation proves highly effective. This ensures that the high-quality model is kept alive precisely during the period (at minute resolution) of an invocation. In many instances of actual invocation, the container with higher accuracy model is kept alive. Conversely, in situations with a lower likelihood of invocation, the container with lower-quality model is kept alive.

Integrating with Existing Techniques. *PULSE effectively functions both as a standalone technique and when integrated with existing state-of-the-art techniques.*

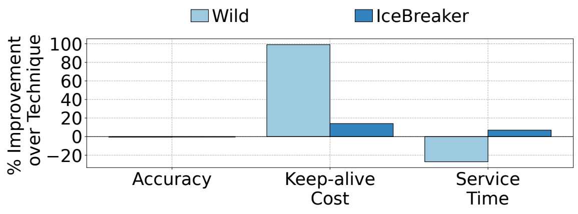

Fig. 8: Illustrates the performance achieved by integrating PULSE into Wild and IceBreaker,as compared to using the original technique.

As outlined in Section IV, the state-of-the-art techniques, Wild and IceBreaker, are not machine-learning model variant aware. Moreover, they do not enforce a memory constraint to prevent peaks in keep-alive memory consumption. Consequently, we aim to integrate both these components into these techniques to assess whether the incorporation of PULSE complements and enhances their performance.

Figure 8 illustrates the trade-off achieved between keep-alive cost, service time, and accuracy upon integrating PULSE into Wild. This integration preserves Wild's predicted concurrency, followed by PULSE's function-centric and global optimization. We observe a 99% reduction in keep-alive cost, accompanied by a 27.1% increase in service time and a 0.6% reduction in accuracy.

Figure 8 illustrates the performance impact of integrating PULSE into IceBreaker. PULSE is incorporated after the function invocation predictor, which determines the concurrency of subsequent periods using past invocation patterns in IceBreaker. The integration of PULSE into IceBreaker demonstrates improvements in both keep-alive cost and service time. We observe a 14% reduction in keep-alive cost, and a 7% decrease in service time, with a 0.5% drop in accuracy.

The core idea and design behind PULSE are flexible and can be adapted to different keep-alive durations depending on the specific needs of the provider.

Overhead. *PULSE incurs a low overhead which makes it well-suited for practical deployment in real-world scenarios.* Mixed Integer Linear Programming (MILP) is an alternative

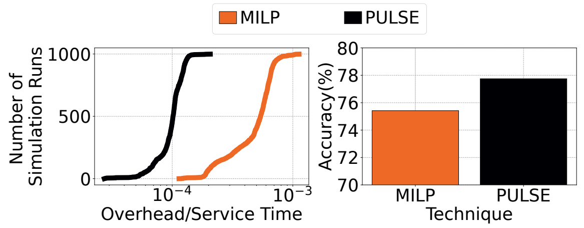

Fig. 9: (a) Compares the overhead incurred by PULSE in contrast to the Mixed Integer Linear Programming optimization technique. (b) Compares the accuracy of Mixed Integer Linear Programming with that of PULSE.

technique for achieving an optimized trade-off between keep-alive cost, service time, and accuracy while preventing keep-alive memory spikes. MILP is used to solve optimization problems where the objective is to find a set of values for the variables that maximizes or minimizes a given objective function, subject to a set of constraints. In this problem, the objective is to maximize overall utility value subject to a strict memory budget constraint.MILP simultaneously evaluates all selected models and their variants, aiming to identify the combination that maximizes utility value while adhering to the memory budget constraint. Figure 9 (a) illustrates that MILP incurs considerably higher overhead compared to PULSE and in Figure 9 (b) , there is an accuracy reduction incurred by MILP in comparison to PULSE as MILP tends to favor lower-quality models due to lack of iterative adaptability. The high overhead makes MILP impractical for production deployments in serverless environments due to their demanding workload characteristics.

PULSE's overhead remains minimal even when handling a large number of concurrent functions. This scalability demonstrates PULSE's suitability for real-world serverless environments.

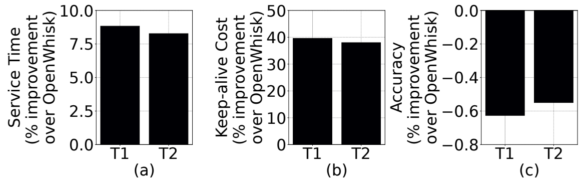

Fig. 10: PULSE proves effective across diverse probability threshold values, serving as criteria for model warm-up selection.In this figure Technique T1 partitions the invocation probability range into N areas, while Technique T2 reserves the variant with the lowest accuracy for zero probability invocations and divides the remaining range into N-1 areas.Here N is the number of variants.

Effectiveness on different probability, keep-alive memory thresholds and local window sizes. *PULSE exhibits*

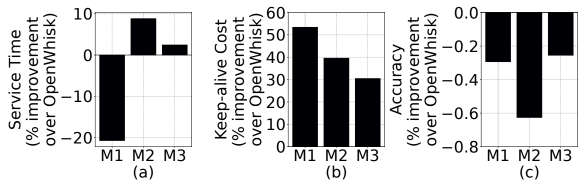

Fig. 11: PULSE demonstrates its effectiveness across various memory constraints. In this figure, M1 represents a keep-alive memory threshold of 5%, M2 represents 10%, and M3 represents 15%.

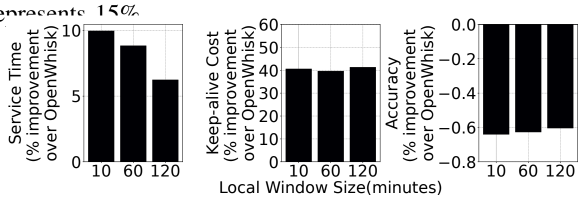

Fig. 12: PULSE demonstrates its effectiveness across various local window sizes.

effectiveness across various probability thresholds in function-centric optimization , keep-alive memory thresholds in global optimization and local window sizes.

In Figure 10, we compare two probability threshold techniques. Technique T1 corresponds to the threshold mechanism outlined in the greedy optimization approach for establishing probability thresholds, as described in Section III. In this method when the total number of variants of a model is N, the invocation probability range (always 1) is divided into N areas. The number of probability thresholds is equal to N-1.

Technique T2, on the other hand, keeps alive the variant with the lowest accuracy when the probability of invocation is 0. For values of probability of invocation greater than 0 and less than or equal to 1, technique T2 divides the probability of invocation into N-1 areas. Here the number of probability thresholds is equal to N-2.

Figure 10 demonstrates that both probability threshold determining techniques, T1 and T2, produce comparable results. This suggests that the effectiveness of PULSE is not significantly influenced by the specific probability threshold selection, as long as the general principle of keeping alive the variant with the highest accuracy at higher invocation probabilities is adhered to. This observation highlights the robustness and adaptability of PULSE in handling diverse probability threshold schemes.

Figure 11 illustrates the performance of PULSE for varying keep-alive memory thresholds . The outcomes are expressed as an improvement over the OpenWhisk 10-minute fixed keep-alive policy. Here, M1 signifies a 5% keep-alive memory threshold, M2 denotes a 10% keep-alive memory threshold, as discussed in Section III, and M3 represents a 15% keep-alive memory threshold. PULSE achieves a balance between keep-alive cost, service time, and accuracy for every memory constraint. This demonstrates that PULSE consistently performs well across a range of keep-alive memory thresholds.

Figure 12 illustrates the performance of PULSE across various local window sizes. PULSE demonstrates equilibrium among keep-alive cost, service time, and accuracy for each local window size. This indicates the consistent performance of PULSE across the spectrum of local window sizes.

VI. Related Work

Serverless Workload Characterization. Numerous studies have examined the behavioral patterns of commercial serverless platforms, considering user perspectives through representative benchmarks [9], [18], [21], [34], [38]. Multiple studies have also presented detailed workload trace analysis [30], [31], [35], [36]. Other studies reveal diverse trends in serverless computing, covering scheduling effects, invocation patterns, and I/O behaviors [8], [14]–[16], [20], [22]. Most prior serverless studies have acknowledged the impact of cold starts on the service time [1], [19], and esp. during the peak periods. In summary, previous research have quantified the significance of cold start times and keep-alive costs [6], [9], [23], [32], [33]. But, they have not proposed or employed the use of mixed-quality machine learning models to reduce the serverless keep-alive cost, esp. during bursty periods.

Machine Learning Inference Service. Previous research extensively explores aspects of ML inference services, such as adaptive query batching [2], [3], [7], and energy efficiency on GPU clusters [12], [17], [24] and battery-powered devices [13], [37], [39]. There are prior works on how ML models can be of different qualities [25] and they can be deployed in different scenarios [4], [25], [29]. This is the first work to show that mixed-quality ML models can be leveraged to reduce the cost of providing serverless ML inference services – PULSE provides that carefully navigating the competing trade-offs among accuracy, cost, and service time. PULSE significantly reduces the cost and improves service time with minimal impact on accuracy.

VII. Conclusion

PULSE proposes a novel approach to address the challenges of cold starts and minimize keep-alive costs while avoiding sudden spikes in keep-alive memory associated with machine learning models in serverless environments. This is achieved through the utilization of diverse quality model variants.

Acknowledgement. We thank the reviewers for their constructive feedback. This work is supported by NSF Award 1910601, 2124897, and Northeastern University.

Appendix: Artifact Description

Artifact Description (AD)

VIII. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 PULSE introduces a novel scheme tailored for serverless functions, leveraging model variants to optimize accuracy, service time, and keep-alive costs.

C_2 PULSE utilizes a predictive mechanism based on past invocations, combined with a greedy optimization technique, to determine which model variant to keep alive and for how long within the 10-minute keep-alive period.

C_3 PULSE devises a utility value-based strategy for downgrading to lower accuracy model variants in response to sudden peaks in keep-alive memory. By considering factors such as arrival probability, the accuracy benefit of retaining the current model variant, and the frequency of prior downgrades for a given function for decision-making it selects model variants to balance resource efficiency and accuracy during peaks.

B. Computational Artifacts

A_1 https://zenodo.org/records/10976369

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1,$C2$,$C3$	Figure 6(a),8

IX. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

A_1 has the source code necessary to conduct simulations experiments described in PULSE. Through these simulations, we aim to evaluate the influence of PULSE on various metrics such as keep-alive cost, service time, and the accuracy of machine learning models in serverless environments. By analyzing the outcomes of these experiments, we can gain valuable insights into the effectiveness and practical implications of integrating PULSE into serverless architectures.

Expected Results

PULSE reduces keep-alive cost with a minimal reduction in accuracy when used standalone or when integrated into other techniques.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact is approximately 1950 minutes for the simulation(1.95 minutes per run in a x86_64, 8-core Intel i7).

Artifact Setup (incl. Inputs)

Extract the 'simulation' folder present in the artifact in the local system.

Artifact Execution

For the simulation experiment, navigate to each technique's respective folder and execute the corresponding Python files. To run PULSE, execute the file located at '/openwhisk/with_technique_T1.py'. For OpenWhisk, execute the file at '/openwhisk/openwhisk_only_high.py'. For Icebreaker, execute the file at '/icebreaker/icebreaker_only_high.py'. For PULSE integrated with Icebreaker, execute the file at '/icebreaker/with_technique_T1.py'. For Wild, execute the file at '/wild/wild_only_high.py'. Finally, for PULSE integrated with Wild, run the file located at '/wild/with_technique_T1.py'.The simulation experiments consist of 1000 runs, with each run covering the entire trace period. In each run, the assignment of models to functions varies.

Artifact Analysis (incl. Outputs)

PULSE reports its accuracy in "technique_accuracy_sliding_with_memory_constraint_T1.txt".The simulation stores total service time per run in "technique_servicetime_sliding_with_memory_constraint_T1.txt". Lastly "technique_keepalive_cost_sliding_with_memory _constraint_T1.txt" stores the total keep-alive cost.Similarly, the other techniques generate .txt files containing service time, keep-alive cost, and accuracy metrics. Using these files, we compute simulation results by averaging the values across all runs for service time, accuracy, and keep-alive cost. Subsequently, we quantify the improvement achieved by implementing PULSE over the original technique. These computed values are then integrated into the plots.ipynb file, which utilizes Matplotlib to generate the bar plots.

References

[1] S. Boucher, A. Kalia, D. G. Andersen, and M. Kaminsky, "Putting the" micro" back in microservice," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 645–650.

[2] Y. Choi, Y. Kim, and M. Rhu, "Lazy batching: An sla-aware batching system for cloud machine learning inference," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 493–506.

[3] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system." in *NSDI*, vol. 17, 2017, pp. 613–627.

[4] D. Daimary, M. B. Bora, K. Amitab, and D. Kandar, "Brain tumor segmentation from mri images using hybrid convolutional neural networks," *Procedia Computer Science*, vol. 167, pp. 2419–2428, 2020, international Conference on Computational Intelligence and Data Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920307614

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[6] S. Eismann, J. Scheuner, E. Van Eyk, M. Schwinger, J. Grohmann, N. Herbst, C. L. Abad, and A. Iosup, "Serverless applications: Why, when, and how?" *IEEE Software*, vol. 38, no. 1, pp. 32–39, 2020.

[7] K. Fu, J. Shi, Q. Chen, N. Zheng, W. Zhang, D. Zeng, and M. Guo, "Qos-aware irregular collaborative inference for improving throughput of dnn services," in *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2022, pp. 993–1006.

[8] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, A. Bruno, J. Hu, B. Ritchken, B. Jackson *et al.*, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 3–18.

[9] J. R. Gunasekaran, P. Thinakaran, N. C. Nachiappan, R. S. Kannan, M. T. Kandemir, and C. R. Das, "Characterizing bottlenecks in scheduling microservices on serverless platforms," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1197–1198.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[12] D.-K. Kang, K.-B. Lee, and Y.-C. Kim, "Cost efficient gpu cluster management for training and inference of deep learning," *Energies*, vol. 15, no. 2, p. 474, 2022.

[13] T. Kannan and H. Hoffmann, "Budget rnns: Multi-capacity neural networks to improve in-sensor inference under energy budgets," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 143–156.

[14] A. Klimovic, H. Litz, and C. Kozyrakis, "Selecta: Heterogeneous cloud storage configuration for data analytics," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 759–773.

[15] A. Klimovic, Y. Wang, C. Kozyrakis, P. Stuedi, J. Pfefferle, and A. Trivedi, "Understanding ephemeral storage for serverless analytics," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 789–794.

[16] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, 2018, pp. 427–444.

[17] T. Komoda, S. Hayashi, T. Nakada, S. Miwa, and H. Nakamura, "Power capping of cpu-gpu heterogeneous systems through coordinating dvfs and task mapping," in *2013 IEEE 31st International Conference on computer design (ICCD)*. IEEE, 2013, pp. 349–356.

[18] H. Lee, K. Satyam, and G. Fox, "Evaluation of production serverless computing environments," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 2018, pp. 442–450.

[19] Z. Li, Q. Chen, S. Xue, T. Ma, Y. Yang, Z. Song, and M. Guo, "Amoeba: Qos-awareness and reduced resource usage of microservices with serverless computing," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 399–408.

[20] X. C. Lin, J. E. Gonzalez, and J. M. Hellerstein, "Serverless boom or bust? an analysis of economic incentives," in *12th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 20)*, 2020.

[21] W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly, and S. Pallickara, "Serverless computing: An investigation of factors influencing microservice performance," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 159–169.

[22] N. Mahmoudi and H. Khazaei, "Performance modeling of serverless computing platforms," *IEEE Transactions on Cloud Computing*, 2020.

[23] A. Mohan, H. Sane, K. Doshi, S. Edupuganti, N. Nayak, and V. Sukhomlinov, "Agile cold starts for scalable serverless," in *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[24] S. M. Nabavinejad, S. Reda, and M. Ebrahimi, "Batchsizer: Power-performance trade-off for dnn inference," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 819–824.

[25] M. A. Ouassil, B. Cherradi, S. Hamida, E. Mouaad, O. el Gannour, and A. Raihani, "A fake news detection system based on combination of word embedded techniques and hybrid deep learning model," *International Journal of Advanced Computer Science and Applications*, vol. 13, 01 2022.

[26] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:160025533

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[28] R. B. Roy, T. Patel, and D. Tiwari, "Icebreaker: Warming serverless functions better with heterogeneity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 753–767. [Online]. Available: https://doi.org/10.1145/3503222.3507750

[29] A. K. Sahu, S. Sharma, M. Tanveer, and R. Raja, "Internet of things attack detection using hybrid deep learning model," *Computer Communications*, vol. 176, pp. 146–154, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366421002164

[30] M. Shahrad, J. Balkind, and D. Wentzlaff, "Architectural implications of function-as-a-service computing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 1063–1075.

[31] M. Shahrad, R. Fonseca, I. n. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC'20. USA: USENIX Association, 2020.

[32] P. Silva, D. Fireman, and T. E. Pereira, "Prebaking functions to warm the serverless cold start," in *Proceedings of the 21st International Middleware Conference*, 2020, pp. 1–13.

[33] D. Taibi, J. Spillner, and K. Wawruch, "Serverless computing-where are we now, and where are we heading?" *IEEE Software*, vol. 38, no. 1, pp. 25–31, 2020.

[34] D. Ustiugov, P. Petrov, M. Kogias, E. Bugnion, and B. Grot, "Benchmarking, analysis, and optimization of serverless function snapshots," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 559–572.

[35] A. Wang, S. Chang, H. Tian, H. Wang, H. Yang, H. Li, R. Du, and Y. Cheng, "Faasnet: Scalable and fast provisioning of custom serverless containerruntimes at alibaba cloud function compute," *arXiv preprint arXiv:2105.11229*, 2021.

[36] L. Wang, M. Li, Y. Zhang, T. Ristenpart, and M. Swift, "Peeking behind the curtains of serverless platforms," in *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, 2018, pp. 133–146.

[37] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, "Approximate query service on autonomous iot cameras," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, 2020, pp. 191–205.

[38] T. Yu, Q. Liu, D. Du, Y. Xia, B. Zang, Z. Lu, P. Yang, C. Qin, and H. Chen, "Characterizing serverless platforms with serverlessbench," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 30–44.

[39] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.