SCOPE: Performance Testing for Serverless Computing

JINFENG WEN, Beijing University of Posts and Telecommunications, China
ZHENPENG CHEN, Nanyang Technological University, Singapore
JIANSHU ZHAO, Beijing University of Posts and Telecommunications, China
FEDERICA SARRO, University College London, United Kingdom
HAODI PING, Beijing University of Technology, China
YING ZHANG, Peking University, China
SHANGGUANG WANG, Beijing University of Posts and Telecommunications, China
XUANZHE LIU*, Peking University, China

Serverless computing is a popular cloud computing paradigm that has found widespread adoption across various online workloads. It allows software engineers to develop cloud applications as a set of functions (called *serverless functions*). However, accurately measuring the performance (i.e., end-to-end response latency) of serverless functions is challenging due to the highly dynamic nature of the environment in which they run. To tackle this problem, a potential solution is to apply checks of performance testing techniques to determine how many repetitions of a given serverless function across a range of inputs are needed to cater to the performance fluctuation. However, the available literature lacks performance testing approaches designed explicitly for serverless computing. In this paper, we propose *SCOPE*, the first serverless computing-oriented performance testing approach. *SCOPE* takes into account the unique performance characteristics of serverless functions, such as their short execution durations and on-demand triggering. As such, *SCOPE* is designed as a fine-grained analysis approach. *SCOPE* incorporates the accuracy check and the consistency check to obtain the accurate and reliable performance of serverless functions. The evaluation shows that *SCOPE* provides testing results with 97.25% accuracy, 33.83 percentage points higher than the best currently available technique. Moreover, the superiority of *SCOPE* over the state-of-the-art holds on all functions that we study.

CCS Concepts: • **Computer systems organization → Cloud computing**; • **Software and its engineering → Software performance**.

Additional Key Words and Phrases: serverless computing, performance testing

1 INTRODUCTION

Serverless computing is becoming a mainstream cloud computing paradigm that has been widely adopted in various online workloads like big data analytics, deep learning, large language models (LLM), and so on [31, 52, 60, 80–82, 85, 91]. It frees software engineers from tedious and error-prone infrastructure management and allows them to focus on developing a cloud application as a set of event-driven functions, called *serverless functions* [75]. To support the execution of serverless functions, mainstream cloud vendors have provided serverless platforms, such as AWS Lambda [2] and Google Cloud Functions [14]. It is predicted that the market size of serverless computing is projected to grow significantly, reaching $41 billion in 2028, compared to $19 billion in 2024 [19]. It indicates that an increasing number of developers will pivot to develop serverless functions.

[*] Corresponding author

Authors' addresses: Jinfeng Wen, Beijing University of Posts and Telecommunications, Beijing, China, jinfeng.wen@bupt.edu.cn; Zhenpeng Chen, Nanyang Technological University, Singapore, Singapore, zhenpeng.chen@ntu.edu.sg; Jianshu Zhao, Beijing University of Posts and Telecommunications, Beijing, China, jianshuzhao@bupt.edu.cn; Federica Sarro, University College London, London, United Kingdom, f.sarro@ucl.ac.uk; Haodi Ping, Beijing University of Technology, Beijing, China, haodi.ping@bjut.edu.cn; Ying Zhang, Peking University, Beijing, China, ying.zhang@pku.edu.cn; Shangguang Wang, Beijing University of Posts and Telecommunications, Beijing, China, sgwang@bupt.edu.cn; Xuanzhe Liu, Peking University, Beijing, China, liuxuanzhe@pku.edu.cn.

The surging popularity of serverless computing has led to heightened interest among diverse research communities, including Software Engineering (SE) and Systems [74]. Notably, performance stands out as the most extensively studied aspect in serverless computing research [74]. However, it is challenging to obtain accurate and reliable performance (i.e., end-to-end response latency) measurements for serverless functions due to the following reasons. Serverless platforms, where serverless functions are executed, have a highly dynamic cloud underlying infrastructure. This introduces various challenges for accurate and reliable performance measurement such as multi-tenancy, resource management, and networking issues [32, 33, 37, 50, 54, 64, 83]. Serverless functions are generally short-lived tasks that require a small memory size to be configured to provide the resource [67, 68]. This results in a high-density deployment environment, increasing the risk of performance fluctuations [63, 77, 89]. Under these circumstances, serverless functions can produce highly fluctuating performance results even with multiple identical runs when executed on serverless platforms. Moreover, developers design a variety of serverless functions with different functionalities, possibly with different levels of performance fluctuations.

To alleviate this issue, a straightforward way is to set a fixed number of measurement repetitions for all evaluated serverless functions, according to experiment repetitions used in prior studies. The results obtained from measurements are used as the accurate and reliable performance for the serverless function. However, due to the diversity of serverless functions developed by various developers, it is evident that not all serverless functions require or benefit from the same level of repetition. Therefore, using a fixed number of repetitions is unreasonable and ineffective in determining the actual performance for different serverless functions executed on different platforms. A better strategy would be to devise a method for recommending a customized number of repetitions for each serverless function, thereby achieving a more accurate and reliable performance evaluation.

To achieve this goal, a possible solution is to use performance testing techniques, standard procedures for obtaining and evaluating the performance of a software application in SE [24, 35, 90]. Generally, the performance testing technique is conducted by repeatedly executing the application-under-test with a set of inputs until a stopping criterion deems that the performance results obtained from the test are accurate [21, 35, 36, 57, 59, 70]. Performance testing techniques are important for serverless computing. Serverless functions can be part of user-facing features where end-to-end response time directly impacts user experience, thus demanding performance-critical considerations. Other serverless functions can be invoked infrequently, such as end-of-month reports or daily reminder emails. Although such functions are often tolerant of cold start latencies, characterizing their performance remains crucial to ensure that service-level objectives (SLOs) are met and to make informed cost-benefit decisions regarding their deployment. Given the diversity of use cases, it is necessary for the developers of serverless functions to employ effective performance testing techniques. However, to the best of our knowledge, the literature lacks a performance testing approach tailored to serverless functions.

In this paper, we propose *SCOPE*, the first serverless computing-oriented performance testing approach. *SCOPE* takes into account the unique performance characteristics of serverless functions, such as their short execution durations and on-demand triggering. Our primary goal with *SCOPE* is to provide a novel stopping criterion to determine a specific repetition number to obtain highly accurate and reliable performance profiles for each given serverless function. *SCOPE* poses a strict requirement on accuracy and utilizes the accuracy check and the consistency check to determine the stopping criterion for repeated runs. The accuracy check first utilizes the non-parametric confidence interval to analyze whether a specific performance profile is accurate. For the performance result set of the current test, if most of its performance is determined to be accurate, this set is considered

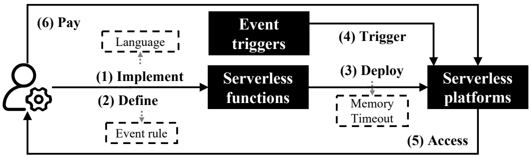

Fig. 1. The process of using serverless computing.

accurate. The consistency check examines whether the performance result set of the current test and the performance result set obtained from the previous run intervals are both accurate. If this is true, *SCOPE* deems that the performance result set of the current test is sufficient to represent the accurate and reliable performance of the serverless function and terminate the repeated runs.

We evaluate the effectiveness of *SCOPE* and state-of-the-art techniques developed for traditional cloud applications by investigating 65 serverless functions from existing work [76]. We use the performance results of 1,000 identical runs (which is the largest number of runs used in the literature) of a serverless function as its ground-truth performance. The evaluation of the 65 serverless functions shows that *SCOPE* provides testing results with 97.25% accuracy, 33.83 percentage points higher than the best currently available technique. Furthermore, *SCOPE* is widely effective, as it outperforms state-of-the-art techniques on all serverless functions that we consider. In contrast to the indiscriminate implementation of a fixed repetition strategy, *SCOPE* shows enhanced flexibility and efficacy in determining a specific repetition and achieving accurate and reliable performance across diverse serverless functions.

To the best of our knowledge, our research is the first to explore both (1) a performance testing technique specifically tailored for serverless computing and (2) an empirical study on the effectiveness of performance testing techniques in serverless computing. These contributions constitute the *novelty* of our work. We also provide a public repository [17] including all the data and code used in this study to facilitate future replication and extension.

2 BACKGROUND

2.1 Serverless computing

In serverless computing, developers focus on application implementation based on serverless functions. Fig. 1 depicts the process of using serverless computing for developers. (1) First, developers implement event-driven serverless functions using high-level programming languages, e.g., Python and JavaScript [1, 16, 29, 30]. (2) Second, developers can define specific rules that bind their functions to the corresponding events, e.g., HTTP requests and data updates in cloud storage. (3) Then, serverless functions are deployed to the serverless platform along with dependent libraries, e.g., *Numpy*. During this phase, developers can provide specific function configurations, e.g., memory size and timeout time [34, 74]. (4) When the serverless function is triggered by predefined events, the serverless platform automatically launches new function instances (e.g., containers or virtual machines) or reuses existing ones to process requests. (5) Upon completion of executions, the serverless platform logs information related to function execution, allowing developers to access and review it later. (6) Finally, developers pay for the cost according to the number of requests and the resources actually allocated or consumed by the serverless function [48, 74].

2.2 Serverless function performance

The performance of serverless computing has gained widespread attention in the serverless computing literature [34, 45, 56, 74]. Researchers have proposed various solutions to optimize serverless function performance [27, 48, 49, 67, 68, 84, 86]. Serverless function performance can be classified into two types: cold-start performance and warm-start performance. When the serverless function is executed on newly launched instances in the serverless platform, it will produce cold-start performance. If the serverless platform has reusable instances for the same function to handle requests within a short keep-alive time (e.g., 7 minutes for AWS Lambda [6]), the serverless function will produce warm-start performance.

The serverless function performance that we focus on is end-to-end response latency, i.e., the time period between sending a request to invoke a serverless function hosted in the serverless platform and receiving the execution result of the function. End-to-end response latency is the most common metric used for performance evaluation and optimization of serverless functions [29, 54, 75, 79, 89]. It contains the serverless platform's preparation time, the function's task processing time, etc. In this paper, we study both cold-start and warm-start end-to-end response latencies.

2.3 Motivation

We present examples of serverless function performance across multiple runs. For instance, we observe *Func52* and *Func30*, which are part of our dataset described in Section 4.3. The maximum and minimum values of the cold-start end-to-end response latencies of *Func52* can vary by as much as 45.83% between runs. Similarly, the warm-start end-to-end response latencies of *Func30* can differ by 369.24%, with the maximum value being 4.69 times greater than the minimum. These examples show the significant variability in serverless function performance, even with repeated, identical runs. Furthermore, as shown in Fig. 15, the bars represent the number of repetitions needed to achieve reliable performance across different serverless functions. The results demonstrate that each serverless function requires a tailored number of repetitions to ensure accuracy and reliability. Overall, these observations emphasize the need for a method that recommends a customized number of repetitions for each serverless function to obtain accurate and reliable performance.

3 OUR PERFORMANCE TESTING APPROACH: *SCOPE*

It is crucial to obtain accurate and reliable performance for serverless functions. However, the existing serverless computing literature does not offer performance testing techniques specifically for it. Therefore, designing a performance testing approach for serverless computing is necessary. Generally, the primary goal of a performance testing approach is to determine whether the tested performance results accurately reflect the actual performance, guiding the decision on whether additional repeated runs from another run interval are necessary. The run interval refers to the specific number of repetitions or trials needed for executing the serverless function. Within each run interval, the serverless function is executed multiple times to acquire new performance data samples. To motivate our approach design, we first summarize key characteristics of serverless functions.

3.1 Key characteristics

● *Serverless functions run for short duration.* The response latency of serverless functions is often measured in milliseconds [67, 89]. However, short-lived serverless functions executed on serverless platforms with highly dynamic cloud underlying infrastructure can exhibit significant performance fluctuations [63, 76, 89].

● *Serverless functions are executed at small run intervals.* In one run interval of performance testing, which refers to the specific number of repetitions required to execute the serverless function, the number of performance results obtained is generally small. This is because the serverless computing scenario inherently uses small repetitions. (1) Serverless functions can be triggered at any time, allowing developers and researchers to obtain performance results as needed. Small

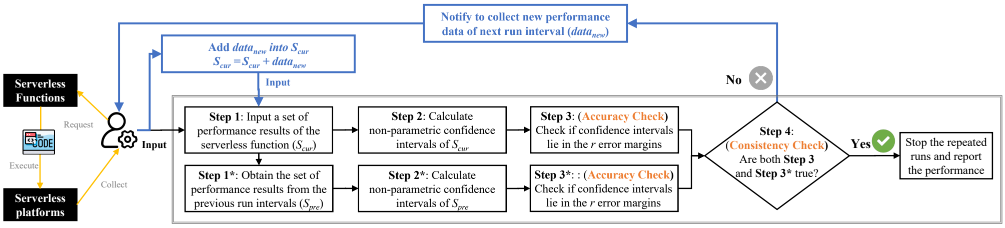
Notify to collect new performance data of next run interval ($data_{new}$)
Add $data_{new}$ into S_{cur}
$S_{cur} = S_{cur} + data_{new}$
Input
No
Serverless Functions
Request
CODE
Execute
Collect
Serverless platforms
Input
Step 1: Input a set of performance results of the serverless function (S_{cur})
Step 2: Calculate non-parametric confidence intervals of S_{cur}
Step 3: (**Accuracy Check**) Check if confidence intervals lie in the r error margins
Step 1*: Obtain the set of performance results from the previous run intervals (S_{pre})
Step 2*: Calculate non-parametric confidence intervals of S_{pre}
Step 3*: : (**Accuracy Check**) Check if confidence intervals lie in the r error margins
Step 4: (**Consistency Check**) Are both **Step 3** and **Step 3*** true?
Yes
Stop the repeated runs and report the performance

Fig. 2. The workflow of ***SCOPE***.

repetitions serve as the foundation, as developers and researchers pay based on the number of invocations and resource consumption. Consequently, they tend to invoke serverless functions with small repetitions on demand. (2) In research work on serverless computing, prior experimental evaluations have employed a predefined number of runs performed on serverless functions for performance analysis. This number is generally small repetitions [77], e.g., 3, 10, and 20 times, thus obtaining a small number of performance results.

Based on these key characteristics, there is a pressing need for a fine-grained, high-accuracy performance testing approach specifically tailored to serverless computing. Such an approach would facilitate accurate and reliable performance measurements and obtain a specified number of repetitions for serverless functions.

3.2 Overview of *SCOPE*

We propose *SCOPE*, a performance testing approach for serverless computing. *SCOPE* is an automated performance tester that provides accurate and reliable performance for serverless functions. To use *SCOPE*, developers collect the performance result set of a function with a given input from the serverless platform and provide it to *SCOPE*. *SCOPE* determines whether this set accurately reflects real performance and whether more repeated runs from another run interval are necessary. A run interval represents the specific number of repetitions required to trigger and execute the serverless function. *SCOPE* provides a fine-grained and high-accuracy guaranteed stopping criterion to determine if the performance result set of the current test is sufficiently accurate and reliable to terminate the execution and collection of further repeated runs. If the performance result set of the current test passes this stopping criterion, it is considered accurate and reliable to represent the actual performance of the serverless function. Otherwise, developers are notified to collect performance results of the serverless function by requesting more repeated runs from another run interval.

Fig. 2 gives the workflow of *SCOPE*. First, developers input a set of end-to-end response latencies generated by the serverless function to be evaluated. These performance results are denoted as S_{cur} (*Step 1*). Second, *SCOPE* calculates the required values (e.g., the non-parametric confidence interval for the median [43]) and then performs the accuracy check (*Steps 2 and 3*). Note that the non-parametric confidence interval allows us to estimate percentile performance without relying on strong assumptions about the data distribution. We also provide three specific methods for calculating these non-parametric confidence intervals in Section 3.4. To ensure the reliability of the performance result acquired with our approach, *SCOPE* also conducts the same performance accuracy analysis for the performance result set obtained from the previous run intervals, denoted as S_{pre}. S_{pre} is obtained by removing performance data produced by the last run interval from the end of the S_{cur} (*Step 1**). *SCOPE* then calculates the required values for S_{pre} and conducts the accuracy check (*Step 2** and *Step 3**). Next, *SCOPE* performs the consistency check of the stopping criterion to check whether the returned results of *Step 3* and *Step 3** are both true (*Step 4*), i.e., satisfying the accuracy check. If yes, *SCOPE* can give information that the performance result set of the current test S_{cur} is available to represent the actual serverless function performance, and no further repeated runs are needed. Otherwise, *SCOPE* waits for developers to collect and provide new end-to-end response latencies when triggering the serverless function repeatedly (i.e., for the number of repetitions specified by the run interval). These new performance data are denoted as $data_{new}$ and added to S_{cur}, resulting in an updated set of performance results for evaluation. *SCOPE* conducts a new round of performance testing using the same processing steps.

3.3 Stopping criterion of *SCOPE*

Our stopping criterion includes the accuracy check and the consistency check. The accuracy check is designed to ensure the high accuracy of the obtained performance result, while the consistency check aims to alleviate the possible influence of result fluctuations and ensure result stability.

For the accuracy check, we assume that a specific performance can be accurate if there exists a desired non-parametric confidence interval for that performance. In our work, the presence of such a desired confidence interval for a given performance is denoted as dci. Specifically, *SCOPE* checks if the confidence intervals for specified metrics (at the given confidence level $cl\%$), calculated from the performance result set of the current test (S_{cur}), lie within the corresponding $r\%$ error margins of the observed true value for metrics. If it is yes, there exists the desired confidence intervals for specified metrics. For example, *SCOPE* determines whether the calculated *95% confidence interval for the 50th percentile* lies within a 1% error margin of the observed true $50th$ percentile performance. In other words, *SCOPE* determines whether dci_{50th} exists. Adopting such an accuracy check aims to find the desired and accurate non-parametric confidence interval (CI), where the obtained empirical value of a specific metric (e.g., $50th$ percentile) differs from its observed true performance by no more than the $r\%$ error at a given confidence level. To improve the overall accuracy of the performance result acquired with *SCOPE*, *SCOPE* extends the same performance accuracy check to most performance of S_{cur}. Most performance of a performance distribution generally includes performance results from the $25th$ percentile to the $75th$ percentile [57, 70, 89]. Moreover, the $50th$ percentile performance represents the midpoint of the performance distribution and plays an important role in performance analysis [57, 70]. Thus, our accuracy check transforms into whether the CIs for the $25th$, $50th$, and $75th$ percentiles, calculated from S_{cur}, fall within the $r\%$ error margin of their respective observed true percentile performance, referred to *Step 3* in Fig. 2. In other words, *SCOPE* checks whether dci_{25th}, dci_{50th}, and dci_{75th} exist. If it is satisfied, S_{cur} is considered accurate. Thus, the accuracy check is formulated as (1).

$$\text{AccuracyCheck} = \begin{cases} \text{True} & \text{if } (dci_{25th} = \text{True}) \land (dci_{50th} = \text{True}) \land (dci_{75th} = \text{True}) \\ \text{False} & \text{otherwise} \end{cases} \tag{1}$$

Our approach primarily relies on CI since it can achieve a robust analysis in the face of random performance fluctuations [70]. Moreover, CI has been established as a valuable metric in performance engineering [43, 57, 70]. Consequently, we are based on CI to design and customize our accuracy check for performance testing of serverless computing in our work.

For the consistency check, *SCOPE* checks whether both S_{cur} and the performance result set obtained from the previous run intervals (S_{pre}) are accurate. This is because potential performance fluctuations could make S_{cur} get a temporary result of meeting the accuracy check. However, performance data might be unstable and thus yield unreliable testing results. Thus, we take the accuracy of S_{pre} into account. Our consistency check is to ensure the accuracy of both S_{cur} and S_{pre}, formulated as (2).

$$\text{ConsistencyCheck} = \begin{cases} \text{True} & \text{if } (AccuracyCheck_{S_{cur}} = \text{True}) \land (AccuracyCheck_{S_{pre}} = \text{True}) \\ \text{False} & \text{otherwise} \end{cases} \tag{2}$$

Therefore, we deem S_{cur} to be stable when S_{pre} obtained from previous run intervals also exhibits most of its performance to be accurate. That is to say, for both S_{cur} and S_{pre}, the calculated CIs for the $25th$, $50th$, and $75th$ percentiles all lie within the $r\%$ error margin of the observed true $25th$ percentile performance, $50th$ percentile performance, and $75th$ percentile performance, respectively. This process refers to *Step 4* in Fig. 2.

3.4 Implementations of *SCOPE*

SCOPE relies on the accuracy check, which compares the range of the calculated non-parametric CI with the $r\%$ error margin. Particularly, we consider non-parametric methods to calculate the required CIs in our approach design. This is because the performance of most serverless functions follows a non-normal distribution in both cold and warm starts [76]. Non-parametric methods are suitable for the performance analysis of serverless functions. Moreover, they work for the performance data with a normal distribution [57]. Note that in non-parametric analysis, the probability distribution of performance data is unknown, and fewer assumptions are required. Therefore, the metrics related to mean and standard deviation are rarely used because they are not robust when dealing with non-normal distribution [43]. In our approach, we choose percentile performance metrics because they do not depend on specific distribution assumptions. Calculating CIs for percentile performance also follows a non-parametric approach because it is a way to perform statistical analysis without assuming that the data follow any particular distribution. In Fig. 2, we support three mainstream non-parametric calculations of CIs for percentile performance (*Step 2* and *Step 2**). Thus, we obtain the following three implementations of *SCOPE* to provide the user with flexibility.

- ***SCOPE 1***: We use the general method [44, 57, 70] to calculate CIs for the percentile. The method involves sorting performance results in ascending order and then calculating two index values based on the data size, the given percentile, and the desired confidence level. Values at these two locations are the lower and upper bounds of the CI for the specified percentile and confidence level.
- ***SCOPE 2***: We use the basic bootstrapping method [5] based on resampling technique to calculate CIs. In this method, a set of n performance data is randomly sampled with replacement to construct a new set. This selection process is repeated n times to form one resample. The resampling process is then repeated c times (at least 1,000 times [35]) to generate c resample sets. For each resample, the performance at a specific percentile is calculated, and the resulting values are sorted. Finally, the lower and upper bounds of the CI are determined based on the sorted c values and a given confidence level $cl\%$.
- ***SCOPE 3***: We use the block bootstrapping method [4] to calculate CIs. Unlike the basic bootstrapping method, in the block bootstrapping method, the data selection of a round resample becomes the selection and combination of the block data with continuous performance results. We apply the automated selection of block size used in this work [35] to this method. Conducting c times of resamples will obtain the total of c values about the percentile performance. These values are still sorted, and the lower and upper bounds of the CI for this percentile are generated with a given confidence level $cl\%$.

3.5 An illustrating example of applying *SCOPE*

SCOPE ensures scalability without requiring additional manual efforts or modifications to the existing serverless platforms. It can be seamlessly integrated as an external service or auxiliary analysis tool to assess performance results. Serverless functions, including complex tasks, are triggered and executed on original serverless platforms. Developers who employ *SCOPE* capture and collect performance data from the serverless platform and input it into *SCOPE*. Then, *SCOPE* automatically analyzes performance data and provides insights into the need for additional repetition runs. This flexibility makes *SCOPE* highly scalable for various serverless functions.

To illustrate how *SCOPE* is used, we provide a real-world example of assessing the performance of a serverless function. The serverless function is from the dataset described in Section 4.3 of the experimental evaluation. We use *Func43* from the work [54] executed on AWS Lambda. First, the developer gives a set of performance results of a serverless function executed on its serverless platform (*Step 1*), e.g., 180 performance data points regarding the function *Func43*. Then, *SCOPE* calculates the corresponding CIs for the $25th$, $50th$, and $75th$ percentiles (*Step 2*). *Step 3* checks whether these CIs fall within the $r\%$ (e.g., 1%) error margin of respective observed true percentile performance, i.e., actual $25th$, $50th$, and $75th$ percentile performance in performance distribution with 180 data points. The result shows that these CIs satisfy the accuracy check. At the same time, *SCOPE* uses a set of performance results from *Step 1* to get the performance result set from previous run intervals, as outlined in *Step 1**. If the number of repetitions of the run interval is set to 5, the performance result set from previous run intervals is composed of the first 175 data points, excluding the last 5 data points. Similarly, *SCOPE* calculates and checks the accuracy of the newly calculated CIs for the $25th$, $50th$, and $75th$ percentiles (*Step 2* and Step 3**). The result shows that these CIs obtained from 175 data points satisfy the accuracy check. Both *Step 3* and *Step 3** satisfy the corresponding accuracy checks (*Step 4*), and then the performance result set given in *Step 1* is reported as the final performance of *Func43*.

4 EXPERIMENTAL EVALUATION

4.1 Research questions

We explore the following research questions:

RQ1: *How general effective is SCOPE for serverless functions compared to the state-of-the-art techniques?* This RQ aims to compare the general effectiveness of *SCOPE* and state-of-the-art performance testing techniques for cloud computing on serverless functions.

RQ2: *How well do SCOPE and state-of-the-art techniques apply to serverless functions under varying parameters?* The performance testing techniques aim to design the stopping criterion to determine whether the tested performance result set accurately reflects real performance and whether additional repeated runs from another run interval are necessary. The number of repetitions within the run interval represents the amount of new performance data added in each round of testing. Based on this, we investigate the effectiveness of *SCOPE* and state-of-the-art techniques under different constraints of the stopping criterion and different numbers of repetitions within the run interval.

RQ3: *How flexible and effective is SCOPE compared to the strategy of setting a fixed number of repetitions for all evaluated serverless functions?* As commonly used in previous studies, a fixed number of repetitions is applied for all serverless functions. This RQ aims to investigate the flexibility and effectiveness of *SCOPE* compared to the strategy of setting a fixed number of repetitions for all evaluated serverless functions.

4.2 Baselines

To answer RQ1 and RQ2, we select state-of-the-art performance testing techniques for comparison. As serverless computing is a cloud computing paradigm, a straightforward idea is to adopt existing performance testing techniques designed for cloud applications to serverless functions. Thus, we consider two state-of-the-art performance testing techniques designed for cloud applications: *PT4Cloud* [36] and *Metior* [35]. These approaches are non-parametric and have been evaluated to be superior to other techniques for cloud applications [35, 36], e.g., detecting repetitiveness of performance data and analyzing coefficient of variation of performance data. The cloud applications that they test are developed based on IaaS [23], a traditional cloud computing pattern that allows developers to lease resources and configure and manage the infrastructure. In addition, we also consider another non-parametric method, *CONFIRM* [57], for cloud environments. *CONFIRM* estimates the required number of repetitions for an experiment.

The stopping criteria of *PT4Cloud* [36] and *Metior* [35] rely on the stability assessment of performance distributions. *PT4Cloud* and *Metior* respectively compare the distribution similarity and changes in a performance metric. Particularly, *PT4Cloud* needs to set an objective probability p_0 (e.g., 90%) to represent accuracy requirements. *Metior* needs to specify a maximum allowed percentage error $e_0\%$ (e.g., 3%). The stopping criterion of *CONFIRM* [57] relies on the correctness assessment of CIs. It uses resampling without replacement and CIs for the median to determine whether the mean CIs fall within a desired error bound $e_0\%$ (e.g., 3%).

4.3 Dataset

To evaluate the effectiveness of performance testing techniques, we use a representative dataset that has recently been made publicly available for serverless function performance analysis [76]. This dataset comprises 65 serverless functions that have been meticulously sourced from peer-reviewed papers published in top-tier academic venues spanning the period from 2014 (the year that serverless computing started to be popular [40, 75]) to 2022. The size of our dataset (i.e., 65 serverless functions) is comparable to and even larger than those used in previous studies of serverless computing performance [49, 72, 78, 79, 87, 88].

The 65 serverless functions span a wide range of task types, ensuring a diverse representation of workloads. Specifically, it covers 25 distinct task categories, such as mathematical operations, image processing, face detection, graph network analysis, video processing, and natural language processing. This broad task variety provides a comprehensive view of real-world serverless computing workloads. Furthermore, the dataset covers widely-used benchmarks in the serverless computing research community [41, 55, 87] and the industry [3], e.g., *ServerlessBench* [15], *FunctionBench* [13], and *FaaSDom* [12]. In terms of serverless platforms, the dataset focuses on AWS Lambda and Google Cloud Functions, which are the most prevalent public serverless platforms to study [16, 29, 34]. For programming languages, the dataset includes functions written in Python and JavaScript, which are the dominant programming languages in serverless computing [1, 16, 29, 30].

4.4 Experimental setup

Execution configurations of serverless functions. We execute 65 serverless functions using the original function configurations and serverless platforms specified in the work [76]. If a configuration is not provided, we use the default configuration of the platform. At the time of our study, AWS Lambda uses a default memory size of 128 MB [8] and a timeout of 3 seconds [10], while Google Cloud Functions adopts a default memory size of 256 MB [9] and a timeout of 60 seconds [11]. If the configured memory size or timeout time is insufficient to support executions, we increase the value of these parameters and test the function again to ensure successful execution.

We repeatedly invoke the serverless function to produce a series of performance data, which are input to performance testing techniques to check when the repeated runs can be stopped. If the stopping criterion is not satisfied, we start the next run interval to generate more performance data. Since our goal is to evaluate the effectiveness of performance testing techniques in the context of serverless computing, by default, we set the number of repetitions of the run interval to five repetitions, an established number of repetitions commonly used in the experimental setting of serverless computing papers [22, 38, 61]. This small number allows us to obtain a fine-grained stop location and reduce the unnecessary overhead of running the function. In RQ2, we investigate the effect of the size of this number. We evaluate the effectiveness of performance testing techniques using cold-start and warm-start performance of the serverless functions. The cold-start performance is obtained by invoking the function after the resources generated by previous invocations have been released. The warm-start performance is obtained by invoking the function before releasing the resources from previous invocations. We use a half-hour invocation frequency for cold-start performance and a five-second frequency for warm-start performance, both after the previous invocation, as they can ensure the serverless function experiences cold and warm starts. For performance testing techniques, we use the same performance data of the functions to fairly compare their effectiveness.

Parameter configurations. In experiment evaluation, we will explore the supported three versions of *SCOPE* (i.e., *SCOPE 1*, *SCOPE 2*, and *SCOPE 3*) to demonstrate the flexibility and applicability of our approach across different non-parametric confidence interval calculations for percentile performance. For the comparison of evaluation results, we use the version of *SCOPE* (e.g., *SCOPE 1*) that is the most effective.

To answer RQ1, the confidence level $cl\%$ and the error margin $r\%$ in *SCOPE* use by default the values that have been widely adopted by previous work on performance analysis [57, 70]. $cl\%$ and $r\%$ are set to 95% and 1%, respectively. The resample times in *SCOPE 2* and *SCOPE 3* are by default set to 1,000, consistent with previous work [35]. For *PT4Cloud* and *Metior*, we use the default settings provided in their open-sourced code. For *PT4Cloud*, we use the default objective probability (p_0) of 90%, i.e., expecting the accuracy of testing results to be at least 90%. For *Metior*, we use the maximum allowed percentage error ($e_0\%$) of 3% for the median performance and the confidence level of 95%. For *CONFIRM*, we use the same error $e_0\%$ of 3% for the CIs for the median performance and the confidence level of 95%, as in *Metior*.

To answer RQ2, we adjust key parameters. (1) First, we adjust $r\%$ of *SCOPE* to 5%, 4%, 3%, 2%, and 1% to investigate the characteristics of the stopping criterion. We also adjust the key parameters for the stopping criterion of the state-of-the-art techniques: p_0 of *PT4Cloud* and $e_0\%$ of *Metior* and *CONFIRM*. We set p_0 to values of 90%, 92%, 94%, 96%, and 98%, where 92%, 94%, 96%, and 98% are stricter constraints that have not been used in previous evaluations of *PT4Cloud* [36]. We set $e_0\%$ to values of 5%, 4%, 3%, 2%, and 1%, where 2% and 1% are also stricter constraints not used in previous evaluations of *Metior* [35, 57]. (2) Second, we adjust the number of repetitions within the run interval to 3, 4, 5, 10, and 20, which are commonly used in serverless computing papers [20, 22, 32, 38, 61, 64, 71] for the total experimental repetitions. Serverless computing scenario generally uses small repetitions. The possible reason is that serverless functions can be triggered at any time, which makes it convenient to obtain any number of performance results. Thus, in the serverless computing scenario, developers tend to use a small run interval to invoke serverless functions on demand at any time without needing to lease resources in advance. Taking these values as the number of repetitions of one run interval, we could investigate if adding a cycle of performance data affects testing results and if the repetitions specified in previous work are sufficient to obtain accurate serverless function performance. We use default configurations for other parameters of *SCOPE*, *PT4Cloud*, *Metior*, and *CONFIRM*.

To answer RQ3, we evaluate the strategy of indiscriminately setting a fixed number of repetitions for all tested serverless functions, in the same way used in prior serverless computing studies. This strategy is different from our compared baselines. We evaluate different values from small to large, including 20 [69], 50 [62], 100 [42], 300 [54], and 500 [25]. They had been used in serverless computing papers.

Evaluation strategy and metrics. We use the performance data of the serverless function being repeatedly executed 1,000 times as the ground truth performance for identifying the effectiveness of testing results. To establish the ground truth, we require a relatively large number of performance tests that can capture all potential impacts of the serverless platform. We observe that 1,000 repetitions are the largest number found in existing serverless computing literature [46, 72, 76]. To confirm whether 1,000 repetitions are sufficient, we execute each serverless function for an additional 500 runs, and the results, discussed in Section 6.2, show consistent effectiveness compared to 1,500 repetitions. Therefore, 1,000 executions provide a trustworthy ground truth. We apply performance testing techniques to determine the termination of the repeated runs for the serverless function, i.e., the stop location. The performance data tested in the stop location is deemed to be the accurate performance distribution of this function acquired with performance testing techniques. We compare this distribution with the performance distribution of the ground truth of this function using the following evaluation metrics.

- *Accuracy:* He et al. [36] define the accuracy of performance testing results as the similarity between the performance distribution acquired with performance testing techniques and the corresponding ground truth distribution. The value of the similarity metric ranges from 0 to 100%, where 100% indicates the same distribution. We calculate the mean accuracy of the testing results obtained for all tested functions.
- *Reliability:* Previous work [70] defines the reliability of the obtained performance result as whether its specific percentile performance is accurate. When a percentile performance obtained from the performance distribution acquired with performance testing techniques falls within the 95% confidence interval for this percentile obtained from the corresponding ground truth distribution, it indicates a 95% probability that this percentile performance is reliable [70]. It also indicates that *the performance testing techniques enable the tested serverless function to get this reliable percentile performance.* This previous work [70] has focused on the performance at the 50*th* and 90*th* percentiles to assess the result reliability. To obtain more comprehensive results, we investigate the reliability of testing results from different performance perspectives by extending the percentiles to include the 25*th*, 50*th*, 75*th*, and 90*th* percentiles. We respectively calculate the percentage of the serverless functions with reliable performance at different percentiles.

Experimental environment. We implemented and ran *SCOPE*, *PT4Cloud*, *Metior*, *CONFIRM* and the invocation scripts for serverless functions on an Ubuntu 18.04.4 LTS server with an Intel Xeon (R) 4-core processor and 24GiB of memory.

5 RESULTS

5.1 RQ1: General Effectiveness of *SCOPE*

This section explores the general effectiveness of *SCOPE* compared to *PT4Cloud*, *Metior*, and *CONFIRM* when testing serverless function performance. Results show that *SCOPE* is highly effective. Table 1 shows the results of three variants of *SCOPE* and state-of-the-art techniques in cold-start and warm-start performance testing for serverless functions. On average, the mean accuracy obtained by *SCOPE 1*, *SCOPE 2*, and *SCOPE 3* is 97.25%, 95.07%, and 94.13%, respectively. *PT4Cloud*, *Metior*, and *CONFIRM* provide testing results with 52.28%, 63.42%, and 60.42% mean accuracy, respectively. Compared to *PT4Cloud*, *Metior*, and *CONFIRM*, *SCOPE* can improve the mean accuracy by 44.97, 33.83, and 36.83 percentage points, respectively. Moreover, *SCOPE* outperforms *PT4Cloud*, *Metior*, and *CONFIRM* on all serverless functions that we consider, thus indicating the feasibility and effectiveness of *SCOPE*.

For the reliability, *SCOPE 1* provides the reliable 25*th*, 50*th*, 75*th*, and 90*th* percentile performance for 87.69%, 93.08%, 92.31%, and 90.77% of the serverless functions, respectively. *SCOPE 2* provides

Table 1. RQ1: The results of performance testing for 65 serverless functions using *SCOPE*, *PT4Cloud*, *Metior*, and *CONFIRM*.

	Mean accuracy	***#Functions*: reliability at 25th/50th/75th/90th percentile**			
SCOPE 1 - cold start	97.39%	89.23%	93.85%	90.77%	92.31%
SCOPE 1 - warm start	97.12%	86.15%	92.31%	93.85%	89.23%
SCOPE 1 (Mean)	**97.25%**	**87.69%**	**93.08%**	**92.31%**	**90.77%**
SCOPE 2 - cold	95.94%	84.62%	86.15%	73.85%	73.85%
SCOPE 2 - warm	94.19%	75.38%	80.00%	75.38%	84.62%
SCOPE 2 (Mean)	**95.07%**	**80.00%**	**83.08%**	**74.62%**	**79.23%**
SCOPE 3 - cold start	95.05%	84.62%	78.46%	70.77%	67.69%
SCOPE 3 - warm start	93.22%	66.15%	78.46%	81.54%	83.08%
SCOPE 3 (Mean)	**94.13%**	**75.38%**	**78.46%**	**76.15%**	**75.38%**
PT4Cloud - cold	61.12%	12.31%	10.77%	18.46%	29.23%
PT4Cloud - warm	43.45%	16.92%	20.00%	18.46%	12.31%
PT4Cloud (Mean)	**52.28%**	**14.62%**	**15.38%**	**18.46%**	**20.77%**
Metior - cold start	69.38%	26.15%	21.54%	20.00%	27.69%
Metior - warm start	57.47%	24.62%	23.08%	23.08%	21.54%
Metior (Mean)	**63.42%**	**25.38%**	**22.31%**	**21.54%**	**24.62%**
CONFIRM - cold start	68.93%	24.62%	21.54%	23.08%	16.92%
CONFIRM - warm start	51.91%	13.85%	16.92%	29.23%	21.54%
CONFIRM (Mean)	**60.42%**	**19.23%**	**19.23%**	**26.15%**	**19.23%**

the reliable 25th, 50th, 75th, and 90th percentile performance for 80.00%, 83.08%, 74.62%, and 79.23% of the functions, respectively. *SCOPE 3* provides similar reliability to *SCOPE 2*. For *PT4Cloud*, on average, it enables 14.62%, 15.38%, 18.46%, and 20.77% of the functions to get the reliable 25th, 50th, 75th, and 90th percentile performance, respectively. Compared to *PT4Cloud*, *SCOPE* provides the reliable 25th, 50th, 75th, and 90th percentile performance for an additional 87.69% - 14.62% = 73.07%, 93.08% - 15.38% = 77.70%, 92.31% - 18.46% = 73.85%, and 90.77% - 20.77% = 70.00% of the functions, respectively. For *Metior*, it provides the reliable 25th, 50th, 75th, and 90th percentile performance for 25.38%, 22.31%, 21.54%, and 24.62% of the functions, respectively. Compared to *Metior*, *SCOPE* provides the reliable 25th, 50th, 75th, and 90th percentile performance for an additional 87.69% - 25.38% = 62.31%, 93.08% - 22.31% = 70.77%, 92.31% - 21.54% = 70.77%, and 90.77% - 24.62% = 66.15% of the functions, respectively. For *CONFIRM*, it provides the reliable 25th, 50th, 75th, and 90th percentile performance for 19.23%, 19.23%, 26.15%, and 19.23% of the functions, respectively. Compared to *CONFIRM*, *SCOPE* provides the reliable 25th, 50th, 75th, and 90th percentile performance for an additional 87.69% - 19.23% = 68.46%, 93.08% - 19.23% = 73.85%, 92.31% - 26.15% = 66.16%, and 90.77% - 19.23% = 71.54% of the functions, respectively. These results show that most of the testing results produced with *SCOPE* are accurate and reliable.

For three implementations of *SCOPE*, we summarize the following points. (1) *SCOPE 1* outperforms the other two, indicating that the CI calculation method of *SCOPE 1* has high accuracy on performance testing for serverless functions. This could be because the other two implementations use bootstrapping methods, which adopt the resampling strategy. The resampling process may break the original data distribution and introduce potential noises. In contrast, *SCOPE 1* is based on the original performance data to calculate CI. (2) *SCOPE 2* and *SCOPE 3* show comparable effectiveness. This indicates that our approach design is insensitive to internal data dependency and has a similar effectiveness for the same type of CI calculation method. (3) All implementations show comparable effectiveness in both cold and warm starts, indicating the stability of *SCOPE* in performance testing for serverless functions.

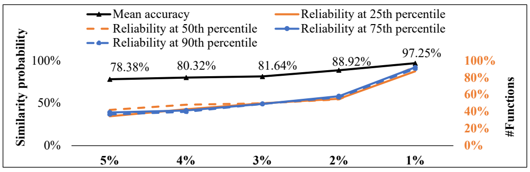

Fig. 3. RQ2: Changes in metric values under different constraints *r%* for ***SCOPE 1*** (mean results in cold and warm starts for tested functions).

Result discussion. We observe that state-of-the-art techniques (*PT4Cloud*, *Metior*, and *CONFIRM*) perform poorly in evaluating serverless functions. This ineffectiveness stems from fundamental differences in their approach: these techniques rely on stability or correctness assessments for specific performance. However, serverless functions, characterized by short durations and small run intervals, make it difficult for these methods to detect significant stability changes or undesired confidence intervals. The minor fluctuations exhibited in performance distributions cause these methods to reach their stopping criterion too early, undermining their ability to accurately assess the need for additional repetitions. This limitation highlights the need for a novel performance testing approach tailored to serverless functions that allows for finer-grained analysis.

However, the assessment strategy of *PT4Cloud*, *Metior*, and *CONFIRM* may be effective for traditional cloud applications or environments [35, 36, 57]. This may be because, traditional cloud applications or task executions have long-lived and minute-level duration, and previous work in cloud computing [23, 35, 36] adopted a period of time of runs to constantly invoke them for execution. This time is often several weeks or days, thus yielding a large number of performance results. Therefore, in state-of-the-art techniques, these characteristics of traditional cloud applications or environments lead to significant changes in stability or undesired confidence intervals on performance distributions, making them effective.

Serverless functions have distinctive performance features from traditional cloud applications or environments. Moreover, state-of-the-art techniques are not effective when applied to serverless function performance. In this paper, we present *SCOPE*, which introduces a novel stopping criterion, incorporating accuracy and consistency checks. These checks enable fine-grained analysis and high accuracy guarantees for performance distributions with these characteristics of serverless functions. This makes *SCOPE* more effective in the performance testing of serverless functions.

> **Ans. to RQ1:** *SCOPE* provides testing results with 97.25% accuracy, 44.97, 33.83, and 36.83 percentage points higher than *PT4Cloud*, *Metior*, and *CONFIRM*, respectively. Moreover, *SCOPE* outperforms all compared baselines on all serverless functions that we consider, thus indicating its feasibility and effectiveness.

5.2 RQ2: Effectiveness under varying parameters

In this section, we investigate the effectiveness of *SCOPE* and state-of-the-art techniques under varying parameters, including different constraints of the stopping criterion and different numbers of repetitions within the run interval.

We compare the effectiveness of *SCOPE* and state-of-the-art techniques under different constraints of the stopping criterion. For *SCOPE*, evaluation metric values have improvements as *r%* is limited from 5% to 1%. *PT4Cloud*, *Metior*, and *CONFIRM* have different sensitivities to the constraint of their stopping criteria. While the effectiveness of *PT4Cloud* does not significantly improve with

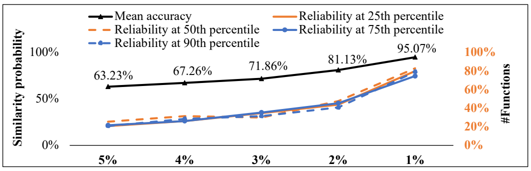

Fig. 4. RQ2: Changes in metric values under different constraints $r\%$ for ***SCOPE 2*** (mean results in cold and warm starts for tested functions).

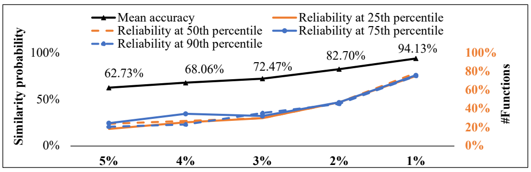

Fig. 5. RQ2: Changes in metric values under different constraints $r\%$ for ***SCOPE 3*** (mean results in cold and warm starts for tested functions).

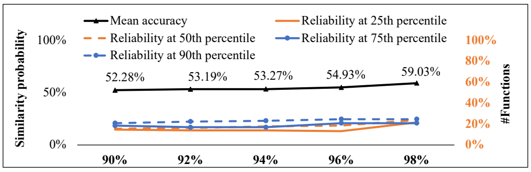

Fig. 6. RQ2: Changes in metric values under different p_0 for ***PT4Cloud***.

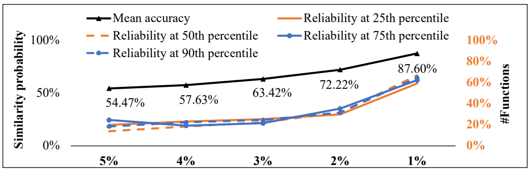

Fig. 7. RQ2: Changes in metric values under different $e_0\%$ for ***Metior***.

increasing constraints, *Metior* and *CONFIRM* show improvement in accuracy from 54.47% to 87.60% and from 55.33% to 82.41%, respectively. However, under the strictest constraint of *Metior* and *CONFIRM*, the obtained accuracy does not exceed 90%, which can be obtained by *SCOPE*. Overall, stopping criteria of state-of-the-art techniques may be insufficient for serverless computing.

Figs. 3, 4, and 5 show mean changes for three implementations of *SCOPE* under cold and warm starts. The mean accuracy improves from 78.38% to 97.25% for *SCOPE 1*, from 63.23% to 95.07% for *SCOPE 2*, and from 62.73% to 94.13% for *SCOPE 3*. For the reliability, with the use of *SCOPE 1*, the proportion of serverless functions that get the reliable $25th$, $50th$, $75th$, and $90th$ percentile performance increases from 34.62% to 87.69%, from 42.31% to 93.08%, from 39.23% to 92.31%, and 36.92% to 90.77%, respectively. *SCOPE 2* and *SCOPE 3* show the same trends for the reliability as *SCOPE 1*. These results show that the effectiveness of *SCOPE* is influenced by the constraints of the designed stopping criterion. Performance testing for serverless functions requires strict error constraints due to the short duration of the test. Using a strict error constraint enhances

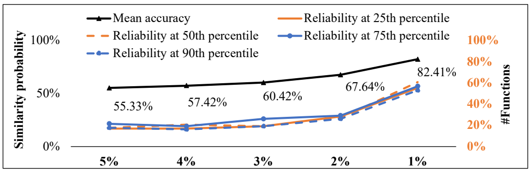

Fig. 8. RQ2: Changes in metric values under different $e_0\%$ for ***CONFIRM***.

accuracy, but also may increase runtime costs. *SCOPE* offers the advantage of achieving accurate performance without the need for excessive repetitions, thereby avoiding unnecessary runtime costs. In real-world scenarios, *SCOPE* provides developers with the flexibility to adjust the error constraint, allowing them to make desired trade-offs in line with specific requirements.

Fig. 6 illustrates the mean changes in metric values obtained by *PT4Cloud* when evaluating the cold-start and warm-start performance of the serverless functions under varying p_0. The mean accuracy changes from 52.28% to 59.03%. Although *PT4Cloud* makes more functions to obtain reliable percentile performance, the improvement is insignificant. Even if p_0 is constrained to 98%, only 21.54%, 23.85%, 20.77%, and 24.62% of the serverless functions can get the reliable $25th$, $50th$, $75th$, and $90th$ percentile performance, respectively. Thus, for *PT4Cloud*, we do not observe significant improvement in evaluation metrics as p_0 increases from 90% to 98%.

Fig. 7 shows the mean changes in metric values obtained by *Metior* under different $e_0\%$. The mean accuracy improves from 54.47% to 87.60% as $e_0\%$ becomes stricter, indicating that using a stricter $e_0\%$ can improve the effectiveness of *Metior*. However, even if $e_0\%$ is limited to 1%, the obtained mean accuracy (87.60%) does not exceed 90%. Moreover, the percentage of serverless functions with reliable performance is low. For example, for the $25th$ percentile performance, when $e_0\% = 1\%$, *Metior* enables 59.23% of the functions to get this reliable performance, while when $e_0\% = 5\%$, only 20.00% of the functions get this reliable performance.

Fig. 8 shows the mean changes in metric values obtained by *CONFIRM* under different $e_0\%$. The mean accuracy improves from 55.33% to 82.41% as $e_0\%$ becomes stricter, indicating that using a stricter $e_0\%$ can improve the effectiveness of *CONFIRM*. However, even if $e_0\%$ is limited to 1%, the obtained mean accuracy (82.41%) does not exceed 90%. Moreover, the percentage of serverless functions with reliable performance is low. For example, for the $90th$ percentile performance, when $e_0\% = 1\%$, *CONFIRM* enables 53.08% of the functions to get this reliable performance, while when $e_0\% = 5\%$, only 17.69% of the functions get this reliable performance.

We further investigate the effectiveness of *SCOPE*, *PT4Cloud*, *Metior*, and *CONFIRM* under varying numbers of repetitions of the run interval. For *SCOPE*, evaluation results do not show significant changes as the number of repetitions of the run interval increases from 3 to 20. *SCOPE* can obtain accuracy results that remain between 96.96% and 97.53%, showing a small variation of about 0.50%. For *PT4Cloud*, *Metior*, and *CONFIRM*, we observe a positive effect of the run interval on their effectiveness. However, the accuracy never reaches 80%.

Figs. 9, 10, and 11 show the mean results obtained by *SCOPE* under cold and warm starts. The mean accuracy obtained by *SCOPE 1*, *SCOPE 2*, and *SCOPE 3* ranges from 96.96% to 97.53%, from 93.56% to 96.72%, and from 93.70% to 95.82%, respectively. This indicates a negligible change in accuracy. The reliability at the percentile performance is also stable. As the number of repetitions of the run interval increases, *SCOPE 1* produces the reliable $25th$, $50th$, $75th$, and $90th$ percentile performance for the functions falling within the following ranges: 84.62% to 87.69%, 90.00% to 93.85%, 88.46% to 92.31%, and 85.38% to 91.54%. *SCOPE 2* produces the reliable $25th$, $50th$, $75th$, and $90th$ percentile performance for the functions falling within the following ranges: 78.46% to 83.08%,

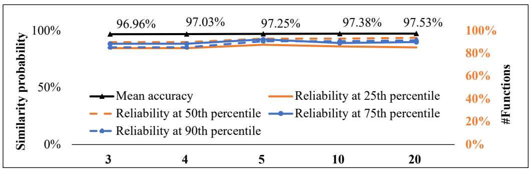

Fig. 9. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***SCOPE 1*** (mean results in cold and warm starts for tested functions).

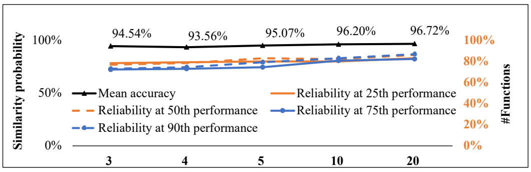

Fig. 10. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***SCOPE 2*** (mean results in cold and warm starts for tested functions).

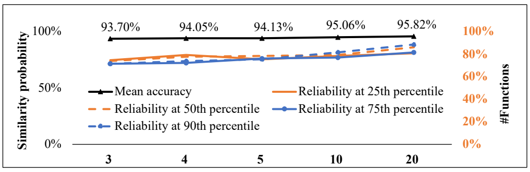

Fig. 11. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***SCOPE 3*** (mean results in cold and warm starts for tested functions).

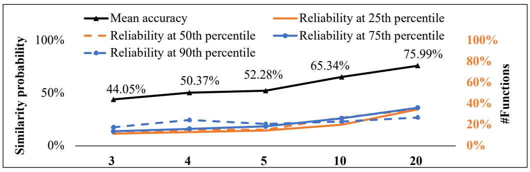

Fig. 12. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***PT4Cloud***.

76.92% to 86.15%, 72.31% to 82.31%, and 73.08% to 86.92%. The reliability results of *SCOPE 3* are highly similar to *SCOPE 2*. Overall, the effectiveness of *SCOPE* is not affected by the number of repetitions of the run interval.

Fig. 12 shows the mean changes obtained by *PT4Cloud* in cold-start and warm-start performance testing. The obtained mean accuracy increases from 44.05% to 75.99%, as the number of repetitions of the run interval increases from 3 to 20. This indicates a positive effect of the run interval on the effectiveness of *PT4Cloud*. However, while increasing the number of repetitions of the run interval enables more functions to get reliable performance, *PT4Cloud* still does not enable over 60% of the serverless functions to achieve it.

Fig. 13 shows the mean results obtained by *Metior* in cold-start and warm-start performance testing. As the number of repetitions of the run interval increases from 3 to 20, the obtained mean accuracy increases from 54.09% to 77.15%. This also indicates a positive effect of the run interval

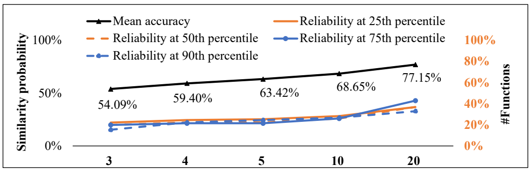

Fig. 13. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***Metior***.

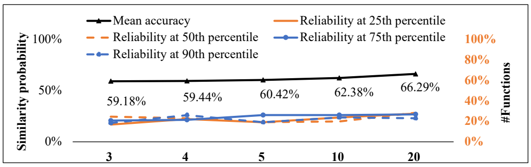

Fig. 14. RQ2: Changes in metric values under different numbers of repetitions of the run interval for ***CONFIRM***.

on the effectiveness of *Metior*. However, the accuracy of testing results is still no more than 80%, similar to the results obtained by *PT4Cloud*. In terms of reliability, for example, *Metior* enables only 36.92% of the serverless functions to get the reliable 50*th* percentile performance when the number of repetitions of the run interval is set to 20. As a result, 63.08% of the functions still cannot get this reliable performance.

Fig. 14 shows the mean results obtained by *CONFIRM*. As the number of repetitions of the run interval increases from 3 to 20, the obtained mean accuracy increases from 59.18% to 66.29%. This also indicates a positive effect of the run interval on the effectiveness of *CONFIRM*. However, the accuracy of testing results is still no more than 70%. In terms of reliability, for example, *CONFIRM* enables only 26.92% of the serverless functions to get the reliable 75*th* percentile performance when the number of repetitions of the run interval is set to 20. As a result, 73.08% of the functions still cannot get this reliable performance.

The slight increase in the metric values for *PT4Cloud*, *Metior*, and *CONFIRM* may also be due to the fact that when the number of repetitions in the run interval increases, the number of data points in the performance distribution for the initial comparison also increases. Overall, the effectiveness of baselines are sensitive to the number of repetitions of the run interval, with improved accuracy and reliability as the number increases. However, even if the number of repetitions of the run interval is set to 20, the maximum accuracy obtained by *PT4Cloud*, *Metior*, and *CONFIRM* is 75.99%, 77.15%, and 66.29%, respectively. These results cannot reach 80%. Moreover, over 60% of the functions still cannot get reliable percentile performance.

> **Ans. to RQ2:** Using a strict error constraint as the stopping criterion can improve the accuracy of *SCOPE*. Even when *PT4Cloud*, *Metior*, and *CONFIRM* apply the strictest constraints, the results obtained are inferior to those of *SCOPE*. *SCOPE* is insensitive to the number of repetitions of the run interval. As this number increases from 3 to 20, *SCOPE* can provide testing results with accuracy ranging from 96.96% to 97.53%, exhibiting a trivial difference of about 0.50%. *PT4Cloud*, *Metior*, and *CONFIRM* exhibit sensitivity to the number of repetitions of the run interval. However, even with a large number, the maximum accuracy obtained by *PT4Cloud*, *Metior*, and *CONFIRM* cannot reach 80%. Moreover, over 60% of the tested serverless functions cannot get reliable percentile performance.

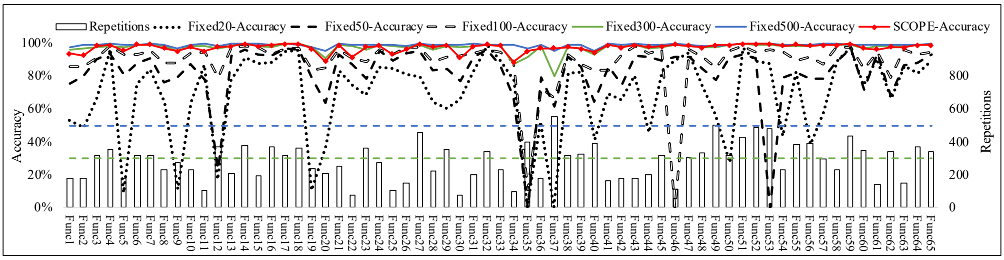

Fig. 15. The comparison of the strategy of blindly setting a uniform number of repetitions and *SCOPE* in each tested serverless function (cold starts).

5.3 RQ3: Flexibility of *SCOPE*

We compare *SCOPE* with the strategy of indiscriminately setting a fixed number of repetitions for all evaluated serverless functions. The results show that *SCOPE* is more flexible and effective than this strategy in determining the accurate performance and providing repetitions across various serverless functions.

Fig. 15 shows their comparison results in each tested serverless function under cold starts. Multiple polylines represent the accuracy results obtained on each serverless function when using the strategy of indiscriminately setting a fixed number of repetitions and *SCOPE*. Specifically, when the fixed number of repetitions is blindly set to 20, 50, and 100, the obtained accuracy for more than 95% of serverless functions is not as high as the accuracy obtained by *SCOPE* for these serverless functions. When this fixed number is set to 300, the accuracy results (green polyline) are comparable to those of *SCOPE*. When this fixed number is set to 500, the accuracy results (blue polyline) may be higher than those of *SCOPE* for most serverless functions.

However, the requirement for repetitions to achieve accurate performance can vary significantly among serverless functions. The bars in the figure represent the stop location of each serverless function determined by *SCOPE*. Some functions may achieve accurate performance with fewer than 100, while others might need over 300 repetitions. It indicates that each serverless function requires a customized repetition to obtain accurate performance. It is impractical to rely on a fixed number of repetitions (e.g., 300 or 500) for all evaluated functions. For example, serverless functions that do not require 300 repetitions may require fewer repetitions to obtain accurate performance, then incurring extra running overhead (e.g., 225 additional runs for *Func22*). Some functions may require more than 300 repetitions, which makes the results less accurate (e.g., 79.96% accuracy at 300 repetitions for *Func37*). For 500-fixed repetitions, although the accuracy results may be higher than those of *SCOPE* on many functions, most serverless functions do not require 500 repetitions to obtain accurate performance, thus incurring significant running overhead. We measure the overhead by evaluating the total number of repetitions used, which directly correlates with overhead. In Fig. 15, the strategy of 500 fixed repetitions necessitates 500*65 = 32,500 repetitions. *SCOPE* requires 18,345 repetitions. Thus, the 500-fixed repetition strategy uses 77.16% more repetitions of the running overhead than *SCOPE*, which makes *SCOPE* become a cost-efficient option.

> **Ans. to RQ3:** Contrasted with the strategy of indiscriminately setting a fixed number of repetitions, *SCOPE* is more flexible and effective in determining specific repetitions and obtaining accurate performance across various serverless functions.

Table 2. The results of other situations using *SCOPE*.

	Accuracy	**Reliability at 25th/50th/75th/90th percentile**			
Func4 - a mix of cold and warm starts	99.47%	✓	✓	✓	✓
Func10 - a mix of cold and warm starts	95.85%	✓	✓	✓	✓
App1 from AWS - cold start	99.43%	✓	✓	✓	✓
App1 from AWS - warm start	98.82%	✓	✓	✓	✓
App2 from Google - cold start	98.55%	✓	✓	✓	×
App2 from Google - warm start	99.77%	✓	✓	✓	✓
Func6 - multiple inputs	96.37%	✓	✓	✓	✓
Func7 - multiple inputs	97.02%	✓	✓	✓	✓
Func5 - file upload	98.98%	✓	✓	✓	✓
Func34 - message queue	96.04%	✓	✓	✓	×
Func61 - database insert	97.63%	✓	✓	✓	✓
Func32 - bursty workloads	99.63%	✓	✓	✓	✓
Func52 - bursty workloads	99.55%	✓	✓	✓	✓
NewFunc1 - Azure - cold start	93.61%	✓	✓	✓	✓
NewFunc2 - Azure - cold start	99.68%	✓	✓	✓	✓
NewFunc3 - Alibaba - cold start	97.32%	✓	✓	✓	×
NewFunc4 - Alibaba - cold start	96.68%	✓	✓	×	✓
Cold start (Mean)	96.82%	4/4	4/4	3/4	3/4
NewFunc1 - Azure - warm start	93.48%	✓	✓	✓	✓
NewFunc2 - Azure - warm start	97.01%	✓	✓	✓	✓
NewFunc3 - Alibaba - warm start	95.31%	✓	✓	✓	✓
NewFunc4 - Alibaba - warm start	97.09%	✓	✓	✓	✓
Warm start (Mean)	95.72%	4/4	4/4	4/4	4/4

5.4 Effectiveness to other situations

In this section, we explore the effectiveness of *SCOPE* in other situations, including mixed cold and warm conditions, serverless applications composed of multiple functions, varying input conditions, functions with other types of triggers, highly bursty workloads, and functions executed across different platforms. We also explore the effectiveness of *SCOPE* when incorporating additional critical performance metrics, such as tail latency and outlier behavior.

We explore the effectiveness of *SCOPE* on performance data from serverless functions under mixed cold and warm start conditions. For this purpose, we select two serverless functions: *Func4* and *Func10*. Each function is invoked, with the next invocation occurring after a randomly determined time, producing a mix of cold and warm starts, as typically observed in production applications. To establish the ground truth, we collect performance data for each function under 1,000 random executions, thereby capturing a realistic blend of cloud and warm start behavior. Table 2 shows the results. Specifically, *SCOPE* achieves an accuracy of 99.47% for *Func4* and 95.85% for *Func10*, showing high accuracy. Further, *SCOPE* provides reliable 25th, 50th, 75th, and 90th percentile performance for these functions. These results demonstrate the effectiveness of our approach in handling mixed cold and warm start performance data for serverless functions.

We explore the effectiveness of *SCOPE* in analyzing serverless applications composed of multiple functions that interact with each other. For this purpose, we implement two serverless applications based on examples provided in the platforms' official documentation. The first serverless application, from the AWS platform [7], consists of five interacting functions, while the second application, from the Google platform [18], is composed of three functions. These applications are numbered as *App1* and *App2*. We analyze their cold-start and warm-start response latencies using *SCOPE*. To evaluate the effectiveness of *SCOPE*, we collect cold-start and warm-start performance data for each application under 1,000 executions, which serves as the ground truth. Table 2 shows their results. For cold starts, *SCOPE* achieves an accuracy of 99.43% for *App1* and 98.55% for *App2*. Furthermore, *SCOPE* reliably estimates performance at 25*th*, 50*th*, and 75*th* percentile performance for both applications. For warm starts, *SCOPE* achieves an accuracy of 98.82% for *App1* and 99.77% for *App2*, demonstrating reliable performance at the 25*th*, 50*th*, 75*th*, and 90*th* percentile percentiles. This illustrates that *SCOPE* can achieve similar effectiveness for serverless application performance.

We explore the effectiveness of *SCOPE* under varying input conditions for serverless functions. For this purpose, we select two serverless functions: *Func6* and *Func7*, which allow for flexible adjustment of input to alter the computation scale. We generate three distinct inputs for each function to ensure a broader evaluation, rather than relying on a single input. Although three inputs provide a reasonable balance between diversity and experimental feasibility, our approach is not limited to this number and can be extended to more inputs if needed. For example, *Func7* solves linear equations with input n representing the matrix size. We generate three sizes, e.g., 15, 18, and 20. During each warm-start invocation, one of these inputs is randomly selected to produce the corresponding performance data. To evaluate *SCOPE*, we collect performance data for each function over 1,000 executions with these varying inputs, which serves as the ground truth. Table 2 shows their results. *SCOPE* can achieve an accuracy of 96.37% for *Func6* and 97.02% for *Func7*. Moreover, *SCOPE* reliably estimates performance at the 25*th*, 50*th*, 75*th*, and 90*th* percentiles for both functions, demonstrating its effectiveness in handling diverse input conditions.

We explore the effectiveness of *SCOPE* under serverless functions with other types of triggers. For this purpose, we select three functions (*Func5*, *Func34*, and *Func61*) from our set of 65 serverless functions. Then, we modify them to use file uploads, message queues, and database inserts as their triggers by incorporating Amazon S3, Amazon SQS, and Amazon DynamoDB, respectively. We analyze their warm-start response latencies using *SCOPE*. To obtain the end-to-end response time, we calculate the time difference between the initiation of the triggering event and the completion of the function execution. This measurement is consistently performed by recording the event's start time and the function's end time within the function code and then logging the computed time difference. The resulting time difference, referred to as the end-to-end response time, includes both the cloud infrastructure overhead and the function execution time. We collect performance data for each function across 1,000 executions, which serves as the ground truth. The results are presented in Table 2. *SCOPE* achieves an accuracy ranging from 96.04% to 98.98%. Moreover, *SCOPE* reliably estimates the performance at 25*th*, 50*th*, 75*th*, and 90*th* percentiles for *Func5* and *Func61*. Furthermore, *SCOPE* provides reliable 25*th*, 50*th*, and 75*th* percentile performance for *Func34*. These results highlight the effectiveness of *SCOPE* in assessing the performance of serverless functions triggered by various event types.

We investigate the effectiveness of *SCOPE* under highly bursty workloads for serverless functions. To this end, we select two serverless functions: *Func32* from AWS Lambda and *Func52* from Google Cloud Functions. These functions are subjected to a series of random burst invocations after prolonged periods of inactivity. The bursty invocations consist of randomly generated concurrent requests. We respectively collect 1,000 performance data triggered by these bursty invocations for both *Func32* and *Func52*, which serve as the ground truth for evaluating the accuracy and reliability of the results produced by *SCOPE*. As shown in Table 2, *SCOPE* achieves an accuracy of 99.63% for *Func32* and 99.55% for *Func52*. Additionally, *SCOPE* provides reliable performance across the 25*th*, 50*th*, 75*th*, and 90*th* percentiles for both functions. This illustrates that *SCOPE* effectively handles serverless function performance with highly bursty workloads.

We investigate the effectiveness of *SCOPE* on performance data from serverless functions executed on different platforms. Specifically, we conduct experiments on serverless functions deployed

Table 3. The results of *SCOPE* considering other performance metrics.

	Accuracy	***#Functions*: reliability at 25*th*/50*th*/75*th*/90*th* percentile**			
SCOPE	97.25%	87.69%	93.08%	92.31%	90.77%
SCOPE + Tail	98.84%	91.54%	96.92%	94.62%	93.08%
SCOPE + Outlier	97.61%	88.46%	93.08%	92.31%	90.00%

on Microsoft Azure Functions and Alibaba Function Compute. Four serverless functions are implemented: two from Microsoft Azure Functions (denoted as *NewFunc1* and *NewFunc2*) and two from Alibaba Function Compute (denoted as *NewFunc3* and *NewFunc4*). We analyze their cold-start and warm-start response latencies using *SCOPE*. To evaluate the effectiveness of *SCOPE*, we collect cold-start and warm-start performance data for each function over 1,000 executions, which serve as the ground truth for comparison. The results are presented in Table 2. For cold starts, *SCOPE* achieves a mean accuracy of 96.82% across these functions. Additionally, *SCOPE* reliably captures performance at the 25*th*, 50*th*, 75*th*, and 90*th* percentiles for four, four, three, and three functions, respectively. For warm starts, *SCOPE* achieves a mean accuracy of 95.72%, with reliable performance at the 25*th*, 50*th*, 75*th*, and 90*th* percentiles for all four functions, respectively. These results demonstrate the effectiveness and generalizability of *SCOPE* in assessing the performance of serverless functions executed on different serverless platforms.

We explore the effectiveness of *SCOPE* when incorporating additional critical performance metrics, such as tail latency and outlier behavior. Specifically, we consider the tail latency of the 95*th* percentile performance in *SCOPE* and check whether its confidence interval falls within a defined error margin (e.g., 3%) of the observed true percentile performance. Since enforcing constraints on tail latency is even more difficult to achieve, the error threshold may need to be relaxed compared to the original constraint settings for non-tail latency. This is because tail latency is inherently more variable and sensitive, making strict enforcement more challenging in practice. Additionally, we investigate the inclusion of outlier behavior in *SCOPE*. Outliers are determined using the Interquartile Range (IQR) method, a statistical approach for detecting outliers. In this method, outliers are identified as data points that fall outside the range defined by the first and third quartiles, typically beyond 1.5 times the IQR. If the number of outliers does not exceed a predefined threshold (e.g., 10% of the total test data), the outlier behavior is considered negligible in our study. Table 3 shows the mean results of two new versions of *SCOPE* in cold-start and warm-start performance testing for serverless functions. The mean accuracy achieved by *SCOPE* considering tail latency is 98.84%, and 97.61% for outlier behavior. Regarding reliability, *SCOPE* with tail latency provides reliable 25*th*, 50*th*, 75*th*, and 90*th* percentile performance for 91.54%, 96.92%, 94.62%, and 93.08% of the serverless functions, respectively. Similarly, *SCOPE* with outlier behavior provides reliable 25*th*, 50*th*, 75*th*, and 90*th* percentile performance for 88.46%, 93.08%, 92.31%, and 90.00% of the functions, respectively. From these results, adding other critical performance metrics, such as tail latency or outlier behavior, can improve the effectiveness of *SCOPE*. It is important to note that even without the integration of these critical metrics, our approach maintains consistency and high effectiveness. This suggests that while adding tail latency and outlier behavior improves the robustness and comprehensiveness of the analysis, *SCOPE* already performs well across different serverless functions.

6 DISCUSSION

6.1 Why *SCOPE*?

Similar to *PT4Cloud*, *Metior*, and *CONFIRM*, *SCOPE* efficiently determines the number of repetitions for each serverless function, ensuring accurate and reliable performance testing outcomes.

Notably, *SCOPE* achieves an impressive accuracy rate of 97.25%, outperforming *PT4Cloud*, *Metior*, and *CONFIRM* by 44.97, 33.83, and 36.83 percentage points, respectively. The ineffectiveness of baselines stems from fundamental differences in approach, which fail to accommodate the unique characteristics of serverless functions.

The question then arises: *does SCOPE's success solely depend on mandating a higher number of repetitions?* The answer is no. While heightened accuracy in performance testing inherently demands increased repetitions, *SCOPE* refines this approach by ensuring the effective determination of additional repetitions. This is evidenced by our analysis in RQ3. Specifically, in RQ3, we adopt a fixed number of 300 repetitions per serverless function, which maintains a comparable total volume of repetitions to *SCOPE* across all functions considered and achieves a similar overall accuracy. However, this comparison exposes the inefficiencies inherent in employing a fixed number. Employing a rigidly high fixed repetition number for serverless functions that do not require 300 repetitions produces unnecessary resource usage and prolonged testing times, without a commensurate increase in result precision. Conversely, this fixed approach proves inadequate for functions that may require more repetitions, risking insufficient coverage and potentially failing to guarantee the accuracy and reliability of serverless function performance.

Therefore, while high-accuracy performance testing inherently demands a greater number of repetitions, *SCOPE* surpasses this requirement by adopting a strategy that does not just increase repetitions but does so with discerning efficiency. This strategy ensures *SCOPE*'s efficacy across a diverse array of serverless functions, validating its effectiveness. Thus, *SCOPE*'s superior effectiveness is not merely a product of increased testing iterations but results from a strategic and judicious application of those additional repetitions, tailored to the unique demands of each serverless function it evaluates.

6.2 Validity of ground truth performance

We use 1,000 repetitions as the ground truth performance for each serverless function to identify the technique's effectiveness. This may raise the question of whether a sufficiently large number of repetitions are chosen for ground truth performance. To address it, we execute each serverless function for an additional 500 runs. When using execution results of 1,000, 1,100, 1,200, 1,300, 1,400, and 1,500 repetitions of the ground truth, the mean accuracy of *SCOPE* is between 96.18% and 97.25%, a difference of about one percentage point. The consistent results indicate that using 1,000 executions has established a trustworthy ground truth. An additional 500 runs are added to our online repository [17].

6.3 Discussion of different design, data, and parameter choices

SCOPE is built with a flexible, effective, and adaptable design aimed at providing accurate performance analysis across a wide range of serverless functions. Central to the design is the stopping criterion, which evaluates the tested performance data of serverless functions. This criterion is based on non-parametric confidence interval (CI) calculations, and we support three mainstream CI calculation methods to demonstrate the flexibility, applicability, and effectiveness of our approach. Furthermore, as discussed in Section 5.4, we explore the inclusion of additional critical performance metrics, such as tail latency and outlier behavior, into the design of *SCOPE*. The results demonstrate that while incorporating these metrics improves the results of the analysis, *SCOPE* performs well across different serverless functions, achieving an accuracy of 97.25%. This indicates that even without these additional metrics, *SCOPE* remains highly effective for performance testing.

SCOPE is evaluated using a diverse dataset that includes 25 distinct task types. These tasks cover a broad array of applications, including mathematical operations, image processing, face detection, graph network analysis, video processing, and natural language processing. This diverse set of functions ensures that our evaluation reflects a wide range of real-world serverless computing workloads, enabling us to assess the generalizability of *SCOPE* across different domains and use cases, both in cold-start and warm-start scenarios. The additional situations explored in Section 5.4 further confirm its applicability across various situations, such as mixed cold and warm conditions, serverless applications composed of multiple functions, varying input conditions, functions with other types of triggers, highly bursty workloads, and functions executed across different platforms. Additionally, we use a trustworthy ground truth to validate the performance of *SCOPE*. To assess the sufficiency of 1,000 repetitions as the ground truth, we conduct an additional 500 runs for each function, as discussed in Section 6.2. The results show that the effectiveness of *SCOPE* remains consistent when compared to the 1,500 repetitions.

SCOPE's evaluation also includes a thorough investigation of parameter choices. In RQ2, we examine the impact of varying key parameters, such as different constraints for the stopping criterion and varying the number of repetitions within each run interval. The results demonstrate that *SCOPE* continues to maintain high effectiveness regardless of these parameter variations. Additionally, we assess *SCOPE* by comparing how well the estimated performance aligns with the true performance across different percentiles ($25th$, $50th$, $75th$, $90th$). This analysis highlights the ability of *SCOPE*.

In summary, the design, data, and parameter choices of *SCOPE* are carefully tailored to provide an effective performance testing approach across a variety of serverless functions.

6.4 Discussion of implications

Our work provides several key implications for the design and implementation of performance testing approaches, spanning the pre-design, design, and post-design phases. These implications are outlined as follows: (1) When designing a performance testing approach, it is essential to assess the magnitude of the performance data in order to determine the most appropriate level of granularity for analysis. If the magnitude is large, a coarse-grained analysis might be more efficient, as it provides a broader overview with less computational overhead. Conversely, for smaller magnitude, a fine-grained analysis may be necessary to capture subtle variations in performance. This consideration helps to target the accuracy in the performance evaluation process. (2) Another critical consideration is the distribution of the performance data. Different analysis methods are suitable depending on whether the data follows a normal or non-normal distribution. In cases where performance data exhibits a non-normal distribution, non-parametric methods are the most appropriate choice. These methods do not assume any specific data distribution, making them more robust for handling irregular data patterns. This underscores the need for careful method selection to ensure reliable and valid performance analysis. (3) Performance characteristics can vary significantly depending on the task type being executed. Therefore, any performance testing approach must incorporate a diverse set of task types to ensure a comprehensive evaluation. By testing across a wide range of tasks, one can ensure that the approach remains effective in a variety of real-world tasks. This variability in performance highlights the importance of adapting the performance testing approach to different contexts and workloads to ensure generalizability.

6.5 Threats to validity

Selection of baseline methods. Embracing serverless computing enables developers to create cloud applications in a new programming paradigm. However, the literature on serverless computing lacks dedicated performance testing approaches. As a result, baselines specifically designed for serverless performance testing are not currently available. To address this, we evaluate serverless function performance using performance testing methods designed for traditional cloud applications. This may raise a threat to the representativeness of baseline methods. To mitigate this threat, we select three state-of-the-art performance testing methods: *PT4Cloud*, *Metior*, and *CONFIRM*. They have demonstrated superiority over previous performance testing techniques for cloud applications [35, 36]. These methods are detailed in Section 4.2 and Section 7. Therefore, our selected baseline methods are representative of the state-of-the-art.

Conclusion of technical effectiveness. We evaluate the effectiveness of *SCOPE* and state-of-the-art techniques on the performance results of serverless functions. Technical effectiveness may vary depending on the types of serverless functions. This may potentially influence the experimental conclusions that we summarize herein. To mitigate it, we investigate the performance of 65 serverless functions from a publicly available dataset curated in previous work [76], which covers a variety of tasks, e.g., video processing, machine learning, and natural language processing. Thus, the conclusion of technical effectiveness is based on testing results for various types of serverless functions. Although we cannot generalize our results to all serverless functions, the ones used herein are representative of those widely used in previous work.

6.6 Limitations of *SCOPE*

While *SCOPE* provides accurate and effective performance evaluations, it currently does not delve into specific distribution characteristics, such as identifying patterns, which could further enhance the depth of performance assessment. *SCOPE* employs a non-parametric approach for evaluating serverless function performance. This method is advantageous as it makes minimal assumptions about the underlying probability distribution of performance data, making it well-suited for scenarios where the distribution is unknown or highly variable. However, if the performance distribution of serverless functions is explicitly determined and incorporated into the analysis, it could enable *SCOPE* to provide even more nuanced insights and potentially improve its evaluation effectiveness.

6.7 Challenges of *SCOPE*

In the process of designing *SCOPE*, we face several key challenges related to selecting appropriate checks, statistical methods, and performance metrics for analysis. First, in serverless computing, it is crucial to determine the appropriate level of granularity for performance analysis. Due to the latency characteristics of serverless functions, a fine-grained approach is necessary to capture subtle changes in performance data and ensure effective analysis. Second, serverless functions generally exhibit irregular performance patterns. This variability necessitates careful consideration of which statistical methods are most appropriate. Since the performance data of many serverless functions, both in cold and warm starts, follows a non-normal distribution, non-parametric methods are more suitable for accurate performance analysis because they require fewer distribution assumptions. Finally, once the appropriate statistical methods are identified, we must determine which performance metrics should be analyzed. A critical aspect of our approach is assessing whether the performance metric falls within a desired confidence interval. Given the fine-grained nature of our analysis, we extend our checks to cover a broader range of the performance distribution—specifically from the $25th$ percentile to the $75th$ percentile. This ensures that results are reliable and accurate.

6.8 Future work

In future work, we aim to extend the capabilities of *SCOPE* by testing a wider variety of serverless functions and applications across different serverless providers. This will involve conducting performance evaluations in diverse real-world scenarios, such as hybrid cold-warm start environments, variations in input types, and different event triggers. Additionally, we plan to integrate *SCOPE* with various monitoring and profiling tools to gain deeper insights into the runtime behavior of serverless functions, enhancing the diagnostic and optimization capabilities of our approach.

Furthermore, we intend to develop *SCOPE* into a serverless API, making it easily accessible for developers and researchers to perform performance evaluations of serverless functions and applications. The API will provide customizable testing parameters, allowing for flexible performance analysis in various contexts. By providing seamless access to *SCOPE*, we hope to promote its wider adoption and support the continuous evolution of performance testing techniques within the serverless computing domain.

7 RELATED WORK

Performance of serverless computing. Performance is the most studied topic in the serverless computing literature [45, 66, 74]. On one hand, researchers have proposed novel solutions for optimizing the performance of serverless functions [39, 47–49, 53, 58, 62, 65, 68]. For example, *FaaSLight* [49] loaded only indispensable code for serverless functions to improve overall performance. *SOCK* [62] cached commonly used libraries to speed up the cold-start performance of serverless functions. A management layer called *SONIC* [53] was designed to improve the communication performance between serverless functions. On the other hand, empirical studies have delved into characterizing the performance of serverless computing. For instance, Eismann *et al.* [28] utilized a case study on AWS Lambda to investigate the stability of performance measurements by exploring the impact of various configuration settings. McGrath*et al.* [58] designed a performance-oriented serverless platform and evaluated the performance characteristics of serverless platforms. Wen *et al.* [79] conducted a thorough measurement study to characterize the performance of serverless functions executed on different commodity serverless platforms. However, there is no performance testing approach specifically tailored to serverless computing to help researchers and engineers determine accurate and reliable performance for serverless functions. To fill the gap, we propose *SCOPE*.

Performance testing of cloud applications. Researchers have proposed the related performance testing work for traditional cloud applications [35, 36, 43, 51, 57, 73]. Laaber *et al.* [43] explored the impact of cloud environments on the variability of performance test results and assessed the reliability of detecting slowdowns. Eismann *et al.* [26] primarily conducted a measurement study of microservices' performance and discussed related challenges without proposing a specific performance testing approach to determine the accurate performance. Wang *et al.* [73] proposed a non-parametric approach for cloud performance testing, based on basic bootstrapping techniques. Similarly, Maricq *et al.* [57] introduced CONFIRM, a non-parametric method that employed bootstrapping for cloud performance testing. However, He *et al.* [35] demonstrated that the bootstrapping method lacks the consideration of internal data dependency, causing a low accuracy in cloud application performance. To address this limitation, He *et al.* [35] improved this kind of approach by incorporating the block bootstrapping method, which accounted for the internal data dependency.

For the stopping criterion of performance testing, Alghmadi *et al.* [21] measured the degree of repetition of data in performance results. However, previous work [36] showed that this stopping criterion was not appropriate for performance testing of cloud applications. *PT4Cloud* [36] and *Metior* [35] have considered the stability of performance distributions to terminate the repeated runs, and have been evaluated to be the state-of-the-art in performance testing for cloud applications [26, 35]. However, our evaluation uncovers that *PT4Cloud* and *Metior* show low effectiveness in serverless computing. This paper proposes *SCOPE* with a novel stopping criterion, which considers accuracy and consistency checks and outperforms *PT4Cloud* and *Metior* in the performance testing for serverless functions.

8 CONCLUSION

This paper explores performance testing in the serverless computing domain. We proposed *SCOPE*, the first performance testing approach specifically tailored for serverless computing, which included accuracy and consistency checks. *SCOPE* emphasized the need for accuracy for most performance of serverless functions to determine accurate and reliable performance. We investigated 65 serverless functions and used their performance results to evaluate the effectiveness of *SCOPE* and state-of-the-art techniques. The results showed that *SCOPE* provided testing results with 97.25% accuracy, 33.83 percentage points higher than the best currently available technique.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China under Grant Nos. 62325201, 62425203, and 62032003, Beijing Natural Science Foundation under Grant No. 4244076, and Beijing Postdoctoral Research Foundation under Grant No. 2024-ZZ-20.

REFERENCES

[1] 2024. 2018 serverless community survey: huge growth in serverless usage. https://www.serverless.com/blog/2018-serverless-community-survey-huge-growth-usage.

[2] 2024. AWS Lambda. https://docs.aws.amazon.com/lambda.

[3] 2024. AWS serverless application repository. https://serverlessrepo.aws.amazon.com/applications.

[4] 2024. Block Bootstrapping. https://medium.com/@jcatankard_76170/block-bootstrapping-with-time-series-and-spatial-data-bd7d7830681e.

[5] 2024. Bootstrapping (statistics). https://en.wikipedia.org/wiki/Bootstrapping_(statistics).

[6] 2024. Comparison of cold starts in serverless functions across AWS, Azure, and GCP. https://mikhail.io/serverless/coldstarts/big3/.

[7] 2024. Create a Serverless workflow with AWS Step Functions and AWS Lambda. https://aws.amazon.com/tutorials/create-a-serverless-workflow-step-functions-lambda.

[8] 2024. Default memory size of AWS Lambda. https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html.

[9] 2024. Default memory size of Google Cloud Functions. https://cloud.google.com/functions/docs/configuring/memory.

[10] 2024. Default timeout of AWS Lambda. https://docs.aws.amazon.com/lambda/latest/dg/configuration-function-common.html.

[11] 2024. Default timeout of Google Cloud Functions. https://cloud.google.com/functions/docs/configuring/timeout.

[12] 2024. FaaSDom. https://github.com/faas-benchmarking/faasdom.

[13] 2024. FunctionBench. https://github.com/ddps-lab/serverless-faas-workbench.

[14] 2024. Google Cloud Functions. https://cloud.google.com/functions.

[15] 2024. ServerlessBench. https://github.com/SJTU-IPADS/ServerlessBench.

[16] 2024. The state of serverless. https://www.datadoghq.com/state-of-serverless/.

[17] 2024. Supplemental material. https://github.com/WenJinfeng/SCOPE_PerformanceTesting.

[18] 2024. Use Workflows with Cloud Functions tutorial. https://cloud.google.com/workflows/docs/tutorials/run/cloud-run.

[19] 2025. A research and markets report. https://omdia.tech.informa.com/pr/2024/jun/omdia-serverless-computing-valued-at-19-billion-dollars-is-the-fastest-growing-cloud-service.

[20] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards high-performance serverless computing. In *Proceedings of the 2018 USENIX Annual Technical Conference*. 923–935.

[21] Hammam M Alghmadi, Mark D Syer, Weiyi Shang, and Ahmed E Hassan. 2016. An automated approach for recommending when to stop performance tests. In *Proceedings of the 2016 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 279–289.

[22] Lixiang Ao, George Porter, and Geoffrey M Voelker. 2022. Faasnap: Faas made fast using snapshot-based vms. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 730–746.

[23] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. 2010. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of Engineering and Information Technology* 2, 1 (2010), 60–63.

[24] Andreas Burger, Heiko Koziolek, Julius Rückert, Marie Platenius-Mohr, and Gösta Stomberg. 2019. Bottleneck identification and performance modeling of OPC UA communication models. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*. 231–242.

[25] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. 2022. ALASTOR: Reconstructing the provenance of serverless intrusions. In *Proceedings of the 31st USENIX Security Symposium*. 2443–2460.

[26] Simon Eismann, Cor-Paul Bezemer, Weiyi Shang, Dušan Okanović, and André van Hoorn. 2020. Microservices: A performance tester's dream or nightmare?. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 138–149.

[27] Simon Eismann, Long Bui, Johannes Grohmann, Cristina Abad, Nikolas Herbst, and Samuel Kounev. 2021. Sizeless: Predicting the optimal size of serverless functions. In *Proceedings of the 22nd International Middleware Conference*. 248–259.

[28] Simon Eismann, Diego Elias Costa, Lizhi Liao, Cor-Paul Bezemer, Weiyi Shang, André van Hoorn, and Samuel Kounev. 2022. A case study on the stability of performance tests for serverless applications. *Journal of Systems and Software* 189 (2022), 111294.

[29] Simon Eismann, Joel Scheuner, Erwin Van Eyk, Maximilian Schwinger, Johannes Grohmann, Nikolas Herbst, Cristina Abad, and Alexandru Iosup. 2021. The state of serverless applications: Collection, characterization, and community consensus. *IEEE Transactions on Software Engineering* 48, 10 (2021), 4152–4166.

[30] Nafise Eskandani and Guido Salvaneschi. 2021. The Wonderless dataset for serverless computing. In *Proceedings of the 2021 IEEE/ACM 18th International Conference on Mining Software Repositories*. IEEE, 565–569.

[31] Sadjad Fouladi, Riad S Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, fast and slow: Low-Latency video processing using thousands of tiny threads. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation*. 363–376.

[32] Alexander Fuerst and Prateek Sharma. 2022. Locality-aware load-balancing for serverless clusters. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*. 227–239.

[33] Diandian Gu, Yihao Zhao, Yinmin Zhong, Yifan Xiong, Zhenhua Han, Peng Cheng, Fan Yang, Gang Huang, Xin Jin, and Xuanzhe Liu. 2023. ElasticFlow: An elastic serverless training platform for distributed deep learning. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 266–280.

[34] Hassan B Hassan, Saman A Barakat, and Qusay I Sarhan. 2021. Survey on serverless computing. *Journal of Cloud Computing* 10, 1 (2021), 1–29.

[35] Sen He, Tianyi Liu, Palden Lama, Jaewoo Lee, In Kee Kim, and Wei Wang. 2021. Performance testing for cloud computing with dependent data bootstrapping. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*. 666–678.

[36] Sen He, Glenna Manns, John Saunders, Wei Wang, Lori Pollock, and Mary Lou Soffa. 2019. A statistics-based performance testing methodology for cloud applications. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 188–199.

[37] Zhuo Huang, Hao Fan, Chaoyi Cheng, Song Wu, and Hai Jin. 2023. Duo: Improving data sharing of stateful serverless applications by efficiently caching multi-read data. In *Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 875–885.

[38] Abhinav Jangda, Donald Pinckney, Yuriy Brun, and Arjun Guha. 2019. Formal foundations of serverless computing. *Proceedings of the ACM on Programming Languages* 3, 149 (2019), 1–26.

[39] Chao Jin, Zili Zhang, Xingyu Xiang, Songyun Zou, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Ditto: Efficient serverless analytics with elastic parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 406–419.

[40] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. Cloud programming simplified: A Berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383* (2019).

[41] Jeongchul Kim and Kyungyong Lee. 2019. Functionbench: A suite of workloads for serverless cloud function service. In *Proceedings of the IEEE 12th International Conference on Cloud Computing*. IEEE, 502–504.

[42] Swaroop Kotni, Ajay Nayak, Vinod Ganapathy, and Arkaprava Basu. 2021. Faastlane: Accelerating function-as-a-service workflows. In *Proceedings of the 2021 USENIX Annual Technical Conference*. 805–820.

[43] Christoph Laaber, Joel Scheuner, and Philipp Leitner. 2019. Software microbenchmarking in the cloud. How bad is it really? *Empirical Software Engineering* 24, 4 (2019), 2469–2508.

[44] Jean-Yves Le Boudec. 2010. *Performance evaluation of computer and communication systems*. Vol. 2. Epfl Press Lausanne.

[45] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Cheng-Zhong Xu. 2022. Serverless computing: State-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing* (2022).

[46] Zijun Li, Yushi Liu, Linsong Guo, Quan Chen, Jiagan Cheng, Wenli Zheng, and Minyi Guo. 2022. FaaSFlow: Enable efficient workflow execution for function-as-a-service. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 782–796.

[47] Zijun Li, Chuhao Xu, Quan Chen, Jieru Zhao, Chen Chen, and Minyi Guo. 2023. DataFlower: Exploiting the data-flow paradigm for serverless workflow orchestration. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 57–72.

[48] Changyuan Lin and Hamzeh Khazaei. 2020. Modeling and optimization of performance and cost of serverless applications. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 615–632.

[49] Xuanzhe Liu, Jinfeng Wen, Zhenpeng Chen, Ding Li, Junkai Chen, Yi Liu, Haoyu Wang, and Xin Jin. 2023. FaaSLight: general application-Level cold-start latency optimization for Function-as-a-Service in serverless computing. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–29.

[50] Xuanzhe Liu, Yihao Zhao, Shufan Liu, Xiang Li, Yibo Zhu, Xin Liu, and Xin Jin. 2024. MuxFlow: efficient GPU sharing in production-level clusters with more than 10000 GPUs. *Science China Information Sciences* 67, 12 (2024), 1–17.

[51] Yi Liu, Yun Ma, Xusheng Xiao, Tao Xie, and Xuanzhe Liu. 2023. LegoDroid: flexible Android app decomposition and instant installation. *Science China Information Sciences* 66, 4 (2023), 142103.

[52] Yushi Liu, Shixuan Sun, Zijun Li, Quan Chen, Sen Gao, Bingsheng He, Chao Li, and Minyi Guo. 2024. FaaSGraph: Enabling scalable, efficient, and cost-effective graph processing with serverless computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 385–400.

[53] Ashraf Mahgoub, Li Wang, Karthick Shankar, Yiming Zhang, Huangshi Tian, Subrata Mitra, Yuxing Peng, Hongqi Wang, Ana Klimovic, Haoran Yang, et al. 2021. SONIC: Application-aware data passing for chained serverless applications. In *Proceedings of the 2021 USENIX Annual Technical Conference*. 285–301.

[54] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the three rights: Sizing, bundling, and prewarming for serverless DAGs. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation*. 303–320.

[55] Pascal Maissen, Pascal Felber, Peter Kropf, and Valerio Schiavoni. 2020. FaaSdom: A benchmark suite for serverless computing. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*. 73–84.

[56] Anupama Mampage, Shanika Karunasekera, and Rajkumar Buyya. 2021. A holistic view on resource management in serverless computing environments: taxonomy and future directions. *Comput. Surveys* (2021).

[57] Aleksander Maricq, Dmitry Duplyakin, Ivo Jimenez, Carlos Maltzahn, Ryan Stutsman, and Robert Ricci. 2018. Taming performance variability. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation*. 409–425.

[58] Garrett McGrath and Paul R Brenner. 2017. Serverless computing: Design, implementation, and performance. In *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops*. IEEE, 405–410.

[59] Shaikh Mostafa, Xiaoyin Wang, and Tao Xie. 2017. Perfranker: Prioritization of performance regression tests for collection-intensive software. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 23–34.

[60] Ingo Müller, Renato Marroquín, and Gustavo Alonso. 2020. Lambada: Interactive data analytics on cold data using serverless cloud infrastructure. In *Proceedings of the 2020 International Conference on Management of Data*. 115–130.

[61] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, et al. 2021. OFC: an opportunistic caching system for FaaS platforms. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 228–244.

[62] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. SOCK: Rapid task provisioning with serverless-optimized containers. In *Proceedings of the 2018 USENIX Annual Technical Conference*. USENIX Association, 57–70.

[63] Liam Patterson, David Pigorovsky, Brian Dempsey, Nikita Lazarev, Aditya Shah, Clara Steinhoff, Ariana Bruno, Justin Hu, and Christina Delimitrou. 2022. HiveMind: a hardware-software system stack for serverless edge swarms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 800–816.

[64] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. 2020. Starling: A scalable query engine on cloud functions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 131–141.

[65] Sheng Qi, Xuanzhe Liu, and Xin Jin. 2023. Halfmoon: Log-optimal fault-tolerant stateful serverless computing. In *Proceedings of the 29th Symposium on Operating Systems Principles*. 314–330.

[66] Joel Scheuner and Philipp Leitner. 2020. Function-as-a-service performance evaluation: A multivocal literature review. *Journal of Systems and Software* 170 (2020), 110708.

[67] Mohammad Shahrad, Rodrigo Fonseca, Íñigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *Proceedings of the 2020 USENIX Annual Technical Conference*. 205–218.

[68] Arjun Singhvi, Arjun Balasubramanian, Kevin Houck, Mohammed Danish Shaikh, Shivaram Venkataraman, and Aditya Akella. 2021. Atoll: A scalable low-latency serverless platform. In *Proceedings of the ACM Symposium on Cloud Computing*. 138–152.

[69] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. 2021. 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*. 559–572.

[70] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. 2020. Is big data performance reproducible in modern cloud networks?. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*. 513–527.

[71] Kai-Ting Amy Wang, Rayson Ho, and Peng Wu. 2019. Replayable execution optimized for page sharing for a managed runtime environment. In *Proceedings of the Fourteenth EuroSys Conference*. 1–16.

[72] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *Proceedings of the 2018 USENIX Annual Technical Conference*. 133–146.

[73] Wei Wang, Ningjing Tian, Sunzhou Huang, Sen He, Abhijeet Srivastava, Mary Lou Soffa, and Lori Pollock. 2018. Testing cloud applications under cloud-uncertainty performance effects. In *Proceedings of the IEEE 11th International Conference on Software Testing, Verification and Validation*. IEEE, 81–92.

[74] Jinfeng Wen, Zhenpeng Chen, Xin Jin, and Xuanzhe Liu. 2023. Rise of the planet of serverless computing: A systematic review. *ACM Transactions on Software Engineering and Methodology* 32, 5 (2023), 1–61.

[75] Jinfeng Wen, Zhenpeng Chen, Yi Liu, Yiling Lou, Yun Ma, Gang Huang, Xin Jin, and Xuanzhe Liu. 2021. An empirical study on challenges of application development in serverless computing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 416–428.

[76] Jinfeng Wen, Zhenpeng Chen, Federica Sarro, and Xuanzhe Liu. 2023. Revisiting the performance of serverless computing: An analysis of variance. *arXiv preprint arXiv:2305.04309v1* (2023).

[77] Jinfeng Wen, Zhenpeng Chen, Federica Sarro, and Shangguang Wang. 2025. Unveiling overlooked performance variance in serverless computing. *Empirical Software Engineering* 30, 2 (2025), 59.

[78] Jinfeng Wen and Yi Liu. 2021. A measurement study on serverless workflow services. In *Proceedings of the 2021 IEEE International Conference on Web Services*. IEEE, 741–750.

[79] Jinfeng Wen, Yi Liu, Zhenpeng Chen, Junkai Chen, and Yun Ma. 2021. Characterizing commodity serverless computing platforms. *Journal of Software: Evolution and Process* (2021), e2394.

[80] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 640–654.

[81] Bingyang Wu, Ruidong Zhu, Zili Zhang, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. dLoRA: Dynamically orchestrating requests and adapters for LoRA LLM serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation*. 911–927.

[82] Hao Wu, Junxiao Deng, Hao Fan, Shadi Ibrahim, Song Wu, and Hai Jin. 2023. QoS-aware and cost-efficient dynamic resource allocation for serverless ML workflows. In *Proceedings of the 2023 IEEE International Parallel and Distributed Processing Symposium*. IEEE, 886–896.

[83] Song Wu, Zhiheng Tao, Hao Fan, Zhuo Huang, Xinmin Zhang, Hai Jin, Chen Yu, and Chun Cao. 2022. Container lifecycle-aware scheduling for serverless computing. *Software: Practice and Experience* 52, 2 (2022), 337–352.

[84] Song Wu, Zhiheng Tao, Hao Fan, Zhuo Huang, Xinmin Zhang, Hai Jin, Chen Yu, and Chun Cao. 2022. Container lifecycle-aware scheduling for serverless computing. *Software: Practice and Experience* 52, 2 (2022), 337–352.

[85] Yuncheng Wu, Tien Tuan Anh Dinh, Guoyu Hu, Meihui Zhang, Yeow Meng Chee, and Beng Chin Ooi. 2022. Serverless data science - are we there yet? A case study of model serving. In *Proceedings of the 2022 International Conference on Management of Data*. 1866–1875.

[86] Zhengjun Xu, Haitao Zhang, Xin Geng, Qiong Wu, and Huadong Ma. 2019. Adaptive function launching acceleration in serverless computing platforms. In *Proceedings of the IEEE 25th International Conference on Parallel and Distributed Systems*. 9–16.

[87] Tianyi Yu, Qingyuan Liu, Dong Du, Yubin Xia, Binyu Zang, Ziqian Lu, Pingchao Yang, Chenggang Qin, and Haibo Chen. 2020. Characterizing serverless platforms with serverlessbench. In *Proceedings of the 2020 ACM Symposium on Cloud Computing*. 30–44.

[88] Miao Zhang, Yifei Zhu, Cong Zhang, and Jiangchuan Liu. 2019. Video processing with serverless computing: A measurement study. In *Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*. 61–66.

[89] Laiping Zhao, Yanan Yang, Yiming Li, Xian Zhou, and Keqiu Li. 2021. Understanding, predicting and scheduling serverless workloads under partial interference. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[90] Yutong Zhao, Lu Xiao, Wang Xiao, Bihuan Chen, and Yang Liu. 2019. Localized or architectural: An empirical study of performance issues dichotomy. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings*. IEEE, 316–317.

[91] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized Large Language Model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation*. 193–210.