How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads

Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow
Huawei Edinburgh Research Centre
Edinburgh, United Kingdom

Jianfeng Wang
Hangzhou Research Centre, Central Software Institute, Huawei
Hangzhou City, China

Adam Barker
sirlab@huawei.com
Huawei Edinburgh Research Centre and School of Computer Science, University of St Andrews
United Kingdom

ABSTRACT

This paper releases and analyzes two new Huawei cloud serverless traces. The traces span a period of over 7 months with over 1.4 trillion function invocations combined. The first trace is derived from Huawei's internal workloads and contains detailed per-second statistics for 200 functions running across multiple Huawei cloud data centers. The second trace is a representative workload from Huawei's public FaaS platform. This trace contains per-minute arrival rates for over 5000 functions running in a single Huawei data center. We present the internals of a production FaaS platform by characterizing resource consumption, cold-start times, programming languages used, periodicity, per-second versus per-minute burstiness, correlations, and popularity. Our findings show that there is considerable diversity in how serverless functions behave: requests vary by up to 9 orders of magnitude across functions, with some functions executed over 1 billion times per day; scheduling time, execution time and cold-start distributions vary across 2 to 4 orders of magnitude and have very long tails; and function invocation counts demonstrate strong periodicity for many individual functions and on an aggregate level. Our analysis also highlights the need for further research in estimating resource reservations and time-series prediction to account for the huge diversity in how serverless functions behave.

SoCC '23, October 30–November 1, 2023, Santa Cruz, CA, USA

ACM ISBN 979-8-4007-0387-4/23/11…$15.00
https://doi.org/10.1145/3620678.3624783

CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; • **Computer systems organization** → **Cloud computing**.

KEYWORDS

cloud, serverless, datasets, neural networks, time series

ACM Reference Format:
Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, and Adam Barker. 2023. How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads. In *ACM Symposium on Cloud Computing (SoCC '23), October 30–November 1, 2023, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3620678.3624783

1 INTRODUCTION

Serverless computing and Function-as-a-Service (FaaS) provides cloud programmers with a convenient programming paradigm for event-based workloads [29]. Today, serverless computing is widely used for applications ranging from media processing [22] to processing requests from vending machines [19]. While serverless adoption has been increasing, there are few available studies into how production serverless systems actually perform, characterizing behavior of both users and systems in operation [29]. However, such an understanding is important for systems research on how to build, manage, and enhance serverless offerings in cloud data centers. In addition, having open datasets allows for new insights into these systems and how they operate. To the best of our knowledge, there is only one existing serverless dataset available for the research community spanning 14 days of Azure function invocations [26]. This dataset contains over 70 thousand functions, but provides only a limited number of features, a short duration of 14 days, and a coarse-grained log of per-minute invocations.

In this paper, we analyze and open-source two traces from Huawei serverless systems [1]. The first trace is at per-second

[1] Huawei traces available at https://github.com/sir-lab/data-release

granularity and consists of 200 functions from private internal Huawei workloads running across multiple data centers with 141 days of data collected over a period of 234 days. This provides insights into fine-grained performance of these systems including response times, cold-start times, and number of invocations per second for each function, as well as average CPU and memory usage per minute for each function. The second trace is a coarse-grained trace of over 5000 functions hosted in one availability zone from one Huawei data center for 26 consecutive days. This second trace provides a coarse-grained understanding of the evolution of the workload on a single availability zone for over 5000 functions.

This paper reports the first in-depth analysis of these two Huawei production traces, covering statistical features of our workloads, followed by a longitudinal analysis of periodicity and ranking of functions across our traces. We then formulate the prediction of these function invocations as a time series forecasting problem using a *global univariate model* and demonstrate the challenges of forecasting with models of varying complexities on both traces. Our main contributions can be summarized as follows:

- **Characterization of production FaaS workloads:** We provide a first-of-its-kind analysis of two long-term traces (26 days; 141 days spanning 234 days, with gaps) from production serverless systems, together comprising over 5200 functions. Our analysis covers request arrival distributions, cold start, scheduling, network, function execution times, code package size, resource usage and feature-level correlations.
- **Longitudinal periodicity and ranking analysis:** We conduct an in-depth longitudinal analysis of periodicity and ranking of functions across our traces. These analyses are of particular importance regarding function invocation, as they demonstrate a paradigm of data that exhibits trends over the long-term (many days), yet which is also fine-grained.
- **Workload forecasting:** We demonstrate the efficacy of several machine learning models on forecasting function request data, with the intended application of autoscaling and scheduling. We provide an argument for why global univariate models are better suited (compared to standard univariate or multivariate models) to this type of data. We also identify a challenge resulting from such data: ingesting and forecasting very fine-grained (per second) and long-term (over several days or weeks). We refer to such data as FGLT data. Facing this challenge may require rethinking how modern methods operate. We hope to see novel modeling approaches emerge based on this data and the associated challenge.
- **Open-source trace release:** We open-source[1] both traces to the research community, encouraging further analysis on serverless systems. Our traces provide a rich set of metrics, and with 141 days of data, facilitate longitudinal studies to better understand serverless systems over longer durations.

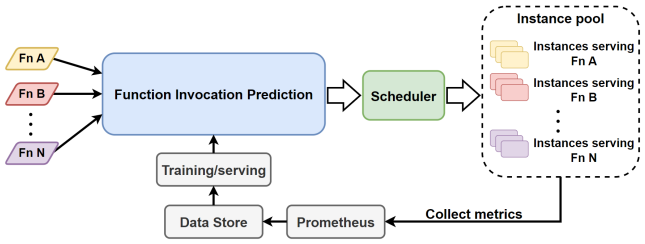


Figure 1: Architecture of our serverless platforms and where our function invocation prediction model will be used.

2 OVERVIEW OF OUR DATASETS

This section describes the lengths and features of our traces as well as granularities and other aggregate characteristics.

2.1 Serverless at Huawei

Serverless computing facilitated by Function-as-a-Service (FaaS) platforms allow developers to build, run and deploy event-driven stateless functions without the overhead of having to provision and manage servers or backend infrastructure. This, in turn, allows developers to focus on the functions they develop, rather than how they will run on the cloud [14, 17]. Today, all major cloud providers have serverless offerings. Under the hood, middleware such as Apache OpenWhisk [3], Knative [18], or OpenShift Serverless [24] are used to manage the functions submitted to the system. The functions are then mapped to either containers [41] or lightweight virtual machines [1]. These containers are then orchestrated into larger applications using cloud orchestration software such as Kubernetes.

Serverless systems at Huawei are built on top of containers and *YuanRong*, a scalable distributed computing kernel for serverless workloads. YuanRong has been widely deployed across China, Europe and Asia Pacific in nearly 20 availability zones called 'regions', and serves tens of thousands of enterprise customers across a diverse range of workloads including data analysis, business IT applications, and deep neural network model training and serving. YuanRong is used to process up to 20 billion requests per day. Figure 1 shows a simplified architecture of YuanRong.

Function invocation prediction is the task of predicting the number of function invocations arriving on the platform at a given time for a given function. Predictions produced by a time series forecasting model inform the scheduler and are used to preemptively allocate an appropriate number of

Dataset	Field	Description	Res	Unit
Huawei Private	Function ID	Unique function identifier	-	-
	Timestamp	Time when request is received	sec	sec
	Requests	Number of function invocations	sec	-
	Function delay	Average execution time averaged over all pods running that function	sec	ms
	Platform delay	Average platform overhead averaged over all pods running that function	sec	ms
	CPU usage	Percentage of allocated CPU used per function averaged over all pods	min	%
	Mem usage	Percentage of allocated memory used per function averaged over all pods	min	%
	CPU limit	Allocated CPU for all pods running that function (normalised)	min	cores
	Mem limit	Allocated memory for all pods running that function	min	MB
	Instances	Number of pods allocated to that function	min	-
Huawei Public	Function ID	Unique function identifier	-	-
	Timestamp	Time when request is received	min	min
	Requests	Number of function invocations	min	-
	Cold start cost*	Cold start times plus some other overheads	sec	ms
	Package sizes*	Size of functions	-	MB
	Language*	Programming language used	-	-

Table 1: Summary of our dataset fields with each field's associated time-measurement granularity. The data will be released as an open-source repository following the format in the Table. Note that fields with an asterisk (*) will not be released.

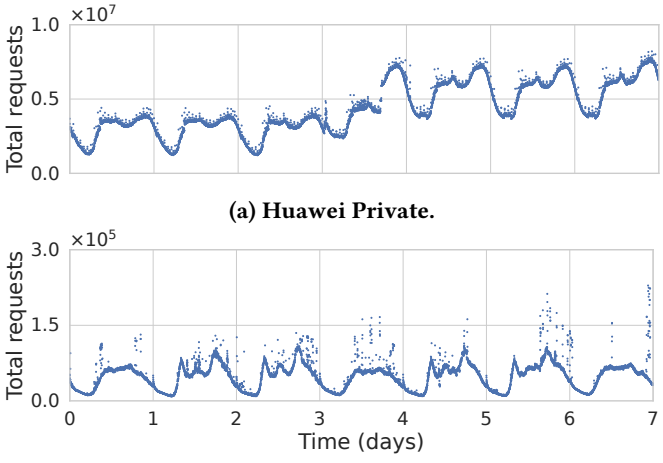
pods to serve a particular function in order to reduce the number and time of cold starts. On Huawei platforms, the most popular functions are predicted with powerful models and predictions are made every day for the entire subsequent day. Predictions may be made for every minute or even every second of the subsequent day. These forecasting models use historical data, which is collected by Prometheus, to inform their predictions, as shown in Figure 1.

2.2 Our Datasets

In this paper, we present and analyze two new datasets from YuanRong's serverless workloads collected using the pipeline in Figure 1. These two traces span a period of more than 7 months with over 1.4 trillion function invocations in both datasets combined. The two traces we open-source and analyze in this paper are:

(1) **Huawei Cloud Private Functions Trace 2023 (Huawei Private):** This trace is derived from Huawei's internal workloads. It contains eight different metrics for 200 functions for a period spanning 234 days. Some metrics are reported per second, while others are reported per minute. This trace contains data on the platform itself including CPU and memory usage and limits, along with platform and function delays.

(2) **Huawei Cloud Public Functions Trace 2023 (Huawei Public):** This trace is a representative workload from applications running on Huawei's public-facing FaaS platform. Huawei Public contains more functions than Huawei Private but is reported at the minute level and over a shorter duration, totaling 5019 functions over 26 days. We also analyze language types, package sizes, and cold start times for this platform, but do not release these.



(a) Huawei Private.

(b) Huawei Public.

Figure 2: Sum of all requests per minute plotted for the first 7 days of Huawei Private and Huawei Public.

Table 1 displays a summary of both traces with their respective fields. Figure 2 shows the total number of invocations for each of the two traces for a sample week. Most of the analysis in this paper is done on the Huawei Private

trace since it contains more features. We split our characterization and analysis into three different parts. Section 3 focuses on the statistical features of our datasets. Section 4 discusses periodicity, rankings, and correlations within and across functions that inform certain design choices for ML approaches in Section 5. Section 5 describes how to use the insights from previous sections in order to formulate function invocation prediction as a tractable time series forecasting problem, as well as some challenges that arise when forecasting fine-grain per-second data.

3 STATISTICAL FEATURES OF OUR WORKLOADS

This Section characterizes the various features of our datasets, including invocation counts, memory and CPU usage, number of instances, execution times, cold start times, overhead times, code package size, and runtime languages. We focus on the statistical features of workloads using cumulative distribution functions (CDFs). In CDF plots with *all*, *median*, *min*, and *max* curves; *all* represents the CDF of all values across all functions; while *median*, *min*, and *max* are calculated per function (i.e., a percentile calculated for all functions over the entire duration of the dataset), and the CDF is calculated as a cumulative fraction of the number of functions.

3.1 Request arrivals

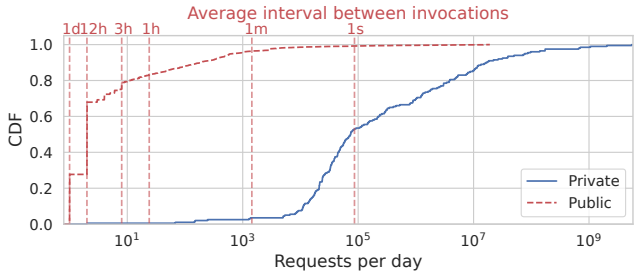


Figure 3: Requests on a median day per function, with the corresponding average interval between requests for Huawei Private and Huawei Public.

Figure 3 shows a CDF of functions by their average number of daily invocations from our two traces. The top horizontal axis shows the mean arrival rate between invocations corresponding to the requests per day on the bottom horizontal axis. To measure requests per day, we compute the median value of each minute of the day across all days in the dataset and take the sum of requests on this median day. Note that the invocation counts from Huawei Private represent 200 functions of internal Huawei applications using serverless across *all* regions, while Huawei Public contains 5093 functions from *one* region.

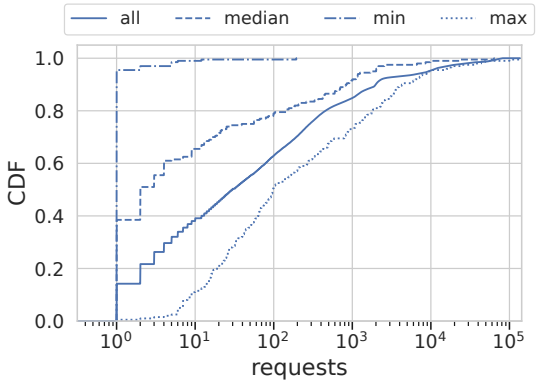
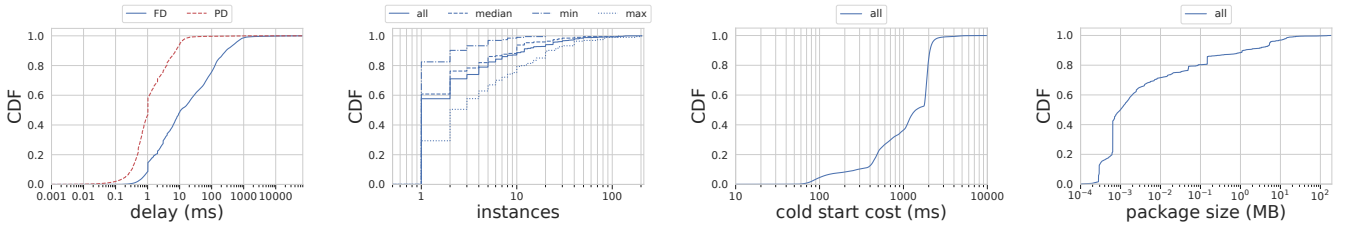


Figure 4: Huawei Private request arrivals per-second.

Huawei Public has a wide distribution of popularity of functions as measured by requests per day and demonstrates the variety of use within the public FaaS platform. Huawei Private on the other hand has fewer but far more highly requested functions, with some being invoked over 1 billion times per day. Approximately 50% of functions in Huawei Private are invoked at least once per second, while less than 10% of functions in Huawei Public are invoked at least once per minute. Thirty percent of functions in Huawei Public are invoked only once per day, and around 70% of Huawei Public functions are invoked at most once every 12 hours. In both datasets, a minority of top functions account for the majority of traffic. This phenomenon is especially present in Huawei Private.

To better understand the request arrival rate on a per-second scale, Figure 4 plots the CDF of *all* function invocation counts per second for the Huawei Private dataset. This CDF shows that at least 40% of all invocation counts have over 100 invocations per-second at any time, with more than 10% having more than 1000 requests per second. In addition to the arrival rate distribution, we plot the distributions of the median, min, and max arrival rates for each of the Huawei Private functions. These distributions can be helpful when designing schedulers using our dataset as they can enable new scheduling approaches that look at the probability of worst and best-case arrivals of requests at any given time.

> The number of requests per day varies by nine orders of magnitude across functions in Huawei Private and seven orders of magnitude in Huawei Public. Huawei Public has a similar profile to Azure [29], most likely because both are public serverless traces, while Huawei Private has much larger invocation counts.



(a) Function Delay, Platform Delay in Huawei Private.

(b) Number of allocated instances in Huawei Private.

(c) Cold start costs measured at Huawei Public.

(d) Package size per function in Huawei Public.

Figure 5: The user-perceived times from serverless can be dominated by the platform delay (which includes the scheduling and network setup), by the execution times, or by the cold-starts suggesting that these three metrics need to be optimized. The number of instances and the package sizes can also affect cold starts.

3.2 Execution time, platform delay, cold-start, and instances

One potential bottleneck with serverless systems is *cold-starts* [12, 31]. Due to the burstiness of serverless workloads, the system may not have enough containers already provisioned to service incoming request bursts [20, 37]. The system, therefore, needs to start new containers to accommodate these bursts, which adds latency and degrades application performance. This cold-start time is affected by the network cold-start time [31], the sizes of the containers, and the need to pull containers from registries [36], among other costs that can affect the total cold-start.

Huawei Private delays analysis. For Huawei Private, we analyze two delays, namely, **function delay** and **platform delay**. Function delay is the function execution time measured by our system, i.e., the time it takes a function to complete its task once scheduled. Platform delay includes both scheduling and network delays. Platform delay has previously been found to dominate user-perceived cold-start times in "burst-parallel" serverless jobs [31]. Burst-parallel serverless jobs are generated by applications that invoke thousands of short-lived distributed functions to complete complex jobs requiring the system to start a large number of containers that requires interconnections [7, 31]. Since these burst-parallel functions dominate our Huawei Private data sets, and since the Huawei Private functions represent real applications used by millions of customers using internal services, we analyze these two delays for the Huawei Private dataset.

Figure 5a shows the CDFs of both platform and function delays for Huawei Private. Over 60% of function requests have a platform delay of less than 1ms. Almost all platform delays are less than 10ms. Confirming previous research [31], the platform delays have a very long tail due to the burst-parallel nature of the workloads. Next to platform delay, we also plot function delay on the same figure. We see that

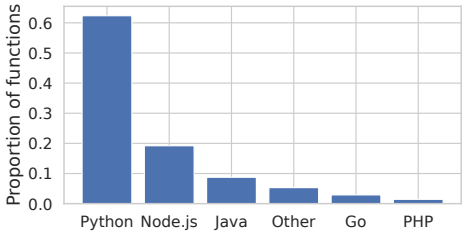


Figure 6: Proportion of functions by runtime language in Huawei Public.

50% of functions execute in less than 10ms, and almost all functions complete within 1 second.

To better understand the long tails in platform delay, we plot the number of instances allocated per function per-minute (solid blue labeled *all*) for Huawei Private in Figure 5b as the number of instances can strongly influence the network-startup times [31]. Even though a significant percentage of our workload has at least 100 invocations per second (roughly 60%), we allocate only one instance to serve 60% of the functions, with only less than 1% of our allocations being above 100 instances. We believe that these larger allocations are one reason for the longer platform delays as for many functions, the platform delay and the number of instances are correlated as we show later.

Huawei Public delays analysis. Huawei Public represents external workloads deployed by our customers, typically with much fewer insights from our teams on what is actually running. Hence, for Huawei Public, we decided to analyze the cold-start costs which measure the total time for a function to start. Figure 5c shows the cold start cost for Huawei Public. We see that approximately 40% of cold starts take less than 1 second, while over 90% of requests of cold starts take less than 2 seconds. Major influences on cold-start times include package sizes, memory usage, and language runtimes used [37]. Figure 5d shows the package

sizes for the functions in Huawei Public which is an indication of the size of the code running, while Figure 6 shows the main languages used for implementing these functions. Our data analysis shows that package sizes, the language of implementation, and memory allocation are the three most decisive factors on the cold-start time [37].

> Platform delays, execution time, and cold-start time distributions vary considerably and have very long tails. This indicates the importance of optimizing all three delays when building serverless platforms. However, this is especially challenging given the diversity of runtime languages used and their associated performance overheads.

3.3 Resource Usage

In serverless systems, each function is typically assigned a resource limit by the user that represents the maximum amount of CPU and memory that the function can use. However, functions do not necessarily use all of their assigned resources. To better understand usage versus allocations, we focus on data from Huawei Private. Figure 7 shows the CDFs of absolute CPU usage, CPU limit, absolute memory usage, and memory limit, for all allocations (marked *all*) as well as median, maximum, and minimum for each function. Our first observation is that most user-specified limits are much higher than the actual usage, with around 60% of all allocations using less than 0.1 cores but asking for a limit of more than 1 core. Looking at memory usage in Figure 7c versus memory limits in Figure 7d, we see a similar trend, with around 50% of the functions having a limit of around 2 GB, but using only around 400 MB. This shows how users of serverless functions are very conservative with their resource requirements. This gives the operators the possibility to reuse some of this slack by over-committing the resources using intelligent scheduling [35].

When over-committing resources, the scheduler needs to consider the worst-case scenarios of usage, which can be calculated from the max curves for CPU and memory usage in Figures 7a and 7c. Over-provisioning by users is less conservative when considering the worst-case scenario, e.g. 10% of functions at some time used 70% or more of their allocated memory. Hence, scheduler over-commitment must reduce the risk of aggressive over-commitments, especially considering that memory over-allocation can result in failures. There is still room for over-committing the resources while not reducing the quality of service.

> Estimating the size of resources required by a function is difficult for many serverless users, leading to over-provisioned resource reservations. This suggests the need for improved and automated approaches for users to determine resource limits, as well as resource reclamation mechanisms that enable serverless operators to reclaim some of these unused resources.

4 THE LIFE OF A FUNCTION

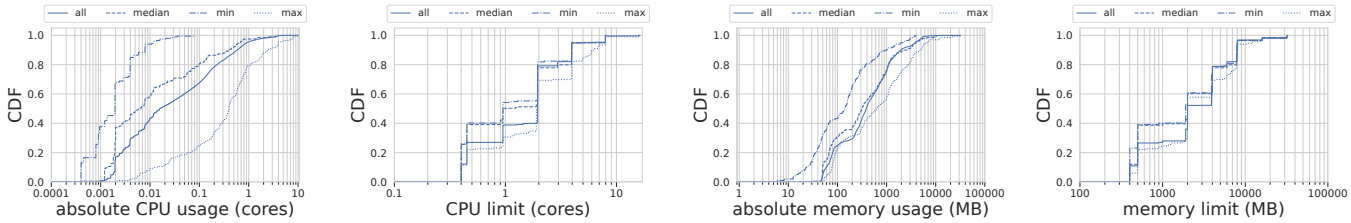
We now focus on individual function behaviors and cross-function relationships. We investigate the following characteristics: periodicity, popularity ranking over time, correlations between top-ranking functions, and differences in burstiness between per-second data and aggregated per-minute data. We also compute and discuss correlations between different features.

4.1 Are functions invocation counts periodic?

One key characteristic of server workload data is periodicity. This periodicity is seen in many server workloads [5, 30, 40], including serverless workloads [20, 29]. Identifying the meaningful periodicities will inform scheduling and autoscaling design choices such as the architecture of the model used for forecasting these traces, how often to invoke it to make a prediction, and how long into the future it should predict.

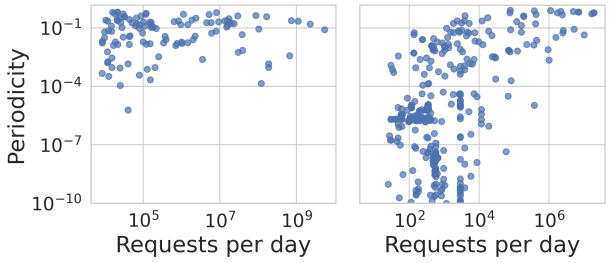
Daily Periodicity. We analyze the workloads' daily periodicity of each function in the traces. We follow this with a comprehensive analysis of periodicity at any period. The daily periodicity of a function can be measured by the normalized peak height of its power spectral density (PSD) at the daily frequency. PSD is a measure of a time-series' power content distribution with frequency. We measure the PSD value for daily periodicity of all functions and plot them in Figure 8 as a scatter plot. Most highly requested functions on each platform have daily periodicity, especially in Huawei Public. Functions with fewer requests per day may also have strong periodicity. There are a few high-request, low-periodicity functions in both datasets, pointing to functions with large, aperiodic bursts. Diurnal patterns are typically due to either human activity during the day, or increased batch processing and analytics jobs during the night.

Other periodicities. To obtain a more complete picture of periodicity in our datasets, we now examine all periodicities. Given a signal's frequency spectrum, the most significant periodicity is defined by the height of the largest peak in the normalized PSD. The frequency at which this peak occurs represents the period with the strongest influence on the



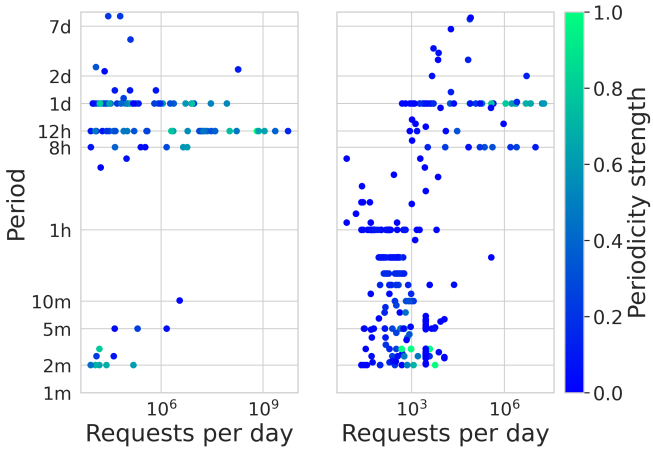
(a) CPU usage. **(b) CPU limit.** **(c) Memory usage.** **(d) Memory limit.**

Figure 7: Distribution of resource usage and limit in Huawei Private.



(a) Huawei Private. **(b) Huawei Public.**

Figure 8: Periodicity vs number of requests on median day. Periodicity of Huawei Private is generally higher than that Huawei Public. Note that both traces have similar trends for periodicity greater than 10^{-4}, but Huawei Public has more low-request functions with insignificant periodicities.



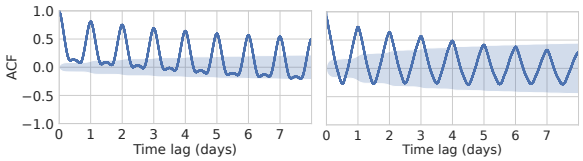
(a) Huawei Private. **(b) Huawei Public.**

Figure 9: Plot of periodicity of different functions. The horizontal axis is the median number of requests received by a function per day. On the vertical axis is the most prominent periodicity of that function (frequency at which PSD is highest). The color is the height of the largest PSD peak in the normalized PSD (at the aforementioned frequency). Many functions have their most significant periods at 1 day and 12-hour periodicities, including some of the most popular functions.

overall signal. We can plot requests per day against the frequency where the PSD is largest, as shown in Figure 9. We can then color the points based on the height of the peak, with 1 representing a signal perfectly periodic at that frequency, and 0 representing a completely aperiodic signal. In this analysis, we only include functions with more than one consecutive day of data since a large number of functions in our workloads are invoked for less than one day.

Figure 9 shows the relationship between invocations per day for each function, the interval at which function invocations have maximum periodicity, and the power of that periodicity from PSD analysis, i.e., the frequency with highest power for each function given its invocation rates. We see significant periodicities at 1 day, 12 hours, 8 hours, and smaller minute-level intervals. This is an interesting observation for serverless system operators. While 24, 12, and 8-hour intervals for strong periodicity can be explained by the day-and-night effect, the per-minute periodicity cannot.

Aggregate periodicity. To understand the overall periodic behavior of workloads, we study the autocorrelation function (ACF) of the total requests for both platforms. At a given lag τ, autocorrelation computes the correlation of a signal with that same signal, but with the latter copy shifted by τ. The ACF is then plotted against lag to visualize the signal's self-similarity when shifted by that lag. Figure 10 shows the ACF with lags up to eight days. Both datasets have peaks in their autocorrelation at integer multiples of one day. Other than small lags of one minute or one hour, autocorrelation is highest at a lag of exactly one day, underscoring the significance of daily periodicity across functions. Comparing the two datasets, Huawei Private has a marginally stronger autocorrelation at a lag of one day. However, Huawei Public has a consistently larger confidence interval. If the correlation is within the confidence interval, then it is not statistically significant. Also note that Huawei Private has minor peaks at half-day lags, where Huawei Public has troughs, which aligns with the greater number of points on the 12h line in Figure 9a than in Figure 9b.



(a) Huawei Private. (b) Huawei Public.

Figure 10: Correlogram of sum of all requests for entire duration against time lag in minutes. The Bartlett confidence intervals are also shown. If autocorrelation is within the confidence interval, it is not significant.

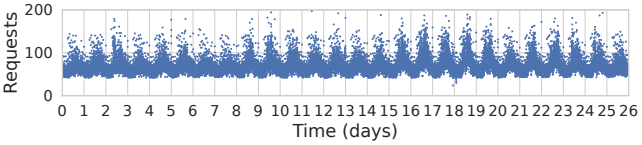


Figure 11: Sum of requests for the 3000 least popular functions in Huawei Public.

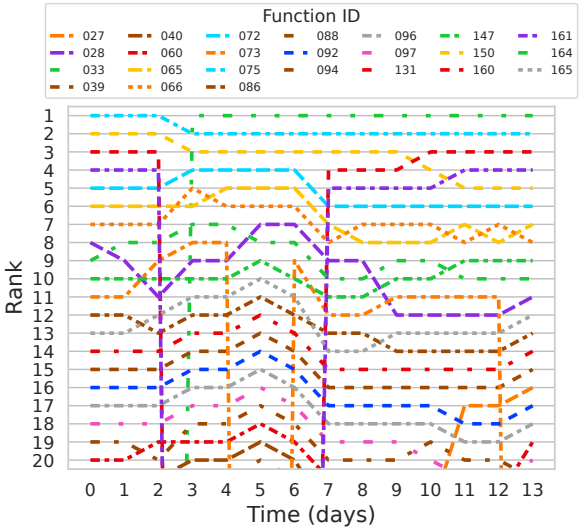
In addition to the top functions and the total number of requests being periodic, we have found that the least popular functions together also exhibit periodicity. This phenomenon is especially significant for Huawei Public, where the bottom 3000 functions together show clear daily periodicity over all 26 days, as shown in Figure 11.This is interesting from the perspective of scheduling and bin packing groups of less popular functions together, which are likely to exhibit higher cold start times.

> A significant number of functions in both platforms have strong periodicity, especially at daily frequencies or frequencies that divide equally into one day, such as 8 or 12 hours. Periodicity is especially significant for more highly requested functions, as well as the aggregate number of requests on our platforms.

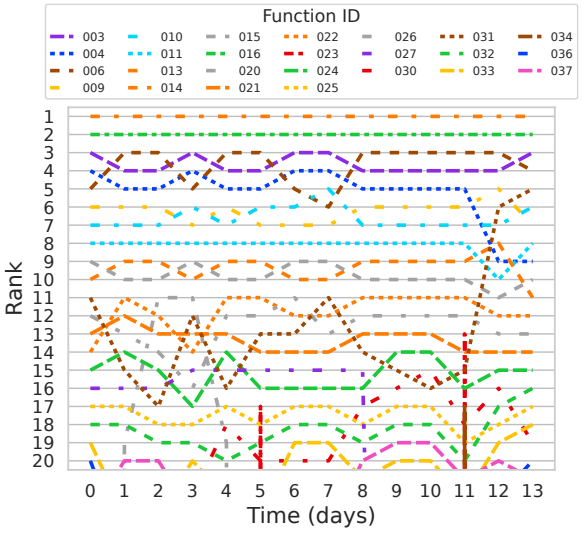
4.2 Function popularity over time

An important aspect to consider is function popularity as the most popular functions constitute a large portion of the traffic on our platforms. Hence, it is important to understand how the ranking of the top functions changes over time.

The ranking of the top functions rarely remains the same for long. To best visualize this dynamic behavior, we show *bump charts* of function ranking for two weeks of each trace. A bump chart shows the ranking on the vertical axis, and how the ranks of these functions change from day to day. Figure 12a shows the bump chart for Huawei Private and Figure 12b shows the bump chart for Huawei Public. Our first observation is that 25 unique functions appear in the top 20 rankings over two weeks for Huawei Public, while 24



(a) Huawei Private.

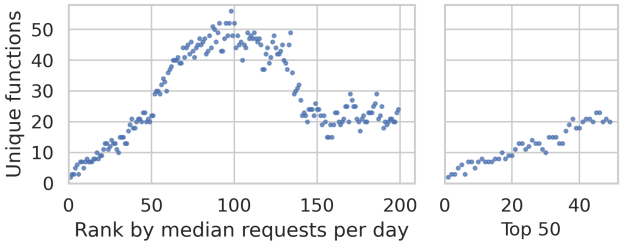


(b) Huawei Public.

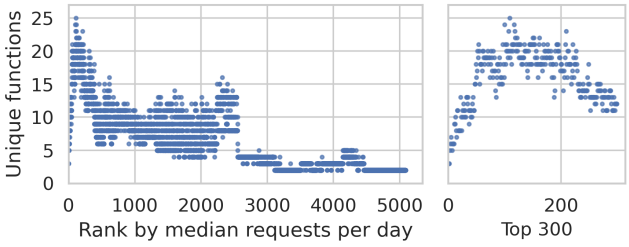
Figure 12: Bump chart visualizing the functions that occupy the top 20 places in the daily ranking of functions by their median number of requests on that day.

appear in the same rankings for Huawei Private, suggesting that there is little change in rankings in the top ranks. Some of these functions appear only briefly.

To better understand the dynamics over the full duration of the trace, we count the number of unique functions per rank, i.e. number of unique functions per row in the bump chart for the entire duration of the trace, and plot this against the rank (ranked by median), as shown in Figure 13a for Huawei Private and Figure 13b for Huawei Public. Figure 13a shows that 2 functions occupy the top spot over the full duration



(a) Huawei Private.



(b) Huawei Public.

Figure 13: Number of unique functions occupying a given rank in Huawei Public and Huawei Private.

of the Huawei Private dataset. Also, note that the number of unique functions per rank peaks at comparable points in both datasets (around rank 100).

> Higher ranks are occupied by a small number of unique functions. Similarly, low-ranking functions also tend to only have very few variations, with most ranking changes happening in the middle ranks. A scheduler could possibly make use of this fact in deciding which functions can be colocated on the same machines.

4.3 Feature level correlations

The Huawei Private dataset has nine metrics (fields) as shown in Table 1. We have added two additional measures of CPU and memory usage: absolute usage (percentage usage multiplied by the limit, so the result is in MB or cores), and total usage (absolute usage multiplied by the number of pods, representing the total memory or CPU used by all pods running this function). Figure 14 shows a heatmap of the correlations between the features for all the Huawei Private functions. The heatmap shows some interesting insights. First of all, both the function and platform delays are only very weakly correlated with any of the other metrics. In addition, the number of requests is only very weakly correlated with any of the other metrics, if at all. There is, however, a correlation between the memory and CPU metrics.

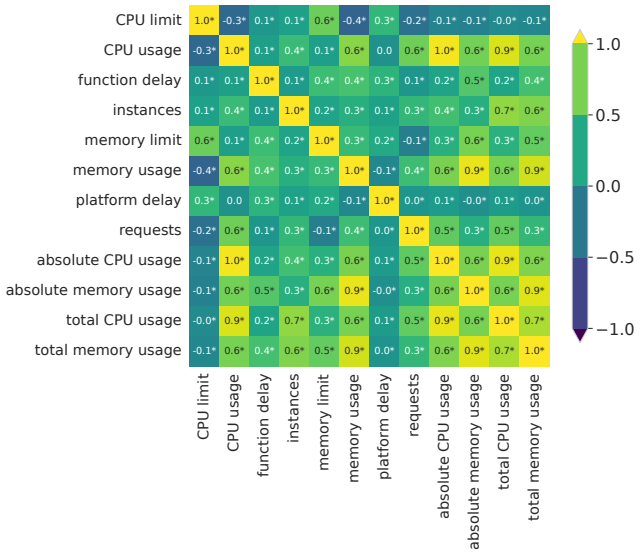


Figure 14: Correlation of all metrics averaged over all functions in Huawei Private.

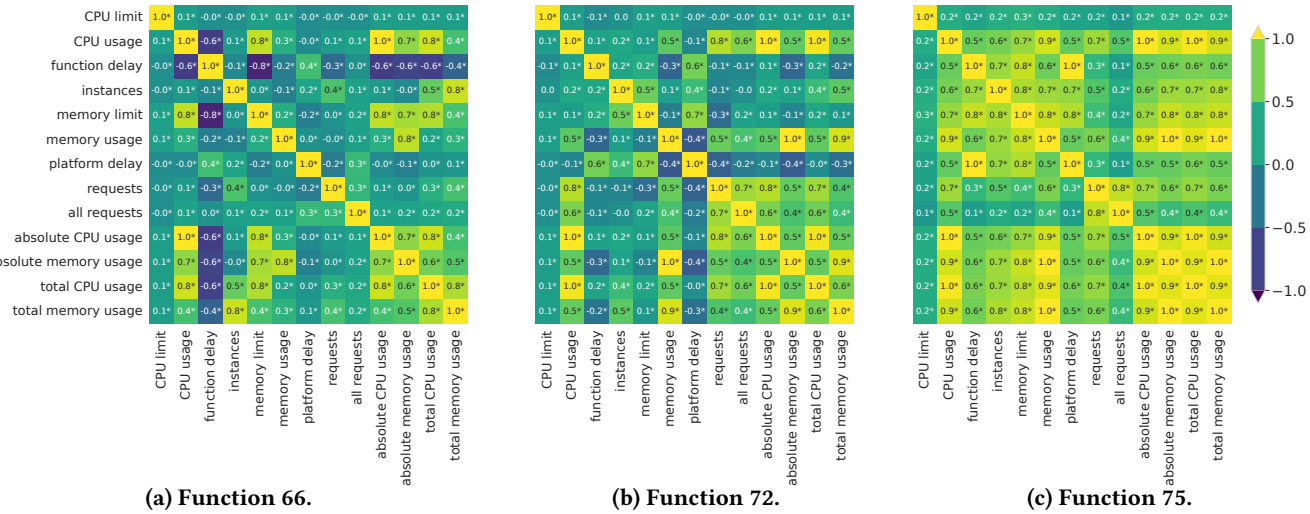
4.4 Resource Usage

Investigating feature-level correlations for individual functions, Figure 15 shows the Spearman correlation heatmap between all features of three example functions from the top 10 most invoked from Huawei Private. Starting with function 72 in Figure 15b, we see that for some metrics, there seem to be slightly stronger correlations compared to the averages in Figure 14 such as between platform delay and memory limit, or between number of requests and resource consumption. These correlations are very strong in function 75 in Figure 15c, with high correlations between function and platform delay and most other metrics. Additionally, for function 66 in Figure 15a we see negative correlation between function delay and a number of other metrics.

4.5 Inter-function correlations

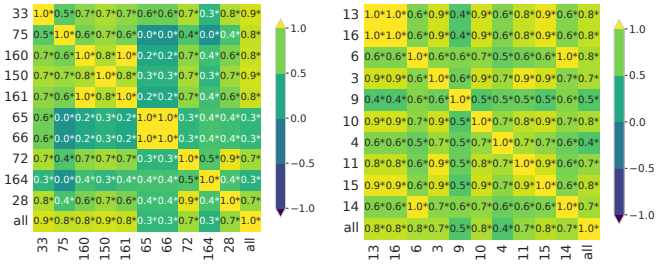
Since a data center hosts a variety of workloads, it is important for operators to know if there are strong correlations between running workloads. For example, knowing if different function invocations have aligned peaks may inform scheduling. Functions can have correlated bursts because of invocation chaining, or the diurnal pattern of users.

Figures 16a and 16b show the correlation heatmap of the top ten functions by invocations for both traces respectively. Each element in the correlation matrix is labeled with its Spearman correlation value, with an asterisk next to values where $p < 0.05$. For both datasets, we see very strong correlations between many of the top ten functions, with some functions almost perfectly correlating with each other, suggesting that they burst together. Since the top ten functions



(a) Function 66. **(b) Function 72.** **(c) Function 75.**

Figure 15: Correlations for three sample functions in Huawei Private.



(a) Huawei Private. **(b) Huawei Public.**

Figure 16: Correlation of requests of top 10 functions and all requests using Spearman correlation.

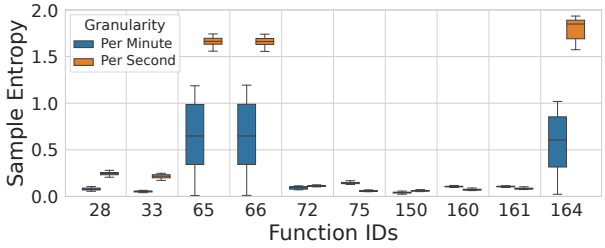


Figure 17: Sample entropy box plot for top 10 functions in Huawei Private for different time granularities.

by invocations can have billions of invocations per day, this means that from a resource management point of view, it is important to understand which functions burst together to be able to allocate enough (warm) resources to reduce or alleviate altogether the cold start-problem.

> Correlated bursts are common within serverless workloads. Understanding these correlations is important to build scalable serverless resource management systems.

4.6 Workload Granularity and Burstiness

One important aspect that needs to be taken into account when building serverless, and cloud management systems in general, is the granularity of data logging. On the one hand, very fine-grained logging increases logging costs. On the other hand, coarse-grained logging may introduce significant changes to some statistical aspects of the data. In this Section, we chose burstiness as a measure of statistical variation due to data logging granularity and demonstrate its effects on arrival requests prediction as a use case.

Measuring burstiness. To measure how bursty a workload is, several measures and approaches have been proposed in the literature [2]. For our analysis, we chose *sample entropy* [27] which is a robust burstiness measure, first developed to classify abnormal (bursty) physiological signals, but has since been used to quantify complexity in network traffic [2, 25, 28]. Sample entropy is defined as "the negative natural logarithm of the (empirical) conditional probability that sequences of length m similar point-wise within a tolerance r are also similar at the next point" [27]. For our workloads, calculating sample entropy effectively compares sliding windows of length m of the workloads checking if they have any deviations in the number of invocations larger than r. We use Python's Antropy library [34] for our burstiness experiments.

Figure 17 shows a box plot of the average per-day sample entropy for the top 10 functions in our Huawei Private dataset when using per-second versus per-minute granularity. For the per-minute data, we use the sum of invocations for a function over a minute and do not average it. The plot

shows that for Functions 65, 66, and 164, the per-second granularity shows increased burstiness compared to the per-minute granularity trace. For other functions, the difference is insignificant, or much smaller, suggesting that the aggregation did not affect that function's burstiness significantly. For three functions, namely 75, 160, and 161, the burstiness increases slightly due to aggregation. This is a side-effect of using the *sum* over a minute rather than the *mean* over a minute since small anomalies can add up during the minute to magnify into a larger burst.

> Our analysis confirms that per-second aggregation reveals more trends compared to per-minute aggregation. Generally, it is important to accurately predict per-second dynamics to be able to adapt apriori to bursts. Per-minute aggregation can lead to sub-par or inaccurate resource management decisions.

5 FORECASTING CHALLENGES

Function request arrival forecasting can be used in combination with an estimate of function execution time to estimate in advance the amount of resources allocated for serving a particular function at a given time. This can mitigate the cold start issue, where demand exceeds allocated resources at that time. Ideally, such predictions should be fine-grained to exploit fine-grained patterns, especially in functions with short execution times. Fine-grained forecasting is important for multiple internal resource management use-cases. For example, per-second forecasting can be used for resource over-allocation for the majority of functions as they have runtimes of a few milliseconds, as shown in Figure 5a.

In this Section, we study and evaluate several time series forecasting models on the fine-grained long-term (FGLT) data described in this paper. We focus on the top 10 functions. Our objective is to demonstrate the challenges of forecasting FGLT data rather than provide a comprehensive solution.

Forecasting is important to cloud performance optimization (e.g. for minimizing cold starts in FaaS). We focus on forecasting function request arrivals as it is a common use-case in cloud systems. The concept of 'fine-grained' data is relative; **in our case, we consider 'fine-grained' to be forecasting tasks where the data sampling rate is much higher than the strongest periodicity**. For example, Figure 10 shows that the strongest periodicity in our data is every 24 hours, while the sampling rate is 86400 samples per day (the number of seconds in a day). Our study illustrates challenges common to all FGLT cloud data as listed below.

(1) **Fine-grained data is expensive for powerful models to ingest and utilize.** For a model to ingest one week of per-second data, it needs to ingest and learn long and short term patterns from $86400 \times 7 = 604,800$ data points. However, ingesting this amount of data is infeasible in most practical cases for most models because of the difficulty of learning such large feature spaces. That is, models have to learn fine-grained minute or second level patterns while also accounting for long-term trends over days or weeks, as shown in Figure 18. As of yet, no models are designed for this paradigm. See Section 5.3 for more information.
(2) **Standard univariate or multivariate models are poorly suited to function request forecasting** since for each new function we want to predict, the model needs to be either retrained or tuned, which is too expensive. This is discussed further in Section 5.1.
(3) **Long-term forecasting** Our system runs tens to hundreds of thousands of functions. Running a prediction for each of these functions frequently is very expensive and the overhead of forecasting can quickly outweigh the utility of predictions. Therefore, we expect a forecast to be relevant for at least one day, such that the model is only queried once per day.

5.1 Experimental setup

Models. We trained eight models on the top 10 functions for both platforms: linear regression, TimesNet [39], N-HiTS [8], FFT extrapolation, NeuralProphet [33], PatchTST [23], TiDE[11], and N-Linear [42]. We describe these models later in detail. Where possible, we trained global univariate models [15] when the original model supports this training regime. *Global univariate models* are models trained on samples from multiple time series. This training regime produces improvements for some scenarios [6, 15, 16] over models that are only trained on samples from the same time series they are trying to predict (local models). Another advantage is that training a global model allows us to predict other single univariate time series that it was not necessarily trained on.

Global univariate models are well-suited to data in cloud infrastructure because:

(1) Global univariate models are typically used where time series are similar or related. Functions often exhibit similar patterns, meaning that a function-specific model is not necessary.
(2) Thousands of functions may require forecasting, so function-specific models are too costly for both training and inference.
(3) Multivariate models (where inputs are variables from multiple functions) are not feasible for thousands of functions. One may consider narrowing down the input to the top-k functions, but the top-k ranking changes over time (see Figure 12) and entirely unseen functions may appear.

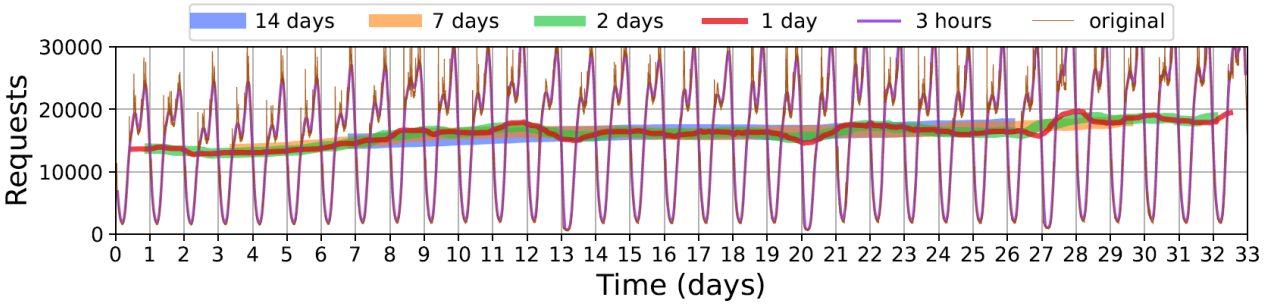


Figure 18: A 33 day chunk of an example function in Huawei Private. The different color lines show different scale trends computed as windowed means of the data with various window sizes (from 14 days to 3 hours).

In our work, only two models do not support global univariate training; NeuralProphet and FFT. These local models are instead fitted to each function individually. We include these two models as they are simple and lightweight models that are widely used. However, they suffer from all the downsides of local models mentioned before.

We now describe the eight models we train in more detail:

(1) **FFT Extrapolation** computes the FFT of the input and extrapolates future values. In our implementation, data is de-trended with a third-degree polynomial and the top 100 frequencies are kept for extrapolation.
(2) **Linear regression** is a simple forecasting model that assumes a linear relationship between output forecasts over the forecast horizon and input features taken from previous values in the sequence.
(3) **N-Linear** is a simple linear model with an additional preprocessing step to account for distributional shift in the data.
(4) **NeuralProphet** combines classical forecasting and deep learning, consisting of trend, seasonality, and residual components. The autoregressive component is disabled because it would require many invocations per day, which is not permitted in our use case.
(5) **N-HiTS** uses multi-rate signal sampling and hierarchical interpolation to learn patterns at different scales, and then recombine these components into an overall forecast. We use 3 stacks, 1 block per stack and 2 fully connected layers preceding the final forking layers in each block of every stack. Each layer has width 512.
(6) **TimesNet** uses a multi-scale approach that converts 1D series to 2D and then uses convolutional kernels. In experiments, we use 2 encoding layers, 1 decoding layer, model dimension of 64.
(7) **PatchTST** is an encoder-only transformer architecture that uses patching, where a series in split into subsequences before passing through the model, and channel independence, where different channels are predicted independently from each other, but sharing model weights while training. We use a patch length of 16, 2 encoding layers and model dimension of 64.
(8) **TiDE** is a multi-layer perceptron (MLP) based encoder-decoder model specifically designed for long-term time series forecasting.

Data Description. For training, we selected the top 10 functions by median requests. For Huawei Private, we train models on 38 days consisting of two chunks with a gap in between. We use the following 7 days for validation to implement early stopping (to mitigate overfitting) and test on the next 7 days. For Huawei Public, we train on the first 20 days, use the following 3 days for validation, and test on the next 3 days.

We study the performance of the eight models on per-minute and per-second data. We aggregate Huawei Private data per minute. For training a global model for per-minute forecasting, we use an input length of two days for Huawei Private and one day for Huawei Public. We set the forecast horizon to one day, i.e., we predict an entire day in advance. This is closest to how the model would be used in production. For training a global model for per-second forecasting (N-HiTS only) we use a 6-hour input length and a one-hour forecast horizon as most models fail with longer inputs.

5.2 Forecasting results

Table 1 shows the results of our forecasting models on Huawei Public, and Huawei Private at per-minute and per-second resolutions. We show root-mean-squared error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). We give results on the top 5 functions due to space constraints. In the table, we highlight the best performing model for each function's workload in boldface. In many cases, an FFT extrapolation performs better than more complex methods. Per-second data disables the use of many models since they cannot ingest enough data to meet our long-term requirement (i.e., forecast for a full day).

Figure 19 visualizes the results from Table 1. Figure 19a shows predictions for per-minute data, where most models make reasonable predictions of large-scale trends. However,

the zoomed-in view shows that models fail to accurately predict finer-grained patterns. Figure 19b shows predictions for per-second data. Again, models struggle with finer-grained patterns. For both per-minute and per-second forecasting, the zoomed-in plots in Figures 19a and 19b show that predictions deviate significantly from ground truth data for several hours. For per-second forecasting, many models run out of memory or require prohibitively long training, so only FFT and N-HiTS could be evaluated.

To better understand the scalability problems of complex prediction models, Figure 20 shows allocated memory of selected models with the number of input days during training. Note that N-HiTS scales best, while linear scales worst.

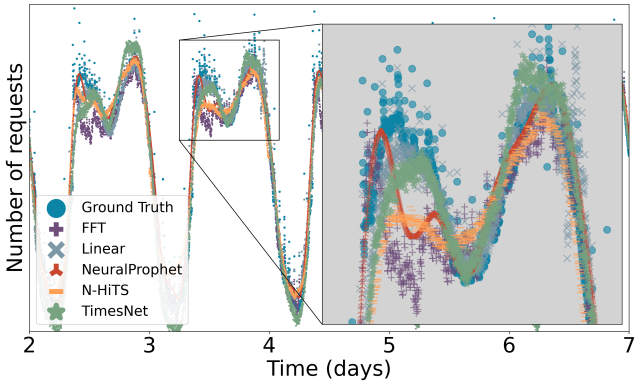
While it was not possible to train N-HiTS with the standard one-day-in-one-day-out setup due to memory and training time constraints, we train N-HiTS with a six-hour input length and one-hour output length as shown in Figure 19b. However, this increases the cost of predicting all functions in our data centers as we now invoke the model 24 times more often. The other model that works with our data is FFT. FFT fits large-scale trends, but does not predict finer-grained patterns. Importantly, Figure 19 shows all models are poorly suited to predicting peaks, especially for per-second data. More work is required to accurately predict peaks of bursty time series, given the importance of bursts in cloud platforms.

It is worth noting that many simple, non-neural network models perform comparably or sometimes better than state of the art neural networks. It appears that for FGLT forecasting, neural networks give marginal improvements over simpler methods. This is especially noteworthy given the overhead of training and inference of neural networks. We found N-HiTS to be one of the more robust and performant models. This may be because its architecture is specifically designed for learning patterns at different scales.

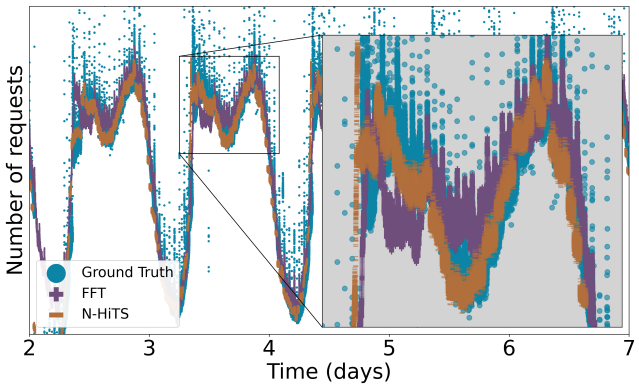
5.3 Per-second challenges

Fine-grained long-term forecasting is under-explored in literature. Even if a model could ingest such amounts of data with trends at multiple scales as shown in Figure 18, it would not necessarily learn the underlying trends. The memory overhead of consuming this much data means that even efficient models such as N-HiTS can only ingest approximately 18 days (see Figure 20).

The per-second results in Table 2 and Figure 19b further show the challenge with more fine-grained data. Therefore, we see the following challenges in fine-grained long-term time series forecasting model capabilities; (1) Ingesting long-term per-second data, where long-term means at least 1 week; (2) Generalizing well under the above requirement,



(a) Per-minute experiments.



(b) Per-second experiments.

Figure 19: Example function request predictions for different machine learning models.

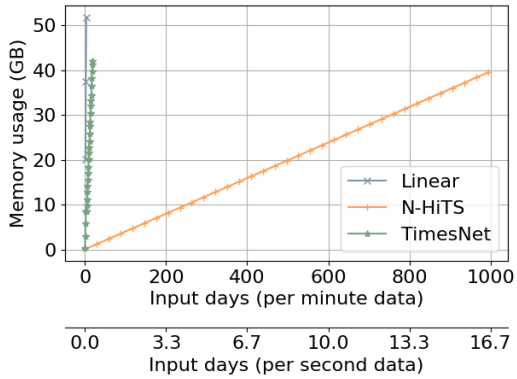


Figure 20: Memory consumption of selected models.

without severe overfitting; and (3) Forecasting (directly or autoregressively) long-term without severe degradation.

6 RELATED WORK

Workload characterization of server systems has been a popular research topic for decades [4, 10, 13]. Recently, there have been a few studies on characterization and analysis

Metric	Method	Pub 10	Pub 13	Pub 16	Pub 3	Pub 6	Pvt(m) 150	Pvt(m) 160	Pvt(m) 161	Pvt(m) 33	Pvt(m) 72	Pvt(s) 150	Pvt(s) 160	Pvt(s) 161	Pvt(s) 33	Pvt(s) 72
RMSE	FFT	**0.505**	0.447	0.452	**0.407**	0.778	0.184	0.257	0.270	0.737	0.248	0.226	0.287	0.285	0.656	**0.330**
	TimesNet	0.598	0.209	0.206	0.416	0.263	0.209	0.231	0.233	**0.252**	0.251					
	TiDE	0.704	0.400	0.401	0.740	0.342	0.615	0.703	0.715	0.820	0.697					
	PatchTST	0.521	0.216	0.216	0.423	0.307	0.232	0.222	0.217	0.356	0.250					
	N-Linear	0.674	0.245	0.245	0.662	0.246	0.455	0.435	0.411	0.907	0.393					
	Linear	0.547	0.221	0.223	0.543	0.214	**0.177**	**0.188**	**0.192**	0.423	**0.190**					
	N-HiTS	0.524	**0.196**	**0.200**	0.520	**0.213**	0.239	0.238	0.248	0.534	0.253	**0.168***	**0.188***	**0.199***	**0.240***	0.354*
	NeuralProphet	0.600	0.474	0.474	0.460	0.274	0.217	0.225	0.231	0.353	0.311					
MAPE	FFT	0.209	0.154	0.155	0.188	0.445	**0.087**	0.062	0.067	0.095	0.159	0.120	0.097	0.087	0.141	0.212
	TimesNet	0.209	0.071	0.070	**0.138**	0.219	0.101	0.067	0.070	**0.033**	0.204					
	TiDE	0.337	0.164	0.163	0.279	0.557	0.449	0.275	0.283	0.103	1.197					
	PatchTST	0.222	0.104	0.103	0.166	0.272	0.123	0.063	0.063	0.042	0.154					
	N-Linear	0.294	0.138	0.137	0.304	0.427	0.215	0.133	0.127	0.113	0.315					
	Linear	**0.206**	0.100	0.100	0.289	**0.207**	0.110	**0.053**	**0.056**	0.047	0.179					
	N-HiTS	0.212	**0.063**	**0.063**	0.186	0.455	0.123	0.072	0.077	0.066	**0.132**	**0.074***	**0.048***	**0.046***	**0.061***	**0.122***
	NeuralProphet	0.372	0.197	0.198	0.184	0.312	0.113	0.064	0.062	0.045	0.192					
MAE	FFT	**0.385**	0.332	0.337	0.325	0.487	**0.135**	0.175	0.182	0.645	0.169	0.152	0.185	0.184	0.526	0.168
	TimesNet	0.426	0.147	0.145	0.302	0.175	0.162	0.160	0.157	**0.205**	0.160					
	TiDE	0.533	0.309	0.309	0.556	0.274	0.468	0.569	0.563	0.646	0.543					
	PatchTST	0.387	0.164	0.164	**0.301**	0.207	0.178	0.161	0.155	0.267	0.153					
	N-Linear	0.490	0.203	0.204	0.488	0.195	0.378	0.337	0.320	0.732	0.322					
	Linear	0.388	0.176	0.178	0.411	**0.144**	**0.135**	**0.127**	**0.125**	0.292	**0.145**					
	N-HiTS	0.392	**0.126**	**0.128**	0.385	0.159	0.187	0.176	0.180	0.446	**0.145**	**0.100***	**0.106***	**0.108***	**0.163***	**0.137***
	NeuralProphet	0.483	0.356	0.360	0.362	0.183	0.163	0.166	0.159	0.304	0.217					

Table 2: Results for forecasting experiments. For Huawei Public (Pub) results are computed in a rolling manner over 2 days with input of 1 day. For Huawei Private per-minute – Pvt(m) – results are computed in a rolling manner over 5 days with an input of 2 days. For Huawei Private per-second – Pvt(s) – results are computed using a rolling forecast over 1 hour. The best results are highlighted in bold. All metrics are computed on normalized data. *For per-second forecasting, N-HiTS uses a six-hour input window one-hour output window as explained in the text.

of cloud system production workloads, e.g., from Google's Borg clusters [32], Azure's VM workloads [9], Alibaba ML workloads [38], and Azure Serverless workloads [21, 29, 43], among many others. The main goal of these studies is to design better systems, where the system management is data-driven. With the exception of the workloads analyzed from Azure on serverless systems [21, 29, 43], existing work on data center workload analysis focuses on CPU and memory utilization, VM types, and other machine and VM level metrics. To the best of our knowledge, the only large-scale publicly available FaaS dataset are the two available from Microsoft Azure [21, 29], which includes two weeks of function request data for tens of thousands of functions. Another dataset is also to be released from Azure that shows the graph structures of FaaS functions [43].

7 CONCLUSION

This paper describes two Huawei serverless traces from public and private infrastructure containing 1.4 trillion function invocations, which we open-source to the research community. It provides a comprehensive analysis covering statistical features of our workloads, followed by a longitudinal analysis of periodicity and ranking of functions across our traces.

Our analysis uncovered several interesting insights which can inform the engineering of future resource schedulers: requests vary by up to 9 orders of magnitude across functions, with some functions being executed over one billion times per day; scheduling time, execution time and cold-start distributions vary across 2 to 4 orders of magnitude and have very long tails; many functions demonstrate strong periodicity; and the highly ranked functions are occupied by only a small number of unique functions. Our analysis also highlights the need for further research and development in estimating resource reservations and time series prediction to account for the huge diversity in how serverless functions behave.

ACKNOWLEDGMENTS

We would like to thank the YuanRong lab for their valuable collaboration and collecting the data. We would also like to thank Wei Wei for his contributions and communication with YuanRong. We would like to thank Blesson Varghese for his helpful proofreading and edits. Lastly, we would like to thank the reviewers at ACM SoCC for their insightful comments and Yue Cheng for shepherding.

REFERENCES

[1] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications.. In *NSDI*, Vol. 20. 419–434.

[2] Ahmed Ali-Eldin, Oleg Seleznjev, Sara Sjöstedt-de Luna, Johan Tordsson, and Erik Elmroth. 2014. Measuring cloud workload burstiness. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 566–572.

[3] Apache OpenWhisk. 2016. Open Source Serverless Cloud Platform. https://openwhisk.apache.org/.

[4] Martin F Arlitt and Carey L Williamson. 1996. Web server workload characterization: The search for invariants. *ACM SIGMETRICS Performance Evaluation Review* 24, 1 (1996), 126–137.

[5] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*. 53–64.

[6] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, François-Xavier Aubet, Laurent Callot, and Tim Januschowski. 2022. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Comput. Surv.* 55, 6, Article 121 (dec 2022), 36 pages. https://doi.org/10.1145/3533382

[7] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: A Scalable and Locality-Enhanced Framework for Serverless Parallel Computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (Virtual Event, USA) *(SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3419111.3421286

[8] Cristian Challu, Kin G Olivares, Boris N Oreshkin, Federico Garza, Max Mergenthaler-Canseco, and Artur Dubrawski. 2022. N-hits: Neural hierarchical interpolation for time series forecasting. *arXiv preprint arXiv:2201.12886* (2022).

[9] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource Central: Understanding and Predicting Workloads for Improved Resource Management in Large Cloud Platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 153–167. https://doi.org/10.1145/3132747.3132772

[10] Mark E Crovella and Azer Bestavros. 1997. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on networking* 5, 6 (1997), 835–846.

[11] Abhimanyu Das, Weihao Kong, Andrew Leach, Shaan K Mathur, Rajat Sen, and Rose Yu. 2023. Long-term Forecasting with TiDE: Time-series Dense Encoder. *Transactions on Machine Learning Research* (2023). https://openreview.net/forum?id=pCbC3aQB5W

[12] Nilanjan Daw, Umesh Bellur, and Purushottam Kulkarni. 2020. Xanadu: Mitigating cascading cold starts in serverless function chain deployments. In *Proceedings of the 21st International Middleware Conference*. 356–370.

[13] Dror G Feitelson. 2015. *Workload modeling for computer systems performance evaluation*. Cambridge University Press.

[14] Joseph M. Hellerstein, Jose M. Faleiro, Joseph Gonzalez, Johann Schleier-Smith, Vikram Sreekanti, Alexey Tumanov, and Chenggang Wu. 2019. Serverless Computing: One Step Forward, Two Steps Back. In *9th Biennial Conference on Innovative Data Systems Research, CIDR 2019, Asilomar, CA, USA, January 13-16, 2019, Online Proceedings*. www.cidrdb.org. http://cidrdb.org/cidr2019/papers/p119-hellerstein-cidr19.pdf

[15] Julien Herzen, Francesco Lässig, Samuele Giuliano Piazzetta, Thomas Neuer, Léo Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, et al. 2022. Darts: User-friendly modern machine learning for time series. *The Journal of Machine Learning Research* 23, 1 (2022), 5442–5447.

[16] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. 2022. Global models for time series forecasting: A Simulation study. *Pattern Recognition* 124 (2022), 108441. https://doi.org/10.1016/j.patcog.2021.108441

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

[18] Knative. 2020. Knative Serverless Platform. https://knative.dev/docs/.

[19] "AWS Lambda". 2020. Coca-Cola Freestyle Launches Touchless Fountain Experience in 100 Days Using AWS Lambda. https://aws.amazon.com/solutions/case-studies/coca-cola-freestyle/.

[20] Zijun Li, Linsong Guo, Quan Chen, Jiagan Cheng, Chuhao Xu, Deze Zeng, Zhuo Song, Tao Ma, Yong Yang, Chao Li, et al. 2022. Help Rather Than Recycle: Alleviating Cold Startup in Serverless Computing Through Inter-Function Container Sharing. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 69–84.

[21] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Eshaan Minocha, Sameh Elnikety, Saurabh Bagchi, and Somali Chaterji. 2022. Wisefuse: Workload characterization and dag transformation for serverless workflows. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 2 (2022), 1–28.

[22] Frank San Miguel. 2021. The Netflix Cosmos Platform. https://netflixtechblog.com/the-netflix-cosmos-platform-35c14d9351ad.

[23] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. 2023. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. In *International Conference on Learning Representations*.

[24] Openshift. 2021. OpenShift Serverless overview. https://docs.openshift.com/serverless/1.29/about/about-serverless.html.

[25] Vladislav Petkov, Ram Rajagopal, and Katia Obraczka. 2013. Characterizing per-application network traffic using entropy. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 23, 2 (2013), 1–25.

[26] "Microsoft Research". 2019. Azure Public Dataset. https://github.com/Azure/AzurePublicDataset/tree/master.

[27] Joshua S Richman, Douglas E Lake, and J Randall Moorman. 2004. Sample entropy. In *Methods in enzymology*. Vol. 384. Elsevier, 172–184.

[28] J Riihijarvi, Matthias Wellens, and P Mahonen. 2009. Measuring complexity and predictability in networks with multiscale entropy analysis. In *IEEE INFOCOM 2009*. IEEE, 1107–1115.

[29] Mohammad Shahrad, Rodrigo Fonseca, Inigo 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 *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[30] Zhiming Shen, Qin Jia, Gur-Eyal Sela, Ben Rainero, Weijia Song, Robbert van Renesse, and Hakim Weatherspoon. 2016. Follow the sun through the clouds: Application migration for geographically shifting workloads. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 141–154.

[31] Shelby Thomas, Lixiang Ao, Geoffrey M Voelker, and George Porter. 2020. Particle: ephemeral endpoints for serverless networking. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 16–29.

[32] Muhammad Tirmazi, Adam Barker, Nan Deng, Md Ehtesam Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. 2020. Borg: the Next Generation. In *EuroSys'20*. Heraklion, Crete.

[33] Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Nikolay Laptev, Christoph Bergmeir, and Ram Rajagopal. 2021. Neuralprophet: Explainable forecasting at scale. *arXiv preprint arXiv:2111.15397* (2021).

[34] Raphael Vallat. 2018. AntroPy: entropy and complexity of (EEG) time-series in Python. https://github.com/raphaelvallat/antropy.

[35] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. 1–17.

[36] Ao Wang, Shuai Chang, Huangshi Tian, Hongqi Wang, Haoran Yang, Huiba Li, Rui Du, and Yue Cheng. 2021. Faasnet: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*.

[37] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2018. Peeking behind the curtains of serverless platforms. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 133–146.

[38] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 945–960.

[39] Haixu Wu, Tengge Hu, Yong Liu, Hang Zhou, Jianmin Wang, and Mingsheng Long. 2022. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis. *arXiv preprint arXiv:2210.02186* (2022).

[40] Hongliang Yu, Dongdong Zheng, Ben Y Zhao, and Weimin Zheng. 2006. Understanding user behavior in large-scale video-on-demand systems. *ACM SIGOPS Operating Systems Review* 40, 4 (2006), 333–344.

[41] Minchen Yu, Tingjia Cao, Wei Wang, and Ruichuan Chen. 2023. Following the data, not the function: Rethinking function orchestration in serverless computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1489–1504.

[42] Ailing Zeng, Muxi Chen, Lei Zhang, and Qiang Xu. 2022. Are transformers effective for time series forecasting? *arXiv preprint arXiv:2205.13504* (2022).

[43] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. 2021. Faster and cheaper serverless computing on harvested resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 724–739.